

Oracle9i

Database Administrator's Guide

Release 2 (9.2)

March 2002

Part No. A96521-01

ORACLE®

Oracle9i Database Administrator's Guide, Release 2 (9.2)

Part No. A96521-01

Copyright © 2001, 2002 Oracle Corporation. All rights reserved.

Primary Author: Ruth Baylis

Contributing Authors: Kathy Rich

Graphic Designer: Valarie Moore

Contributors: Lance Ashdown, Allen Brumm, Michele Cyran, Mary Ann Davidson, Harvey Eneman, Amit Ganesh, Carolyn Gray, Wei Huang, Robert Jenkins, Mark Kennedy, Sushil Kumar, Bill Lee, Yunrui Li, Diana Lorentz, Sujatha Muthulingam, Gary Ngai, Waleed Ojeil, Lois Price, Ananth Raghavan, Ann Rhee, Rajiv Sinha, Jags Srinivasan, Anh-Tuan Tran, Deborah Steiner, Janet Stern, Michael Stewart, Alex Tsukerman, Kothanda Umamageswaran, Steven Wertheimer, Daniel Wong

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, 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. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, 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 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

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

Oracle is a registered trademark, and Oracle9i, Oracle8i, Oracle8, Oracle7, Oracle Names, Oracle Store, Oracle Transparent Gateway, PL/SQL, SQL*Net, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments	xxxix
Preface.....	xxxiii
What's New in Oracle9i?	xlvi
Part I Basic Database Administration	
1 The Oracle Database Administrator	
Types of Oracle Users.....	1-2
Database Administrators.....	1-2
Security Officers.....	1-3
Network Administrators	1-3
Application Developers.....	1-3
Application Administrators.....	1-4
Database Users.....	1-4
Tasks of a Database Administrator	1-4
Task 1: Evaluate the Database Server Hardware.....	1-5
Task 2: Install the Oracle Software.....	1-5
Task 3: Plan the Database	1-5
Task 4: Create and Open the Database.....	1-6
Task 5: Back Up the Database	1-7
Task 6: Enroll System Users	1-7
Task 7: Implement the Database Design	1-7

Task 8: Back Up the Fully Functional Database	1-7
Task 9: Tune Database Performance	1-8
Identifying Your Oracle Database Software Release	1-8
Release Number Format	1-8
Checking Your Current Release Number	1-10
Database Administrator Security and Privileges	1-10
The Database Administrator's Operating System Account	1-10
Database Administrator Usernames	1-11
Database Administrator Authentication	1-13
Administrative Privileges	1-13
Selecting an Authentication Method	1-15
Using Operating System (OS) Authentication	1-17
Using Password File Authentication	1-18
Creating and Maintaining a Password File	1-20
Using ORAPWD	1-20
Setting REMOTE_LOGIN_PASSWORDFILE	1-22
Adding Users to a Password File	1-23
Maintaining a Password File	1-24
Database Administrator Utilities	1-26
SQL*Loader	1-26
Export and Import	1-26

2 Creating an Oracle Database

Considerations Before Creating a Database	2-2
Planning for Database Creation	2-2
Meeting Creation Prerequisites	2-4
Deciding How to Create an Oracle Database	2-4
Using the Database Configuration Assistant	2-5
Advantages of Using DBCA	2-6
Creating a Database Using DBCA	2-7
Configuring Database Options	2-9
Deleting a Database Using DBCA	2-9
Managing DBCA Templates	2-9
Using DBCA Silent Mode	2-13
Manually Creating an Oracle Database	2-14

Step 1: Decide on Your Instance Identifier (SID)	2-14
Step 2: Establish the Database Administrator Authentication Method.....	2-15
Step 3: Create the Initialization Parameter File	2-15
Step 4: Connect to the Instance	2-17
Step 5: Start the Instance	2-17
Step 6: Issue the CREATE DATABASE Statement	2-18
Step 7: Create Additional Tablespaces	2-20
Step 8: Run Scripts to Build Data Dictionary Views.....	2-21
Step 9: Run Scripts to Install Additional Options (Optional).....	2-21
Step 10: Create a Server Parameter File (Recommended)	2-21
Step 11: Back Up the Database.....	2-22
Understanding the CREATE DATABASE Statement	2-22
Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM.....	2-23
Clauses that Simplify Database Creation and Management.....	2-23
Creating a Locally Managed SYSTEM Tablespace	2-26
Specifying the Database Time Zone and Time Zone File	2-28
Specifying FORCE LOGGING Mode	2-29
Troubleshooting Database Creation	2-31
Dropping a Database	2-31
Considerations After Creating a Database	2-31
Some Security Considerations	2-32
Installing Oracle's Sample Schemas.....	2-33
Initialization Parameters and Database Creation.....	2-34
Determining the Global Database Name	2-35
Specifying Control Files.....	2-36
Specifying Database Block Sizes.....	2-37
Setting Initialization Parameters that Affect the Size of the SGA	2-38
Specifying the Maximum Number of Processes	2-41
Specifying the Method of Undo Space Management.....	2-42
Setting License Parameters.....	2-43
Managing Initialization Parameters Using a Server Parameter File	2-44
What is a Server Parameter File?.....	2-44
Migrating to a Server Parameter File.....	2-45
Creating a Server Parameter File.....	2-46
The SPFILE Initialization Parameter.....	2-47

Using ALTER SYSTEM to Change Initialization Parameter Values	2-47
Exporting the Server Parameter File	2-49
Backing Up the Server Parameter File	2-50
Errors and Recovery for the Server Parameter File	2-50
Viewing Parameter Settings	2-51

3 Using Oracle-Managed Files

What are Oracle-Managed Files?	3-2
Who Can Use Oracle-Managed Files?	3-2
Benefits of Using Oracle-Managed Files	3-3
Oracle-Managed Files and Existing Functionality	3-4
Enabling the Creation and Use of Oracle-Managed Files	3-4
Setting the DB_CREATE_FILE_DEST Initialization Parameter	3-5
Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameter	3-6
Creating Oracle-Managed Files	3-6
How Oracle-Managed Files are Named	3-7
Creating Oracle-Managed Files at Database Creation	3-8
Creating Datafiles for Tablespaces	3-14
Creating Tempfiles for Temporary Tablespaces	3-16
Creating Control Files	3-17
Creating Online Redo Log Files	3-19
Behavior of Oracle-Managed Files	3-21
Dropping Datafiles and Tempfiles	3-21
Dropping Online Redo Log Files	3-22
Renaming Files	3-22
Managing Standby Databases	3-22
Scenarios for Using Oracle-Managed Files	3-22
Scenario 1: Create and Manage a Database with Multiplexed Online Redo Logs	3-23
Scenario 2: Add Oracle-Managed Files to an Existing Database	3-27

4 Starting Up and Shutting Down

Starting Up a Database	4-2
Options for Starting Up a Database	4-2
Preparing to Start an Instance	4-3
Using SQL*Plus to Start Up a Database	4-3

Starting an Instance: Scenarios	4-5
Altering Database Availability	4-9
Mounting a Database to an Instance.....	4-9
Opening a Closed Database	4-10
Opening a Database in Read-Only Mode	4-10
Restricting Access to an Open Database	4-11
Shutting Down a Database	4-11
Shutting Down with the NORMAL Option	4-12
Shutting Down with the IMMEDIATE Option	4-12
Shutting Down with the TRANSACTIONAL Option	4-13
Shutting Down with the ABORT Option	4-13
Quiescing a Database	4-14
Placing a Database into a Quiesced State.....	4-15
Restoring the System to Normal Operation	4-16
Viewing the Quiesce State of an Instance	4-16
Suspending and Resuming a Database	4-17

Part II Oracle Server Processes and Storage Structure

5 Managing Oracle Processes

Server Processes	5-2
Dedicated Server Processes.....	5-2
Shared Server Processes.....	5-3
Configuring Oracle for the Shared Server	5-5
Initialization Parameters for Shared Server	5-5
Setting the Initial Number of Dispatchers (DISPATCHERS).....	5-6
Setting the Initial Number of Shared Servers (SHARED_SERVERS)	5-7
Modifying Dispatcher and Server Processes	5-8
Monitoring Shared Server	5-10
About Oracle Background Processes	5-11
Monitoring the Processes of an Oracle Instance	5-13
Process and Session Views	5-14
Monitoring Locks.....	5-15
Trace Files and the Alert File	5-15
Managing Processes for Parallel Execution	5-18

Managing the Parallel Execution Servers.....	5-18
Altering Parallel Execution for a Session	5-19
Managing Processes for External Procedures.....	5-20
Terminating Sessions	5-21
Identifying Which Session to Terminate	5-22
Terminating an Active Session	5-22
Terminating an Inactive Session.....	5-23

6 Managing Control Files

What Is a Control File?.....	6-2
Guidelines for Control Files	6-2
Provide Filenames for the Control Files	6-2
Multiplex Control Files on Different Disks.....	6-3
Place Control Files Appropriately.....	6-3
Back Up Control Files.....	6-3
Manage the Size of Control Files	6-4
Creating Control Files	6-4
Creating Initial Control Files.....	6-4
Creating Additional Copies, Renaming, and Relocating Control Files.....	6-5
Creating New Control Files.....	6-5
Troubleshooting After Creating Control Files	6-9
Checking for Missing or Extra Files	6-9
Handling Errors During CREATE CONTROLFILE.....	6-10
Backing Up Control Files	6-10
Recovering a Control File Using a Current Copy	6-10
Recovering from Control File Corruption Using a Control File Copy	6-10
Recovering from Permanent Media Failure Using a Control File Copy	6-11
Dropping Control Files.....	6-11
Displaying Control File Information.....	6-12

7 Managing the Online Redo Log

What Is the Online Redo Log?.....	7-2
Redo Threads.....	7-2
Online Redo Log Contents	7-2
How Oracle Writes to the Online Redo Log.....	7-3

Planning the Online Redo Log	7-5
Multiplexing Online Redo Log Files.....	7-5
Placing Online Redo Log Members on Different Disks.....	7-9
Setting the Size of Online Redo Log Members.....	7-9
Choosing the Number of Online Redo Log Files.....	7-10
Controlling Archive Lag.....	7-10
Creating Online Redo Log Groups and Members	7-12
Creating Online Redo Log Groups	7-13
Creating Online Redo Log Members	7-13
Relocating and Renaming Online Redo Log Members	7-14
Dropping Online Redo Log Groups and Members	7-16
Dropping Log Groups.....	7-16
Dropping Online Redo Log Members.....	7-17
Forcing Log Switches	7-18
Verifying Blocks in Redo Log Files	7-18
Clearing an Online Redo Log File	7-19
Viewing Online Redo Log Information	7-20

8 Managing Archived Redo Logs

What Is the Archived Redo Log?	8-2
Choosing Between NOARCHIVELOG and ARCHIVELOG Mode	8-2
Running a Database in NOARCHIVELOG Mode.....	8-2
Running a Database in ARCHIVELOG Mode	8-3
Controlling Archiving	8-4
Setting the Initial Database Archiving Mode	8-5
Changing the Database Archiving Mode.....	8-5
Enabling Automatic Archiving.....	8-6
Disabling Automatic Archiving	8-8
Performing Manual Archiving	8-9
Specifying the Archive Destination	8-9
Specifying Archive Destinations	8-10
Understanding Archive Destination Status.....	8-12
Specifying the Mode of Log Transmission	8-14
Normal Transmission Mode	8-14
Standby Transmission Mode	8-14

Managing Archive Destination Failure	8-16
Specifying the Minimum Number of Successful Destinations	8-16
Re-Archiving to a Failed Destination.....	8-18
Tuning Archive Performance by Specifying Multiple ARCn Processes	8-19
Controlling Trace Output Generated by the Archivelog Process	8-21
Viewing Information About the Archived Redo Log	8-23
Dynamic Performance Views.....	8-23
The ARCHIVE LOG LIST Command	8-24

9 Using LogMiner to Analyze Redo Logs

Potential Uses for Data Stored in Redo Logs	9-2
Accessing Information Stored in Redo Logs	9-3
Redo Logs and Dictionary Files	9-4
Redo Logs.....	9-4
Dictionary Options	9-5
Tracking DDL Statements.....	9-9
LogMiner Recommendations and Restrictions	9-10
Recommendations	9-10
Restrictions.....	9-11
Filtering Data That is Returned	9-12
Showing Only Committed Transactions	9-12
Skipping Redo Corruptions	9-14
Filtering Data By Time	9-14
Filtering Data By SCN.....	9-15
Accessing LogMiner Information	9-15
Querying VSLOGMNR_CONTENTS	9-16
Executing Reconstructed SQL Statements	9-17
Formatting of Returned Data.....	9-17
Extracting Actual Data Values from Redo Logs	9-18
NULL Returns From the MINE_VALUE Function	9-18
Usage Rules for the MINE_VALUE and COLUMN_PRESENT Functions	9-19
Supplemental Logging	9-19
Database Supplemental Logging.....	9-20
Table Supplemental Logging	9-22
Steps in a Typical LogMiner Session	9-23

Perform Initial Setup Activities	9-24
Extract a Dictionary	9-24
Specify Redo Logs for Analysis	9-24
Start a LogMiner Session	9-26
Query VSLOGMNR_CONTENTS	9-28
End a LogMiner Session	9-28
Example Uses of LogMiner	9-28
Example: Using LogMiner to Track Changes Made By a Specific User	9-28
Example: Using LogMiner to Calculate Table Access Statistics	9-30

10 Managing Job Queues

Enabling Processes Used for Executing Jobs	10-2
Managing Job Queues	10-3
The DBMS_JOB Package	10-3
Submitting a Job to the Job Queue	10-4
How Jobs Execute	10-9
Removing a Job from the Job Queue	10-10
Altering a Job	10-11
Broken Jobs	10-12
Forcing a Job to Execute	10-14
Terminating a Job	10-14
Viewing Job Queue Information	10-15
Displaying Information About a Job	10-15
Displaying Information About Running Jobs	10-15

11 Managing Tablespaces

Guidelines for Managing Tablespaces	11-2
Use Multiple Tablespaces	11-2
Specify Tablespace Default Storage Parameters	11-3
Assign Tablespace Quotas to Users	11-3
Creating Tablespaces	11-3
Locally Managed Tablespaces	11-5
Dictionary-Managed Tablespaces	11-10
Temporary Tablespaces	11-12
Coalescing Free Space in Dictionary-Managed Tablespaces	11-16

How Oracle Coalesces Free Space	11-16
Manually Coalescing Free Space	11-17
Monitoring Free Space	11-18
Specifying Nonstandard Block Sizes for Tablespaces	11-19
Controlling the Writing of Redo Records	11-20
Altering Tablespace Availability	11-21
Taking Tablespaces Offline	11-21
Bringing Tablespaces Online.....	11-23
Altering the Availability of Datafiles or Tempfiles.....	11-23
Using Read-Only Tablespaces	11-24
Making a Tablespace Read-Only	11-25
Making a Read-Only Tablespace Writable	11-27
Creating a Read-Only Tablespace on a WORM Device	11-27
Delaying the Opening of Datafiles in Read Only Tablespaces	11-28
Dropping Tablespaces	11-29
Diagnosing and Repairing Locally Managed Tablespace Problems	11-30
Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap)	11-32
Scenario 2: Dropping a Corrupted Segment	11-32
Scenario 3: Fixing Bitmap Where Overlap is Reported.....	11-32
Scenario 4: Correcting Media Corruption of Bitmap Blocks	11-33
Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace ..	11-33
Migrating the SYSTEM Tablespace to a Locally Managed Tablespace	11-34
Transporting Tablespaces Between Databases	11-34
Introduction to Transportable Tablespaces	11-35
Limitations.....	11-36
Compatibility Considerations for Transportable Tablespaces.....	11-36
Transporting Tablespaces Between Databases: A Procedure	11-37
Object Behaviors	11-43
Using Transportable Tablespaces.....	11-46
Viewing Tablespace Information	11-50
Listing Tablespaces and Default Storage Parameters: Example.....	11-51
Listing the Datafiles and Associated Tablespaces of a Database: Example	11-51
Displaying Statistics for Free Space (Extents) of Each Tablespace: Example.....	11-51

12 Managing Datafiles

Guidelines for Managing Datafiles	12-2
Determine the Number of Datafiles.....	12-2
Determine the Size of Datafiles	12-4
Place Datafiles Appropriately.....	12-4
Store Datafiles Separate from Redo Log Files	12-4
Creating Datafiles and Adding Datafiles to a Tablespace	12-5
Changing a Datafile's Size	12-6
Enabling and Disabling Automatic Extension for a Datafile	12-6
Manually Resizing a Datafile.....	12-7
Altering Datafile Availability	12-8
Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode	12-9
Taking Datafiles Offline in NOARCHIVELOG Mode	12-9
Altering the Availability of All Datafiles or Tempfiles in a Tablespace.....	12-9
Renaming and Relocating Datafiles	12-10
Renaming and Relocating Datafiles for a Single Tablespace	12-11
Renaming and Relocating Datafiles for Multiple Tablespaces	12-13
Dropping Datafiles	12-14
Verifying Data Blocks in Datafiles	12-14
Mapping Files to Physical Devices	12-15
Overview of Oracle's File Mapping Interface	12-16
How Oracle's File Mapping Interface Works.....	12-16
Using Oracle's File Mapping Interface.....	12-21
File Mapping Examples	12-25
Viewing Datafile Information	12-28

13 Managing Undo Space

What is Undo?	13-2
Specifying the Mode for Undo Space Management	13-3
Starting an Instance in Automatic Undo Management Mode	13-3
Starting an Instance in Manual Undo Management Mode	13-4
Managing Undo Tablespaces	13-5
Creating an Undo Tablespace.....	13-6
Altering an Undo Tablespace	13-7
Dropping an Undo Tablespace.....	13-7

Switching Undo Tablespaces	13-8
Establishing User Quotas for Undo Space	13-9
Specifying the Retention Period for Undo Information	13-9
Viewing Information About Undo Space.....	13-11
Managing Rollback Segments	13-13
Guidelines for Managing Rollback Segments	13-13
Creating Rollback Segments.....	13-19
Altering Rollback Segments	13-21
Explicitly Assigning a Transaction to a Rollback Segment	13-24
Dropping Rollback Segments.....	13-25
Viewing Rollback Segment Information	13-25

Part III Schema Objects

14 Managing Space for Schema Objects

Managing Space in Data Blocks	14-2
Specifying the PCTFREE Parameter	14-2
Specifying the PCTUSED Parameter	14-5
Selecting Associated PCTUSED and PCTFREE Values	14-7
Specifying the Transaction Entry Parameters: INITRANS and MAXTRANS.....	14-8
Setting Storage Parameters	14-8
Identifying the Storage Parameters.....	14-9
Setting Default Storage Parameters for Segments in a Tablespace	14-11
Setting Storage Parameters for Data Segments	14-11
Setting Storage Parameters for Index Segments.....	14-12
Setting Storage Parameters for LOBs, Varrays, and Nested Tables	14-12
Changing Values for Storage Parameters	14-13
Understanding Precedence in Storage Parameters.....	14-13
Example of How Storage Parameters Effect Space Allocation	14-14
Managing Resumable Space Allocation	14-14
Resumable Space Allocation Overview.....	14-15
Enabling and Disabling Resumable Space Allocation.....	14-19
Detecting Suspended Statements	14-21
Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger.....	14-23
Deallocating Space	14-25

Viewing the High Water Mark	14-25
Issuing Space Deallocation Statements	14-26
Examples of Deallocating Space	14-26
Understanding Space Use of Datatypes	14-29

15 Managing Tables

Guidelines for Managing Tables	15-2
Design Tables Before Creating Them	15-2
Specify How Data Block Space Is to Be Used	15-2
Specify the Location of Each Table	15-3
Consider Parallelizing Table Creation	15-4
Consider Using NOLOGGING When Creating Tables	15-4
Estimate Table Size and Set Storage Parameters	15-4
Plan for Large Tables	15-5
Table Restrictions	15-6
Creating Tables	15-6
Creating a Table	15-7
Creating a Temporary Table	15-8
Parallelizing Table Creation	15-8
Automatically Collecting Statistics on Tables	15-9
Altering Tables	15-10
Altering Physical Attributes of a Table	15-11
Moving a Table to a New Segment or Tablespace	15-12
Manually Allocating Storage for a Table	15-12
Modifying an Existing Column's Definition	15-13
Adding Table Columns	15-13
Renaming Table Columns	15-14
Dropping Table Columns	15-14
Redefining Tables Online	15-16
Features of Online Table Redefinition	15-16
The DBMS_REDEFINITION Package	15-17
Steps for Online Redefinition of Tables	15-17
Intermediate Synchronization	15-19
Abort and Cleanup After Errors	15-20
Example of Online Table Redefinition	15-20

Restrictions.....	15-22
Dropping Tables	15-23
Managing Index-Organized Tables	15-24
What are Index-Organized Tables.....	15-24
Creating Index-Organized Tables	15-25
Maintaining Index-Organized Tables.....	15-30
Analyzing Index-Organized Tables	15-32
Using the ORDER BY Clause with Index-Organized Tables	15-33
Converting Index-Organized Tables to Regular Tables.....	15-33
Managing External Tables	15-33
Creating External Tables.....	15-35
Altering External Tables	15-38
Dropping External Tables.....	15-39
System and Object Privileges for External Tables.....	15-39
Viewing Information About Tables	15-40

16 Managing Indexes

Guidelines for Managing Indexes	16-2
Create Indexes After Inserting Table Data.....	16-3
Index the Correct Tables and Columns	16-4
Order Index Columns for Performance.....	16-5
Limit the Number of Indexes for Each Table.....	16-5
Drop Indexes That Are No Longer Required	16-5
Specify Index Block Space Use.....	16-5
Estimate Index Size and Set Storage Parameters	16-6
Specify the Tablespace for Each Index.....	16-6
Consider Parallelizing Index Creation	16-7
Consider Creating Indexes with NOLOGGING	16-7
Consider Costs and Benefits of Coalescing or Rebuilding Indexes.....	16-8
Consider Cost Before Disabling or Dropping Constraints	16-9
Creating Indexes	16-9
Creating an Index Explicitly.....	16-10
Creating a Unique Index Explicitly.....	16-11
Creating an Index Associated with a Constraint	16-11
Collecting Incidental Statistics when Creating an Index	16-13

Creating a Large Index	16-13
Creating an Index Online	16-13
Creating a Function-Based Index	16-14
Creating a Key-Compressed Index	16-18
Altering Indexes	16-19
Altering Storage Characteristics of an Index	16-20
Rebuilding an Existing Index.....	16-20
Monitoring Index Usage.....	16-21
Monitoring Space Use of Indexes	16-21
Dropping Indexes	16-22
Viewing Index Information	16-23

17 Managing Partitioned Tables and Indexes

What Are Partitioned Tables and Indexes?	17-2
Partitioning Methods	17-3
When to Use the Range Partitioning Method.....	17-4
When to Use the Hash Partitioning Method	17-5
When to Use the List Partitioning Method	17-5
When to Use the Composite Range-Hash Partitioning Method.....	17-7
When to Use the Composite Range-List Partitioning Method	17-8
Creating Partitioned Tables	17-10
Creating Range-Partitioned Tables	17-11
Creating Hash-Partitioned Tables.....	17-12
Creating List-Partitioned Tables.....	17-13
Creating Composite Range-Hash Partitioned Tables	17-14
Creating Composite Range-List Partitioned Tables	17-15
Using Subpartition Templates to Describe Composite Partitioned Tables.....	17-17
Creating Partitioned Index-Organized Tables	17-19
Partitioning Restrictions for Multiple Block Sizes	17-22
Maintaining Partitioned Tables	17-22
Updating Global Indexes Automatically	17-26
Adding Partitions	17-27
Coalescing Partitions.....	17-31
Dropping Partitions.....	17-32
Exchanging Partitions	17-35

Merging Partitions	17-38
Modifying Default Attributes	17-43
Modifying Real Attributes of Partitions	17-44
Modifying List Partitions: Adding Values	17-45
Modifying List Partitions: Dropping Values	17-46
Modifying a Subpartition Template	17-48
Moving Partitions	17-48
Rebuilding Index Partitions	17-50
Renaming Partitions	17-51
Splitting Partitions	17-52
Truncating Partitions	17-59
Partitioned Tables and Indexes Examples	17-61
Moving the Time Window in a Historical Table	17-61
Converting a Partition View into a Partitioned Table	17-62
Viewing Information About Partitioned Tables and Indexes	17-64

18 Managing Clusters

Guidelines for Managing Clusters	18-2
Choose Appropriate Tables for the Cluster	18-4
Choose Appropriate Columns for the Cluster Key	18-4
Specify Data Block Space Use	18-5
Specify the Space Required by an Average Cluster Key and Its Associated Rows	18-5
Specify the Location of Each Cluster and Cluster Index Rows	18-6
Estimate Cluster Size and Set Storage Parameters	18-6
Creating Clusters	18-6
Creating Clustered Tables	18-7
Creating Cluster Indexes	18-8
Altering Clusters	18-8
Altering Clustered Tables	18-9
Altering Cluster Indexes	18-10
Dropping Clusters	18-10
Dropping Clustered Tables	18-11
Dropping Cluster Indexes	18-11
Viewing Information About Clusters	18-11

19 Managing Hash Clusters

When to Use Hash Clusters	19-2
Situations Where Hashing Is Useful	19-3
Situations Where Hashing Is Not Advantageous	19-3
Creating Hash Clusters	19-4
Creating Single-Table Hash Clusters	19-5
Controlling Space Use Within a Hash Cluster	19-5
Estimating Size Required by Hash Clusters	19-8
Altering Hash Clusters	19-9
Dropping Hash Clusters	19-9
Viewing Information About Hash Clusters	19-9

20 Managing Views, Sequences, and Synonyms

Managing Views	20-2
Creating Views	20-2
Updating a Join View	20-5
Altering Views	20-10
Dropping Views	20-10
Replacing Views	20-10
Managing Sequences	20-11
Creating Sequences	20-12
Altering Sequences	20-13
Dropping Sequences	20-13
Managing Synonyms	20-13
Creating Synonyms	20-14
Dropping Synonyms	20-14
Viewing Information About Views, Synonyms, and Sequences	20-15

21 General Management of Schema Objects

Creating Multiple Tables and Views in a Single Operation	21-2
Renaming Schema Objects	21-3
Analyzing Tables, Indexes, and Clusters	21-3
Collecting Statistics for Tables, Indexes, and Clusters	21-4
Validating Tables, Indexes, Clusters, and Materialized Views	21-6

Listing Chained Rows of Tables and Clusters.....	21-7
Truncating Tables and Clusters	21-9
Using DELETE	21-10
Using DROP and CREATE.....	21-10
Using TRUNCATE	21-10
Enabling and Disabling Triggers	21-11
Enabling Triggers.....	21-13
Disabling Triggers	21-13
Managing Integrity Constraints	21-14
Integrity Constraint States.....	21-15
Setting Integrity Constraints Upon Definition	21-17
Modifying, Renaming, or Dropping Existing Integrity Constraints.....	21-18
Deferring Constraint Checks.....	21-20
Reporting Constraint Exceptions.....	21-21
Viewing Constraint Information	21-23
Managing Object Dependencies	21-23
Manually Recompiling Views.....	21-25
Manually Recompiling Procedures and Functions.....	21-25
Manually Recompiling Packages	21-25
Managing Object Name Resolution	21-25
Changing Storage Parameters for the Data Dictionary	21-27
Structures in the Data Dictionary.....	21-28
Errors that Require Changing Data Dictionary Storage	21-30
Displaying Information About Schema Objects	21-30
Using PL/SQL Packages to Display Information About Schema Objects	21-30
Using Views to Display Information About Schema Objects	21-32

22 Detecting and Repairing Data Block Corruption

Options for Repairing Data Block Corruption	22-2
About the DBMS_REPAIR Package	22-2
DBMS_REPAIR Procedures	22-2
Limitations and Restrictions	22-3
Using the DBMS_REPAIR Package	22-3
Task 1: Detect and Report Corruptions	22-4
Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR	22-5

Task 3: Make Objects Usable.....	22-7
Task 4: Repair Corruptions and Rebuild Lost Data.....	22-7
DBMS_REPAIR Examples	22-8
Using ADMIN_TABLES to Build a Repair Table or Orphan Key Table	22-9
Using the CHECK_OBJECT Procedure to Detect Corruption	22-10
Fixing Corrupt Blocks with the FIX_CORRUPT_BLOCKS Procedure	22-12
Finding Index Entries Pointing into Corrupt Data Blocks: DUMP_ORPHAN_KEYS...	22-13
Rebuilding Free Lists Using the REBUILD_FREELISTS Procedure	22-13
Enabling or Disabling the Skipping of Corrupt Blocks: SKIP_CORRUPT_BLOCKS.....	22-14

Part IV Database Security

23 Establishing Security Policies

System Security Policy	23-2
Database User Management	23-2
User Authentication	23-2
Operating System Security	23-3
Data Security Policy	23-3
User Security Policy	23-4
General User Security	23-4
End-User Security.....	23-6
Administrator Security	23-8
Application Developer Security	23-10
Application Administrator Security	23-12
Password Management Policy	23-12
Account Locking.....	23-13
Password Aging and Expiration	23-14
Password History	23-15
Password Complexity Verification	23-16
Auditing Policy	23-20
A Security Checklist	23-20

24 Managing Users and Resources

Managing Oracle Users	24-2
------------------------------------	------

Creating Users	24-2
Altering Users.....	24-6
Dropping Users	24-7
User Authentication Methods	24-8
Database Authentication	24-9
External Authentication.....	24-11
Global Authentication and Authorization	24-13
Proxy Authentication and Authorization	24-16
Managing Resources with Profiles	24-18
Enabling and Disabling Resource Limits	24-19
Creating Profiles	24-20
Assigning Profiles.....	24-20
Altering Profiles	24-21
Using Composite Limits	24-21
Dropping Profiles	24-23
Viewing Information About Database Users and Profiles	24-23
Listing All Users and Associated Information	24-25
Listing All Tablespace Quotas	24-25
Listing All Profiles and Assigned Limits	24-25
Viewing Memory Use for Each User Session	24-26

25 Managing User Privileges and Roles

Understanding User Privileges and Roles	25-2
System Privileges	25-2
Object Privileges.....	25-4
User Roles	25-5
Managing User Roles	25-6
Creating a Role	25-7
Specifying the Type of Role Authorization.....	25-8
Dropping Roles	25-10
Granting User Privileges and Roles	25-11
Granting System Privileges and Roles.....	25-11
Granting Object Privileges.....	25-12
Revoking User Privileges and Roles	25-16
Revoking System Privileges and Roles.....	25-16

Revoking Object Privileges	25-16
Cascading Effects of Revoking Privileges.....	25-19
Granting to and Revoking from the User Group PUBLIC	25-20
When Do Grants and Revokes Take Effect?	25-20
The SET ROLE Statement	25-21
Specifying Default Roles.....	25-21
Restricting the Number of Roles that a User Can Enable.....	25-22
Granting Roles Using the Operating System or Network	25-22
Using Operating System Role Identification	25-23
Using Operating System Role Management	25-25
Granting and Revoking Roles When OS_ROLES=TRUE	25-25
Enabling and Disabling Roles When OS_ROLES=TRUE.....	25-25
Using Network Connections with Operating System Role Management	25-25
Viewing Privilege and Role Information.....	25-26
Listing All System Privilege Grants.....	25-27
Listing All Role Grants	25-28
Listing Object Privileges Granted to a User.....	25-28
Listing the Current Privilege Domain of Your Session.....	25-29
Listing Roles of the Database.....	25-30
Listing Information About the Privilege Domains of Roles.....	25-30

26 Auditing Database Use

Guidelines for Auditing.....	26-2
Decide Whether to Use the Database or Operating System Audit Trail	26-2
Keep Audited Information Manageable	26-3
Guidelines for Auditing Suspicious Database Activity	26-3
Guidelines for Auditing Normal Database Activity	26-4
What Information is Contained in the Audit Trail?.....	26-4
Information Stored in the Database Audit Trail	26-4
Information Stored in an Operating System File	26-5
Actions Audited by Default.....	26-6
Auditing Administrative Users.....	26-6
Managing the Audit Trail.....	26-7
Enabling and Disabling Auditing	26-8
Setting Auditing Options	26-9

Auditing in a Multi-Tier Environment	26-13
Turning Off Audit Options	26-13
Controlling the Growth and Size of the Audit Trail.....	26-15
Protecting the Audit Trail.....	26-18
Fine-Grained Auditing	26-18
Viewing Database Audit Trail Information	26-19
Creating the Audit Trail Views.....	26-19
Deleting the Audit Trail Views.....	26-20
Using Audit Trail Views to Investigate Suspicious Activities	26-20

Part V Database Resource Management

27 Using the Database Resource Manager

What Is the Database Resource Manager?	27-2
What Problems Does the Database Resource Manager Address?.....	27-2
How Does the Database Resource Manager Address These Problems?	27-2
What are the Elements of the Database Resource Manager?	27-3
Understanding Resource Plans.....	27-4
Administering the Database Resource Manager	27-8
Creating a Simple Resource Plan	27-10
Creating Complex Resource Plans	27-11
Using the Pending Area for Creating Plan Schemas	27-12
Creating Resource Plans	27-14
Creating Resource Consumer Groups.....	27-16
Specifying Resource Plan Directives.....	27-17
Managing Resource Consumer Groups	27-20
Assigning an Initial Resource Consumer Group	27-21
Changing Resource Consumer Groups.....	27-21
Managing the Switch Privilege.....	27-22
Enabling the Database Resource Manager	27-24
Putting It All Together: Database Resource Manager Examples	27-25
Multilevel Schema Example.....	27-25
Example of Using Several Resource Allocation Methods.....	27-27
An Oracle Supplied Plan	27-28
Monitoring and Tuning the Database Resource Manager	27-29

Creating the Environment.....	27-29
Why Is This Necessary to Produce Expected Results?.....	27-30
Monitoring Results.....	27-31
Viewing Database Resource Manager Information.....	27-31
Viewing Consumer Groups Granted to Users or Roles.....	27-32
Viewing Plan Schema Information.....	27-33
Viewing Current Consumer Groups for Sessions.....	27-33
Viewing the Currently Active Plans.....	27-34

Part VI Distributed Database Management

28 Distributed Database Concepts

Distributed Database Architecture	28-2
Homogenous Distributed Database Systems.....	28-2
Heterogeneous Distributed Database Systems.....	28-5
Client/Server Database Architecture.....	28-6
Database Links.....	28-8
What Are Database Links?.....	28-8
What Are Shared Database Links?.....	28-10
Why Use Database Links?.....	28-11
Global Database Names in Database Links.....	28-12
Names for Database Links.....	28-14
Types of Database Links.....	28-15
Users of Database Links.....	28-16
Creation of Database Links: Examples.....	28-19
Schema Objects and Database Links.....	28-20
Database Link Restrictions.....	28-22
Distributed Database Administration.....	28-23
Site Autonomy.....	28-24
Distributed Database Security.....	28-24
Auditing Database Links.....	28-31
Administration Tools.....	28-31
Transaction Processing in a Distributed System.....	28-33
Remote SQL Statements.....	28-33
Distributed SQL Statements.....	28-34

Shared SQL for Remote and Distributed Statements	28-34
Remote Transactions	28-35
Distributed Transactions.....	28-35
Two-Phase Commit Mechanism.....	28-35
Database Link Name Resolution	28-36
Schema Object Name Resolution.....	28-38
Global Name Resolution in Views, Synonyms, and Procedures	28-42
Distributed Database Application Development.....	28-44
Transparency in a Distributed Database System	28-44
Remote Procedure Calls (RPCs)	28-46
Distributed Query Optimization	28-47
Character Set Support for Distributed Environments.....	28-47
Client/Server Environment	28-48
Homogeneous Distributed Environment.....	28-48
Heterogeneous Distributed Environment.....	28-49

29 Managing a Distributed Database

Managing Global Names in a Distributed System	29-2
Understanding How Global Database Names Are Formed.....	29-2
Determining Whether Global Naming Is Enforced	29-3
Viewing a Global Database Name.....	29-4
Changing the Domain in a Global Database Name.....	29-4
Changing a Global Database Name: Scenario	29-5
Creating Database Links	29-8
Obtaining Privileges Necessary for Creating Database Links	29-8
Specifying Link Types.....	29-9
Specifying Link Users.....	29-11
Using Connection Qualifiers to Specify Service Names Within Link Names	29-13
Creating Shared Database Links.....	29-14
Determining Whether to Use Shared Database Links.....	29-14
Creating Shared Database Links.....	29-15
Configuring Shared Database Links	29-16
Managing Database Links.....	29-18
Closing Database Links.....	29-19
Dropping Database Links.....	29-19

Limiting the Number of Active Database Link Connections.....	29-20
Viewing Information About Database Links.....	29-21
Determining Which Links Are in the Database	29-21
Determining Which Link Connections Are Open	29-24
Creating Location Transparency	29-26
Using Views to Create Location Transparency	29-26
Using Synonyms to Create Location Transparency	29-28
Using Procedures to Create Location Transparency	29-30
Managing Statement Transparency.....	29-32
Managing a Distributed Database: Scenarios.....	29-34
Creating a Public Fixed User Database Link	29-34
Creating a Public Fixed User Shared Database Link.....	29-35
Creating a Public Connected User Database Link.....	29-35
Creating a Public Connected User Shared Database Link.....	29-36
Creating a Public Current User Database Link.....	29-37
30 Developing Applications for a Distributed Database System	
Managing the Distribution of an Application's Data.....	30-2
Controlling Connections Established by Database Links.....	30-2
Maintaining Referential Integrity in a Distributed System.....	30-3
Tuning Distributed Queries	30-3
Using Collocated Inline Views	30-4
Using Cost-Based Optimization	30-5
Using Hints.....	30-8
Analyzing the Execution Plan.....	30-9
Handling Errors in Remote Procedures.....	30-11
31 Distributed Transactions Concepts	
What Are Distributed Transactions?.....	31-2
Session Trees for Distributed Transactions	31-4
Clients.....	31-5
Database Servers.....	31-5
Local Coordinators	31-6
Global Coordinator.....	31-6
Commit Point Site.....	31-6

Two-Phase Commit Mechanism	31-10
Prepare Phase	31-11
Commit Phase	31-14
Forget Phase.....	31-15
In-Doubt Transactions	31-15
Automatic Resolution of In-Doubt Transactions	31-16
Manual Resolution of In-Doubt Transactions	31-18
Relevance of System Change Numbers for In-Doubt Transactions	31-19
Distributed Transaction Processing: Case Study	31-19
Stage 1: Client Application Issues DML Statements.....	31-19
Stage 2: Oracle Determines Commit Point Site	31-21
Stage 3: Global Coordinator Sends Prepare Response	31-21
Stage 4: Commit Point Site Commits	31-23
Stage 5: Commit Point Site Informs Global Coordinator of Commit.....	31-23
Stage 6: Global and Local Coordinators Tell All Nodes to Commit	31-24
Stage 7: Global Coordinator and Commit Point Site Complete the Commit.....	31-24

32 Managing Distributed Transactions

Specifying the Commit Point Strength of a Node	32-2
Naming Transactions	32-2
Viewing Information About Distributed Transactions	32-3
Determining the ID Number and Status of Prepared Transactions	32-3
Tracing the Session Tree of In-Doubt Transactions	32-5
Deciding How to Handle In-Doubt Transactions	32-7
Discovering Problems with a Two-Phase Commit	32-8
Determining Whether to Perform a Manual Override.....	32-9
Analyzing the Transaction Data	32-9
Manually Overriding In-Doubt Transactions	32-10
Manually Committing an In-Doubt Transaction	32-11
Manually Rolling Back an In-Doubt Transaction.....	32-12
Purging Pending Rows from the Data Dictionary	32-13
Executing the PURGE_LOST_DB_ENTRY Procedure	32-13
Determining When to Use DBMS_TRANSACTION.....	32-14
Manually Committing an In-Doubt Transaction: Example	32-15
Step 1: Record User Feedback	32-16

Step 2: Query DBA_2PC_PENDING	32-16
Step 3: Query DBA_2PC_NEIGHBORS on Local Node	32-18
Step 4: Querying Data Dictionary Views on All Nodes.....	32-19
Step 5: Commit the In-Doubt Transaction	32-22
Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING	32-22
Data Access Failures Due To Locks	32-23
Transaction Timeouts.....	32-23
Locks from In-Doubt Transactions	32-24
Simulating Distributed Transaction Failure	32-24
Managing Read Consistency	32-25

Index

Send Us Your Comments

Oracle9i Database Administrator's Guide, Release 2 (9.2)

Part No. A96521-01

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

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

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

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
Oracle Corporation
Server Technologies Documentation
500 Oracle Parkway, Mailstop 4op11
Redwood Shores, CA 94065
USA

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

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

Preface

This guide is for people who administer the operation of an Oracle database system. Referred to as database administrators (DBAs), they are responsible for creating Oracle databases, ensuring their smooth operation, and monitoring their use.

This preface contains these topics:

- [Audience](#)
- [Organization](#)
- [Related Documentation](#)
- [Conventions](#)
- [Documentation Accessibility](#)

Note: The *Oracle9i Database Administrator's Guide* contains information that describes the features and functionality of the Oracle9i [Standard Edition], Oracle9i Enterprise Edition, and Oracle9i Personal Edition products. These products have the same basic features. However, several advanced features are available only with the Oracle9i Enterprise Edition or Oracle9i Personal Edition, and some of these are optional. For example, to create partitioned tables and indexes, you must have the Oracle9i Enterprise Edition or Oracle9i Personal Edition.

For information about the differences between the various editions of Oracle9i and the features and options that are available to you, please refer to *Oracle9i Database New Features*.

Audience

Readers of this guide are assumed to be familiar with relational database concepts. They are also assumed to be familiar with the operating system environment under which they are running Oracle.

Readers Interested in Installation and Upgrade Information

Administrators frequently participate in installing the Oracle server software and upgrading existing Oracle databases to newer formats (for example, version 8 databases to Oracle9i format). This guide is not an installation or upgrade manual.

If your primary interest is installation, see your operating system specific Oracle installation guide.

If your primary interest is upgrading a database or application, see the *Oracle9i Database Migration* manual.

Readers Interested in Application Design Information

In addition to administrators, experienced users of Oracle and advanced database application designers might also find information in this guide useful.

However, database application developers should also see the *Oracle9i Application Developer's Guide - Fundamentals* and the documentation for the tool or language product they are using to develop Oracle database applications.

Organization

This document contains:

Part I: Basic Database Administration

Chapter 1, "The Oracle Database Administrator"

This chapter serves as a general introduction to typical tasks performed by database administrators, such as installing software and planning a database.

Chapter 2, "Creating an Oracle Database"

This chapter discusses considerations for creating a database and takes you through the steps of creating one. Consult this chapter when in the database planning and creation stage.

Chapter 3, "Using Oracle-Managed Files"

This chapter describes how you can direct the Oracle database server to create and manage your:

- Datafiles
- Tempfiles
- Online redo log files
- Control files

Chapter 4, "Starting Up and Shutting Down"

Consult this chapter when you wish to start up a database, alter its availability, or shut it down. Parameter files related to starting up and shutting down are also described here.

Part II: Oracle Server Processes and Storage Structure

Chapter 5, "Managing Oracle Processes"

This chapter helps you to identify different Oracle processes, such as dedicated server processes and shared server processes. Consult this chapter when configuring, modifying, tracking and managing processes.

Chapter 6, "Managing Control Files"

This chapter describes all aspects of managing control files: naming, creating, troubleshooting, and dropping control files.

Chapter 7, "Managing the Online Redo Log"

This chapter describes all aspects of managing the online redo log: planning, creating, renaming, dropping, or clearing online redo log files.

Chapter 8, "Managing Archived Redo Logs"

Consult this chapter for information about archive modes and tuning archiving.

Chapter 9, "Using LogMiner to Analyze Redo Logs"

This chapter describes the use of LogMiner to analyze redo log files.

Chapter 10, "Managing Job Queues"

Consult this chapter before working with job queues. All aspects of submitting, removing, altering, and fixing job queues are described.

Chapter 11, "Managing Tablespaces"

This chapter provides guidelines to follow as you manage tablespaces, and describes how to create, manage, alter, drop and move data between tablespaces.

Chapter 12, "Managing Datafiles"

This chapter provides guidelines to follow as you manage datafiles, and describes how to create, change, alter, rename and view information about datafiles.

Chapter 13, "Managing Undo Space"

Consult this chapter to learn how to manage undo space, either by using an undo tablespace or rollback segments.

Part III: Schema Objects

Chapter 14, "Managing Space for Schema Objects"

Consult this chapter for descriptions of common tasks, such as setting storage parameters, deallocating space and managing space.

Chapter 15, "Managing Tables"

Consult this chapter for general table management guidelines, as well as information about creating, altering, maintaining and dropping tables.

Chapter 16, "Managing Indexes"

Consult this chapter for general guidelines about indexes, including creating, altering, monitoring and dropping indexes.

Chapter 17, "Managing Partitioned Tables and Indexes"

Consult this chapter to learn about partitioned tables and indexes and how to create and manage them.

Chapter 18, "Managing Clusters"

Consult this chapter for general guidelines to follow when creating, altering, or dropping clusters.

Chapter 19, "Managing Hash Clusters"

Consult this chapter for general guidelines to follow when creating, altering, or dropping hash clusters.

Chapter 20, "Managing Views, Sequences, and Synonyms"

This chapter describes all aspects of managing views, sequences and synonyms.

Chapter 21, "General Management of Schema Objects"

This chapter covers more varied aspects of schema management. The operations described in this chapter are not unique to any one type of schema objects. Consult this chapter for information about analyzing objects, truncation of tables and clusters, database triggers, integrity constraints, and object dependencies.

Chapter 22, "Detecting and Repairing Data Block Corruption"

This chapter describes methods for detecting and repairing data block corruption.

Part IV: Database Security

Chapter 23, "Establishing Security Policies"

This chapter describes all aspects of database security, including system, data and user security policies, as well as specific tasks associated with password management.

Chapter 24, "Managing Users and Resources"

This chapter describes session and user licensing, user authentication, and provides specific examples of tasks associated with managing users and resources.

Chapter 25, "Managing User Privileges and Roles"

This chapter contains information about all aspects of managing user privileges and roles. Consult this chapter to find out how to grant and revoke privileges and roles.

Chapter 26, "Auditing Database Use"

This chapter describes how to create, manage and view audit information.

Part V: Database Resource Management

Chapter 27, "Using the Database Resource Manager"

This chapter describes how to use the Database Resource Manager to allocate resources.

Part VI: Distributed Database Management

Chapter 28, "Distributed Database Concepts"

This chapter describes the basic concepts and terminology of Oracle's distributed database architecture.

Chapter 29, "Managing a Distributed Database"

This chapter describes how to manage and maintain a distributed database system.

Chapter 30, "Developing Applications for a Distributed Database System"

This chapter describes considerations important when developing an application to run in a distributed database system.

Chapter 31, "Distributed Transactions Concepts"

This chapter describes what distributed transactions are and how Oracle maintains their integrity.

Chapter 32, "Managing Distributed Transactions"

This chapter describes how to manage and troubleshoot distributed transactions.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Database Concepts*

Chapter 1 of *Oracle9i Database Concepts* contains an overview of the concepts and terminology related to Oracle and provides a foundation for the more detailed information in this guide. This chapter is a starting point to become familiar with the Oracle database server, and is recommended reading before starting *Oracle9i Database Administrator's Guide*. The remainder of *Oracle9i Database Concepts* explains the Oracle architecture and features, and how they operate in more detail.

- *Oracle9i Backup and Recovery Concepts*
This book introduces you to the concepts of backup and recovery.
- *Oracle9i User-Managed Backup and Recovery Guide*
This guide contains details of backup and recovery and enables you back up, copy, restore, and recover datafiles, control files, and archived redo logs.
- *Oracle9i Recovery Manager User's Guide*
This guide contains information for using Recovery Manager (RMAN). RMAN is an Oracle tool that manages and automates backup and recovery operations.
- *Oracle9i Database Performance Planning*
This book exposes important considerations in setting up a database system and can help you understand tuning your database. It is mainly conceptual, defining terms, architecture, and design principles, and then outlines proactive and reactive tuning methods.
- *Oracle9i Database Performance Tuning Guide and Reference*
This book can be used as a reference guide for tuning your Oracle database system.
- *Oracle9i Application Developer's Guide - Fundamentals*
Many of the tasks done by DBAs are shared by application developers. In some cases, descriptions of tasks seemed better located in an application level book, and in those cases, this fundamentals book is the primary reference.

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

In North America, printed documentation is available for sale in the Oracle Store at <http://oraclestore.oracle.com/>

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

<http://www.oraclebookshop.com/>

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

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

<http://otn.oracle.com/admin/account/membership.html>

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

<http://otn.oracle.com/docs/index.htm>

To access the database documentation search engine directly, please visit

<http://tahiti.oracle.com>

Conventions

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

- [Conventions in Text](#)
- [Conventions in Code Examples](#)
- [Conventions for Windows Operating Systems](#)

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an index-organized table .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> reside on the same disk.

Convention	Meaning	Example
UPPERCASE monospace (fixed-width) font	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.
lowercase monospace (fixed-width) font	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter sqlplus to open SQL*Plus. The password is specified in the orapwd file. Back up the datafiles and control files in the /disk1/oracle/dbs directory. The department_id, department_name, and location_id columns are in the hr.departments table. Set the QUERY_REWRITE_ENABLED initialization parameter to true. Connect as oe user. The JRepuTil class implements these methods.
lowercase italic monospace (fixed-width) font	Lowercase italic monospace font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <i>Uold_release</i> .SQL where <i>old_release</i> refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[]	Brackets enclose one or more optional items. Do not enter the brackets.	DECIMAL (<i>digits</i> [, <i>precision</i>])
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	{ENABLE DISABLE}
	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	{ENABLE DISABLE} [COMPRESS NOCOMPRESS]
...	Horizontal ellipsis points indicate either: <ul style="list-style-type: none"> That we have omitted parts of the code that are not directly related to the example That you can repeat a portion of the code 	CREATE TABLE ... AS <i>subquery</i> ; SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> FROM employees;
.	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	SQL> SELECT NAME FROM V\$DATAFILE; NAME ----- /fsl/dbs/tbs_01.dbf /fsl/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/ <i>system_password</i> DB_NAME = <i>database_name</i>
UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;

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

Conventions for Windows Operating Systems

The following table describes conventions for Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Database Configuration Assistant, choose Start > Programs > Oracle - <i>HOME_NAME</i> > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	<p>File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.</p>	<pre>c:\winnt\"\"system32 is the same as C:\WINNT\SYSTEM32</pre>
C:\>	<p>Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the <i>command prompt</i> in this manual.</p>	<pre>C:\oracle\oradata></pre>

Convention	Meaning	Example
Special characters	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (') do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	<pre>C:\>exp scott/tiger TABLES=emp QUERY=\ "WHERE job='SALESMAN' and sal<1600\" C:\>imp SYSTEM/password FROMUSER=scott TABLES=(emp, dept)</pre>
<i>HOME_NAME</i>	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	<pre>C:\> net start OracleHOME_ NAMETNSListener</pre>

Convention	Meaning	Example
<i>ORACLE_HOME</i> and <i>ORACLE_BASE</i>	<p>In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level <i>ORACLE_HOME</i> directory that by default used one of the following names:</p> <ul style="list-style-type: none"> ■ C:\orant for Windows NT ■ C:\orawin98 for Windows 98 <p>This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level <i>ORACLE_HOME</i> directory. There is a top level directory called <i>ORACLE_BASE</i> that by default is C:\oracle. If you install the latest Oracle release on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is C:\oracle\orann, where <i>nn</i> is the latest release number. The Oracle home directory is located directly under <i>ORACLE_BASE</i>.</p> <p>All directory path examples in this guide follow OFA conventions.</p> <p>Refer to <i>Oracle9i Database Getting Started for Windows</i> for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.</p>	Go to the <i>ORACLE_BASE\ORACLE_HOME\rdms\admin</i> directory.

Documentation Accessibility

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

<http://www.oracle.com/accessibility/>

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

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

What's New in Oracle9i?

This section introduces new administrative features of Oracle9i Release 2 (9.2) that are discussed in this book and provides pointers to additional information.

For a summary of all new features for Oracle9i, see *Oracle9i Database New Features*.

The following section describes the new features discussed in the *Oracle9i Database Administrator's Guide*.

- [Oracle9i Release 2 \(9.2\) New Features](#)
- [Oracle9i Release 1 \(9.0.1\) New Features](#)

Oracle9i Release 2 (9.2) New Features

Oracle9i, Release 2, further advances and refines the goals achieved by Oracle9i, Release 1.

The following are summaries of the new features of Oracle9i, Release 2, that are discussed in this book.

- **Specifying Passwords for SYS and SYSTEM at Database Creation**

Oracle enables you to specify passwords for users SYS and SYSTEM using the following CREATE DATABASE clauses:

- USER SYS IDENTIFIED BY *password*
- USER SYSTEM IDENTIFIED BY *password*

See Also: ["Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM"](#) on page 2-23

- **Specifying FORCE LOGGING Mode**

The FORCE LOGGING clause of the CREATE DATABASE, CREATE CONTROLFILE, and CREATE TABLESPACE statement enables you to force redo log records to be written even when NOLOGGING has been specified in a DDL statement.

See Also:

- ["Specifying FORCE LOGGING Mode"](#) on page 2-29
- ["The CREATE CONTROLFILE Statement"](#) on page 6-6
- ["Controlling the Writing of Redo Records"](#) on page 11-20

- **Recovery Manager (RMAN) Backup of a Server Parameter File**

RMAN can now be used to create backups of a server parameter file.

See Also: ["Backing Up the Server Parameter File"](#) on page 2-50

- **New In LogMiner for Release 9.2**

LogMiner release 9.2 has added support for several new features and changed some default behavior as follows:

- LONG and LOB datatypes are supported for redo logs generated on a release 9.2 or later Oracle database.
- Supplemental logging is turned *off* by default. This is a change from release 9.0.1, in which minimal supplemental logging was turned on by default. In release 9.2, the level of supplemental logging desired must be specified.
- Different levels of supplemental logging are available: database supplemental logging and table supplemental logging. Within database supplemental logging, you have a choice of minimal or identification key logging. Within table supplemental logging, you have a choice of using conditional or unconditional log groups.

See Also: [Supplemental Logging](#) on page 9-2

- Two new options have been added that affect the formatting of returned data. The `DBMS_LOGMNR.NO_SQL_DELIMITER` option suppresses the semi-colon at the end of `SQL_REDO` and `SQL_UNDO` statements. The `DBMS_LOGMNR.PRINT_PRETTY_SQL` option formats the reconstructed SQL statements so that they are easier to read.

See Also: [Formatting of Returned Data](#) on page 9-17

- A new option, `DBMS_LOGMNR.CONTINUOUS_MINE`, directs LogMiner to automatically add and mine redo log files that are archived after the LogMiner session has started.

See Also: [Continuous Mining](#) on page 9-25

- Use of the `DBMS_LOGMNR.NO_DICT_RESET_ONSELECT` option is no longer necessary. When DDL tracking is enabled, LogMiner stores old metadata definitions so that a second select operation has all the needed metadata versions.
- A new procedure, `DBMS_LOGMNR_D.SET_TABLESPACE`, recreates all LogMiner tables in a tablespace other than `SYSTEM`, which is the default.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference*

■ **Creating a Locally Managed SYSTEM Tablespace**

You can now create a locally managed `SYSTEM` tablespace. This can be done at create database time by specifying the `EXTENT MANAGEMENT LOCAL` clause of

the `CREATE DATABASE` statement, or you can migrate your existing `SYSTEM` tablespace to locally managed by using the `DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL` procedure.

See Also:

- ["Creating a Locally Managed SYSTEM Tablespace"](#) on page 2-26
- ["Migrating the SYSTEM Tablespace to a Locally Managed Tablespace"](#) on page 11-34

- **Mapping Files to Physical Devices**

With the introduction of host based Logical Volume Managers (LVM), and sophisticated storage subsystems that provide RAID features, it is not easy to determine file to device mapping. Oracle has created new views and a new `DBMS_STORAGE_MAP` package to enable you to map files to physical devices.

See Also: ["Mapping Files to Physical Devices"](#) on page 12-15

- **Renaming Columns**

Existing table columns can now be renamed.

See Also: ["Renaming Table Columns"](#) on page 15-14

- **Renaming Constraints**

Existing constraints on tables can now be renamed.

See Also: ["Renaming Constraints"](#) on page 21-19

- **Specifying a Default List Partition**

You can now use the keyword `DEFAULT` as the value-list descriptor for a partition defined for a list-partitioned table. This partition is used for inserting rows into the list-partitioned table when the partition key column does not match any of the literal values specified in the value-list descriptor for the partition.

See Also: ["Creating List-Partitioned Tables"](#) on page 17-13

- **Specifying Composite Range-List Partitioning**

This release introduces a new type of composite partitioning. You can now partition tables by the range-list method, where partitions are defined as range partitions and the subpartitions are defined as list partitions.

See Also: ["When to Use the Composite Range-List Partitioning Method"](#) on page 17-8

- **Optimizing SPLIT PARTITION and SPLIT SUBPARTITION Operations**

Under certain conditions, Oracle can perform a fast split operation that is more efficient than the normal split operation on a partition or subpartition.

See Also: ["Optimizing SPLIT PARTITION and SPLIT SUBPARTITION Operations"](#) on page 17-58

- **Auditing User SYS**

All operations done by user SYS (includes all AS SYSDBA and AS SYSOPER connections) can now be audited.

See Also: ["Auditing Administrative Users"](#) on page 26-6

- **Granting Object Privileges on Behalf of the Object Owner**

A new system privilege named GRANT ANY OBJECT PRIVILEGE allows you to grant object privileges as if you were the owner. It will appear (in views) as if the object owner granted the privilege, but audit records show the real person who granted the privilege.

See Also:

- ["Granting Object Privileges on Behalf of the Object Owner"](#) on page 25-14
- ["Revoking Object Privileges on Behalf of the Object Owner"](#) on page 25-17

- **Removal of LOB Column Restriction**

You can now create LOB columns in tablespaces that specify automatic segment-space management.

- **Elimination of Certain Distributed Database Initialization Parameters**

In earlier releases of Oracle, the `DISTRIBUTED_TRANSACTIONS` initialization parameter allowed you to specify a maximum number of distributed transactions in which a database can concurrently participate. This parameter has been eliminated and the number of concurrent distributed transactions is not limited. If specified, the `DISTRIBUTED_TRANSACTIONS` initialization parameter is ignored.

Also in earlier releases of Oracle, the max number of branches for each distributed transaction was specified by the `MAX_TRANSACTION_BRANCHES` initialization parameter. This parameter was eliminated in Oracle8i, but the maximum number of branches of a distributed transaction is still limited to 32. If specified, the `MAX_TRANSACTION_BRANCHES` initialization parameter is ignored.

Oracle9i Release 1 (9.0.1) New Features

Oracle9i brings a major new release of the Oracle database server. It includes features to make the database more available. More online operations reduce the need for offline maintenance. Management of the database requires less effort. Oracle9i can automatically create and manage the underlying operating system files required by the database. There is a theme of self management.

Performance is enhanced. The Database Resource Manager has new options that allow for more granular control of resources. The performance level required of a resource consumer group can be better sustained. Partitioning enhancements allow tables and indexes to be better partitioned for performance. Security enhancements are an important part of this release. Applications have available more and finer grained methods of implementing security and auditing.

The following are summaries of the new features of Oracle9i that are discussed in this book.

- **Online redefinition of tables**

The new `DBMS_REDEFINITION` PL/SQL package provides a mechanism to redefine tables online. When a table is redefined online, it is accessible to DML during much of the redefinition process. This provides a significant increase in availability compared to traditional methods of redefining tables that require tables to be taken offline.

See Also: ["Redefining Tables Online"](#) on page 15-16

- **ONLINE option for ANALYZE VALIDATE STRUCTURE statement**

The `ANALYZE` statement can now perform validation while DML is ongoing within the object being analyzed.

See Also: ["Validating Tables, Indexes, Clusters, and Materialized Views"](#) on page 21-6

- **Controlling Archive Lag**

Oracle now provides a time-based means of switching the current online redo log group. In a primary/standby configuration, where all noncurrent logs of the primary site are archived and shipped to the standby database, this effectively limits the number of redo records, as measured in time, that will not be applied in the standby database.

See Also: ["Controlling Archive Lag"](#) on page 7-10

- **Suspending a database**

Oracle9i includes a database suspend/resume feature. The `ALTER SYSTEM SUSPEND` statement suspends a database by halting all input and output (I/O) to datafiles and control files. When the database is suspended all preexisting I/O operations are allowed to complete and any new database accesses are placed in a queued state. The `ALTER SYSTEM RESUME` statement resumes normal database operation.

See Also: ["Suspending and Resuming a Database"](#) on page 4-17

- **Quiescing a database**

Oracle9i allows you to place the database into a quiesced state, where only DBA transactions, queries, or PL/SQL statements are allowed. This quiesced state allows you to perform administrative actions that cannot safely be done otherwise. The `ALTER SYSTEM QUIESCE RESTRICTED` statement places a database into a quiesced state.

See Also: ["Quiescing a Database"](#) on page 4-14

- **Resumable Space Allocation**

Oracle provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. This enables you to take corrective action, instead of the Oracle database server returning an

error to the user. After the error condition is corrected, the suspended operation automatically resumes.

See Also: ["Managing Resumable Space Allocation"](#) on page 14-14

- **More archiving destinations**

The maximum number of destinations to which you can archive the online redo log, has been increased from 5 to 10.

See Also: ["Specifying the Archive Destination"](#) on page 8-9

- **Automatic segment- space management**

Locally managed tablespaces allow extents to be managed automatically by Oracle. Oracle9i allows free and used space within segments stored in locally managed tablespaces to also be managed automatically. Using the `SEGMENT SPACE MANAGEMENT` clause of `CREATE TABLESPACE` you specify `AUTO` or `MANUAL` to specify the type of segment space management Oracle will use.

See Also: ["Specifying Segment Space Management in Locally Managed Tablespaces"](#) on page 11-8

- **Update of global indexes when partition maintenance is performed**

By default, many table maintenance operations on partitioned tables invalidate (mark `UNUSABLE`) global indexes. You must then rebuild the entire global index or, if partitioned, all of its partitions. Oracle9i allows you to override this default behavior. When you specify the `UPDATE GLOBAL INDEX` clause in your `ALTER TABLE` statement for the maintenance operation, the global index is updated in conjunction with the base table operation.

See Also: ["Maintaining Partitioned Tables"](#) on page 17-22

- **Multiple block sizes**

Oracle now supports multiple block sizes. It has a standard block size, as set by the `DB_BLOCK_SIZE` initialization parameter, and additionally up to 4 nonstandard block sizes. Nonstandard block sizes are specified when creating tablespaces. The standard block size is used for the `SYSTEM` tablespace and most other tablespaces. Multiple block size support allows for the transporting of tablespaces with unlike block sizes between databases.

See Also: ["Specifying Database Block Sizes"](#) on page 2-37

- **Dynamic buffer cache**

The size of the buffer cache subcomponent of the System Global Area is now dynamic. The `DB_BLOCK_BUFFERS` initialization parameter has been replaced by a new dynamic parameter, `DB_CACHE_SIZE`, where the user specifies the size of the buffer subcache for the standard database block size. The buffer cache now consists of subcaches when multiple block sizes are specified for the database. Up to four `DB_nK_CACHE_SIZE` initialization parameters allow you to specify the sizes of buffer subcaches for the additional block sizes.

See Also: ["Setting Initialization Parameters that Affect the Size of the SGA"](#) on page 2-38

- **Dynamic SGA**

The initialization parameters affecting the size of SGA have been made dynamic. It is possible to alter the size of SGA dynamically through an `ALTER SYSTEM SET` statement.

See Also: ["Setting Initialization Parameters that Affect the Size of the SGA"](#) on page 2-38

- **Automatic undo management**

Historically, Oracle has used rollback segments to store undo. Undo is defined as information that can be used to roll back, or undo, changes to the database when necessary. Oracle now enables you to create an undo tablespace to store undo. Using an undo tablespace eliminates the complexities of managing rollback segment space, and enables you to exert control over how long undo is retained before being overwritten.

See Also: [Chapter 13, "Managing Undo Space"](#)

- **Oracle managed files**

The Oracle managed files feature of Oracle9i eliminates the need for you to directly manage the files comprising an Oracle database. Through the `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters, you specify the file system directory to be used for a particular type of file comprising a tablespace, online redo log file, or control file. Oracle then

ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed.

See Also: [Chapter 3, "Using Oracle-Managed Files"](#)

- **Automatic deletion of datafiles**

Oracle9i provides an option to automatically remove a tablespaces's operating system files (datafiles) when the tablespace is dropped using the `DROP TABLESPACE` statement. A similar option for the `ALTER DATABASE TEMPFILE` statement, causes deletion the operating system files associated with a temporary file.

See Also:

- ["Dropping Tablespaces"](#) on page 11-29
- ["Altering a Locally Managed Temporary Tablespace"](#) on page 11-14

- **Metadata API**

A new PL/SQL package, `DBMS_METADATA.GET_DDL`, allows you to obtain metadata (in the form of DDL used to create the object) about a schema object.

See Also: ["Using PL/SQL Packages to Display Information About Schema Objects"](#) on page 21-30

- **External tables**

Oracle9i allows you read-only access to data in external tables. External tables are defined as tables that do not reside in the database, and can be in any format for which an access driver is provided. The `CREATE TABLE . . . ORGANIZATION EXTERNAL` statement specifies metadata describing the external table. Oracle currently provides the `ORACLE_LOADER` access driver which provides data mapping capabilities that are a subset of the SQL*Loader control file syntax.

See Also: ["Managing External Tables"](#) on page 15-33

- **Constraint enhancements**

Enhancements to the `USING INDEX` clause of `CREATE TABLE` or `ALTER TABLE` allow you to specify the creation or use of a specific index when a

unique or primary key constraint is created or enabled. Additionally, you can prevent the dropping of the index enforcing a unique or primary key constraint when the constraint is dropped or disabled.

See Also:

- ["Creating an Index Associated with a Constraint"](#) on page 16-11
- ["Managing Integrity Constraints"](#) on page 21-14

- **Server parameter file**

Oracle has traditionally stored initialization parameters in a text initialization parameter file, often on a client machine. Starting with Oracle9i, you can elect to maintain initialization parameters in a server parameter file, which is a binary parameter file stored on the database server. Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running persist across instance shutdown and startup.

See Also: ["Managing Initialization Parameters Using a Server Parameter File"](#) on page 2-44

- **Default temporary tablespace**

The new `DEFAULT TEMPORARY TABLESPACE` clause of the `CREATE DATABASE` statement allows you to create a default temporary tablespace at database creation time. This tablespace is used as the default temporary tablespace for users who are not otherwise assigned a temporary tablespace.

See Also: ["Creating a Default Temporary Tablespace"](#) on page 2-24

- **Setting the database time zone**

The `CREATE DATABASE` statement now has a `SET TIME_ZONE` clause that allows you to set the time zone of the database as a displacement from UTC (Coordinated Universal Time—formerly Greenwich Mean Time). Oracle normalizes all `TIMESTAMP WITH LOCAL TIME_ZONE` data to the time zone of the database when the data is stored on disk. Additionally, a new session parameter `TIME_ZONE` has been added to the `SET` clause of `ALTER SESSION`.

See Also: ["Step 6: Issue the CREATE DATABASE Statement"](#) on page 2-18

- **Transaction Naming**

Oracle now allows you to assign a name to a transaction. The transaction name is helpful in resolving in-doubt distributed transactions, and replaces a `COMMIT COMMENT`.

See Also: ["Naming Transactions"](#) on page 32-2

- **Database Configuration Assistant changes**

The Database Configuration Assistant has been redesigned. It now provides templates, which are saved definitions of databases, from which you can generate your database. Oracle provides templates, or you can create your own templates by modifying existing ones, defining new ones, or by capturing the definition of an existing database.

When creating a database with the Database Configuration Assistant, you can either initially include, or later add as an option, Oracle's new Sample Schemas. These schemas are the basis for many of the examples used in Oracle documentation.

See Also: ["Using the Database Configuration Assistant"](#) on page 2-5

- **Monitoring index usage**

A `MONITORING USAGE` clause has been added for the `ALTER INDEX` statement. It allows you to monitor an index to determine if it is actively being used.

See Also: ["Monitoring Index Usage"](#) on page 16-21

- **List partitioning**

Oracle introduces list partitioning, which enables you to specify a list of discrete values for the partitioning column in the description for each partition. The list partitioning method is specifically designed for modeling data distributions that follow discrete values. This cannot be easily done by range or hash partitioning.

See Also: [Chapter 17, "Managing Partitioned Tables and Indexes"](#)

- **Hash partitioning of index-organized tables**

In this release, support has been added for partitioning index-organized tables by the hash method. Previously, they could be partitioned, but only by the range method.

See Also: ["Creating Partitioned Index-Organized Tables"](#) on page 17-19

- **Dynamic job queue processes**

The job queue process creation has been made dynamic so that only the required number of processes are created to execute the jobs that are ready for execution. A job queue coordinator background process (CJQ) dynamically spawns *Jnnn* processes to execute jobs.

See Also: ["Enabling Processes Used for Executing Jobs"](#) on page 10-2

- **New in the Database Resource Manager for Oracle9i**

The following new functionality has been added to the Database Resource Manager:

- Ability to create an active session pool. This pool consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will abort.
- Automatic switching of users from one group to another group based on administrator defined criteria. If a member of a particular group of users creates a session that executes for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.
- Ability to prevent the execution of operations that are estimated to run for a longer time than a predefined limit
- Ability to create an undo pool. This pool consists of the amount of undo space that can be consumed in by a group of users.

See Also: [Chapter 27, "Using the Database Resource Manager"](#)

- **Proxy authentication and authorization**

Oracle9i enables you to authorize a middle-tier server to act on behalf of a client. The `GRANT CONNECT THROUGH` clause of the `ALTER USER` statement specifies this functionality. You can also specify roles that the middle tier is permitted to activate when connecting as the client.

See Also: ["Proxy Authentication and Authorization"](#) on page 24-16

- **Application roles**

Oracle provides a mechanism by which roles granted to application users are enabled using a designated PL/SQL package. This feature introduces the `IDENTIFIED USING package` clause for the `CREATE ROLE` statement.

See Also: ["Role Authorization by an Application"](#) on page 25-8

- **Fine-grained auditing**

In Oracle's traditional auditing methods, a fixed set of facts is recorded in the audit trail. Audit options can only be set to monitor access of objects or privileges. A new PL/SQL package, `DBMS_FGA`, allows applications to implement fine-grained auditing of data access based on content.

See Also: ["Fine-Grained Auditing"](#) on page 26-18

- **New in LogMiner for Release 9.0.1**

LogMiner release 9.0.1 has added support for many new features. Some of the new features work with any redo log files from an Oracle 8.0 or later database. Other features only work with redo log files produced on Oracle9i or later.

New Features for Redo Log Files Generated by Oracle9i or Later

For any redo log files generated by Oracle9i or later, LogMiner now provides support for the following:

- Index clusters
- Chained and migrated rows
- Direct path inserts (with `ARCHIVELOG` mode enabled)
- Extracting the data dictionary into the redo log files. See [Extracting a Dictionary to the Redo Logs](#) on page 9-6.

- Using the online catalog as the data dictionary. See "[Using the Online Catalog](#)" on page 9-8.
- Tracking of all data definition language (DDL) operations, which enables you to monitor schema evolution. See "[Tracking DDL Statements](#)" on page 9-9.
- Viewing user-executed DDL in the `SQL_REDO` column. Information regarding the original database user is also returned.
- Generating `SQL_REDO` and `SQL_UNDO` with primary key information for updates. That is, updated rows are identified by primary keys and `ROWIDS` (provided supplemental logging is enabled), thereby making it easier to apply the statements to a different database.

New Features for Redo Log Files Generated by Oracle Release 8.0 or Later

For any redo log files generated by Oracle release 8.0 or later, LogMiner now provides support for the following:

- Limiting `V$LOGMNR_CONTENTS` data to rows belonging to committed transactions only. This option enables you to filter out rolled back transactions and transactions that are in progress. See the information about options in "[Start a LogMiner Session](#)" on page 9-26.
- Performing queries based on actual data values in the redo log files. See "[Extracting Actual Data Values from Redo Logs](#)" on page 9-18.

See Also: [Chapter 9, "Using LogMiner to Analyze Redo Logs"](#)

Part I

Basic Database Administration

Part I provides an overview of the responsibilities of a database administrator, and describes the creation of a database and how to start up and shut down an instance of the database. It contains the following chapters:

- [Chapter 1, "The Oracle Database Administrator"](#)
- [Chapter 2, "Creating an Oracle Database"](#)
- [Chapter 3, "Using Oracle-Managed Files"](#)
- [Chapter 4, "Starting Up and Shutting Down"](#)

The Oracle Database Administrator

This chapter describes your responsibilities as a database administrator (DBA) who administers the Oracle database server.

The following topics are discussed:

- [Types of Oracle Users](#)
- [Tasks of a Database Administrator](#)
- [Identifying Your Oracle Database Software Release](#)
- [Database Administrator Security and Privileges](#)
- [Database Administrator Authentication](#)
- [Creating and Maintaining a Password File](#)
- [Database Administrator Utilities](#)

Types of Oracle Users

The types of users and their roles and responsibilities at a site can vary. A small site can have one database administrator who administers the database for application developers and users. A very large site can find it necessary to divide the duties of a database administrator among several people, and among several areas of specialization.

This section contains the following topics:

- [Database Administrators](#)
- [Security Officers](#)
- [Network Administrators](#)
- [Application Developers](#)
- [Application Administrators](#)
- [Database Users](#)

Database Administrators

Each database requires at least one database administrator (DBA) to administer it. Because an Oracle database system can be large and can have many users, often this is not a one person job. In such cases, there is a group of DBAs who share responsibility.

A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application
- Creating primary objects (tables, views, indexes) once application developers have designed an application
- Modifying the database structure, as necessary, from information given by application developers
- Enrolling users and maintaining system security
- Ensuring compliance with your Oracle license agreement

- Controlling and monitoring user access to the database
- Monitoring and optimizing the performance of the database
- Planning for backup and recovery of database information
- Maintaining archived data on tape
- Backing up and restoring the database
- Contacting Oracle Corporation for technical support

Security Officers

In some cases, a site assigns one or more security officers to a database. A security officer enrolls users, controls and monitors user access to the database, and maintains system security. As a DBA, you might not be responsible for these duties if your site has a separate security officer.

Network Administrators

Some sites have one or more network administrators. A network administrator can administer Oracle networking products, such as Oracle Net Services.

See Also: Part VI, "[Distributed Database Management](#)" for information on network administration in a distributed environment

Application Developers

Application developers design and implement database applications. Their responsibilities include the following tasks:

- Designing and developing the database application
- Designing the database structure for an application
- Estimating storage requirements for an application
- Specifying modifications of the database structure for an application
- Relaying the above information to a database administrator
- Tuning the application during development
- Establishing an application's security measures during development

Application developers can perform some of these tasks in collaboration with DBAs.

Application Administrators

An Oracle site can assign one or more application administrators to administrate a particular application. Each application can have its own administrator.

Database Users

Database users interact with the database through applications or utilities. A typical user's responsibilities include the following tasks:

- Entering, modifying, and deleting data, where permitted
- Generating reports from the data

Tasks of a Database Administrator

The following tasks present a prioritized approach for designing, implementing, and maintaining an Oracle Database:

Task 1: Evaluate the Database Server Hardware

Task 2: Install the Oracle Software

Task 3: Plan the Database

Task 4: Create and Open the Database

Task 5: Back Up the Database

Task 6: Enroll System Users

Task 7: Implement the Database Design

Task 8: Back Up the Fully Functional Database

Task 9: Tune Database Performance

These tasks are discussed in succeeding sections.

Note: If upgrading to a new release, back up your existing production database before installation. For information on preserving your existing production database, see *Oracle9i Database Migration*.

Task 1: Evaluate the Database Server Hardware

Evaluate how Oracle and its applications can best use the available computer resources. This evaluation should reveal the following information:

- How many disk drives are available to Oracle and its databases
- How many, if any, dedicated tape drives are available to Oracle and its databases
- How much memory is available to the instances of Oracle you will run (see your system's configuration documentation)

Task 2: Install the Oracle Software

As the database administrator, you install the Oracle database server software and any front-end tools and database applications that access the database. In some distributed processing installations, the database is controlled by a central computer (database server) and the database tools and applications are executed on remote computers (clients). In this case, you must also install the Oracle Net components necessary to connect the remote machines to the computer that executes Oracle.

For more information on what software to install, see "[Identifying Your Oracle Database Software Release](#)" on page 1-8.

See Also: For specific requirements and instructions for installation, refer to the following documentation:

- Your operating system specific Oracle documentation
- Your installation guides for your front-end tools and Oracle Net drivers.

Task 3: Plan the Database

As the database administrator, you must plan:

- The logical storage structure of the database
- The overall database design
- A backup strategy for the database

It is important to plan how the logical storage structure of the database will affect system performance and various database management operations. For example, before creating any tablespaces for your database, you should know how many datafiles will make up the tablespace, what type of information will be stored in

each tablespace, and on which disk drives the datafiles will be physically stored. When planning the overall logical storage of the database structure, take into account the effects that this structure will have when the database is actually created and running. Such considerations include how the logical storage structure database will affect the following:

- The performance of the computer executing Oracle
- The performance of the database during data access operations
- The efficiency of backup and recovery procedures for the database

Plan the relational design of the database objects and the storage characteristics for each of these objects. By planning the relationship between each object and its physical storage before creating it, you can directly affect the performance of the database as a unit. Be sure to plan for the growth of the database.

In distributed database environments, this planning stage is extremely important. The physical location of frequently accessed data dramatically affects application performance.

During the planning stage, develop a backup strategy for the database. You can alter the logical storage structure or design of the database to improve backup efficiency.

It is beyond the scope of this book to discuss relational and distributed database design. If you are not familiar with such design issues, refer to accepted industry-standard documentation.

Part II, "[Oracle Server Processes and Storage Structure](#)" and Part III, "[Schema Objects](#)" provide specific information on creating logical storage structures, objects, and integrity constraints for your database.

Task 4: Create and Open the Database

When you complete the database design, you can create the database and open it for normal use. You can create a database at installation time, using the Database Configuration Assistant, or you can supply your own scripts for creating a database.

Either way, refer to [Chapter 2, "Creating an Oracle Database"](#), for information on creating a database and [Chapter 4, "Starting Up and Shutting Down"](#) for guidance in starting up the database.

Task 5: Back Up the Database

After you create the database structure, carry out the backup strategy you planned for the database. Create any additional redo log files, take the first full database backup (online or offline), and schedule future database backups at regular intervals.

See Also: For instructions on customizing your backup operations and performing recovery procedures see either of the following:

- *Oracle9i User-Managed Backup and Recovery Guide*
- *Oracle9i Recovery Manager User's Guide*

Task 6: Enroll System Users

After you back up the database structure, you can enroll the users of the database in accordance with your Oracle license agreement, create appropriate roles for these users, and grant these roles.

The following chapters will help you in this endeavor:

- [Chapter 23, "Establishing Security Policies"](#)
- [Chapter 24, "Managing Users and Resources"](#)
- [Chapter 25, "Managing User Privileges and Roles"](#)

Task 7: Implement the Database Design

After you create and start the database, and enroll the system users, you can implement the planned logical structure database by creating all necessary tablespaces. When you complete this, you can create the objects for the database.

Part II, "[Oracle Server Processes and Storage Structure](#)" and Part III, "[Schema Objects](#)" contain information which can help you create logical storage structures and objects for your database.

Task 8: Back Up the Fully Functional Database

Now that the database is fully implemented, again back up the database. In addition to regularly scheduled backups, you should always back up your database immediately after implementing changes to the database structure.

Task 9: Tune Database Performance

Optimizing the performance of the database is one of your ongoing responsibilities as a DBA. Additionally, Oracle provides a database resource management feature that enables you to control the allocation of resources to various user groups.

The database resource manager is described in [Chapter 27, "Using the Database Resource Manager"](#).

See Also: *Oracle9i Database Performance Tuning Guide and Reference* contains information about tuning your database and applications.

Identifying Your Oracle Database Software Release

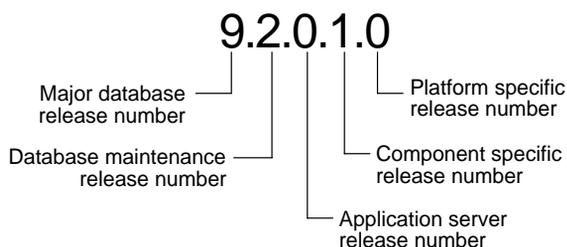
Because the Oracle database server continues to evolve and can require maintenance, Oracle periodically produces new releases. Because only some users initially subscribe to a new release or require specific maintenance, multiple releases of the product can exist simultaneously.

As many as five numbers may be required to fully identify a release. The significance of these numbers is discussed below.

Release Number Format

To understand the release level nomenclature used by Oracle, examine the following example of an Oracle database server labeled "Release 9.2.0.1.0."

Figure 1–1 Example of an Oracle Release Number



Note: Starting with release 9.2, maintenance releases of Oracle are denoted by a change to the second digit of a release number. In previous releases, the third digit indicated a particular maintenance release.

Major Database Release Number

This is the most general identifier. It represents a major new edition (or version) of the software that contains significant new functionality.

Database Maintenance Release Number

This digit represents a maintenance release level. Some new features may also be included.

Application Server Release Number

This digit reflects the release level of the Oracle9i Application Server (Oracle9iAS).

Component Specific Release Number

This digit identifies a release level specific to a component. Different components can have different numbers in this position depending upon, for example, component patch sets or interim releases.

Platform Specific Release Number

This digit identifies a platform specific release. Usually this is a patch set. Where different platforms require the equivalent patch set, this digit will be the same across the effected platforms.

Checking Your Current Release Number

To identify the release of the Oracle database server that is currently installed and to see the release levels of other Oracle components you are using, query the data dictionary view `PRODUCT_COMPONENT_VERSION`. A sample query is shown below. Other product release levels may increment independently of the database server.

```
COL PRODUCT FORMAT A35
COL VERSION FORMAT A15
COL STATUS FORMAT A15
SELECT * FROM PRODUCT_COMPONENT_VERSION;
```

PRODUCT	VERSION	STATUS
NLSRTL	9.2.0.1.0	Production
Oracle9i Enterprise Edition	9.2.0.1.0	Production
PL/SQL	9.2.0.1.0	Production
TNS for Solaris:	9.2.0.1.0	Production

It's important to convey to Oracle the information displayed by this query when you report problems with the software.

Optionally, you can query the `V$VERSION` view to see component-level information.

Database Administrator Security and Privileges

To accomplish the administrative tasks of an Oracle DBA, you need extra privileges both within the database and possibly in the operating system of the server on which the database runs. Access to a database administrator's account should be tightly controlled.

This section contains the following topics:

- [The Database Administrator's Operating System Account](#)
- [Database Administrator Usernames](#)

The Database Administrator's Operating System Account

To perform many of the administrative duties for a database, you must be able to execute operating system commands. Depending on the operating system that executes Oracle, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require more operating system privileges or access rights than many database users require (for

example, to perform Oracle software installation). Although you do not need the Oracle files to be stored in your account, you should have access to them.

See Also: Your operating system specific Oracle documentation. The method of distinguishing a database administrator's account is operating system specific.

Database Administrator Usernames

Two user accounts are automatically created with the database:

- SYS (default password: CHANGE_ON_INSTALL)
- SYSTEM (default password: MANAGER)

Note: Oracle recommends that you specify passwords for SYS and SYSTEM at database creation time, rather than using these default passwords. This is explained in "[Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM](#)" on page 2-23.

If you use the default passwords, to prevent inappropriate access to the data dictionary tables or other tampering with the database, it is important that you change the passwords for the SYS and SYSTEM usernames immediately after creating an Oracle database.

It is suggested that you create at least one additional administrator user, and grant that user the DBA role, to use when performing daily administrative tasks. It is recommended that you do not use SYS and SYSTEM for these purposes.

Note Regarding Security Enhancements: In this release of Oracle and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts.

- During initial installation with the Database Configuration Assistant (DCBA), all default database user accounts except SYS, SYSTEM, SCOTT, DBSNMP, OUTLN, AURORA\$JIS\$UTILITY\$, AURORA\$ORB\$UNAUTHENTICATED and OSE\$HTTP\$ADMIN are locked and expired. To activate a locked account, the DBA must manually unlock it and reassign it a new password.
 - In addition, the DBCA prompts for passwords for users SYS and SYSTEM during initial installation of the database rather than assigning default passwords to them. A CREATE DATABASE statement issued manually also lets you supply passwords for these two users.
-
-

SYS

When any database is created, the user SYS is automatically created and granted the DBA role.

All of the base tables and views for the database's data dictionary are stored in the schema SYS. These base tables and views are critical for the operation of Oracle. To maintain the integrity of the data dictionary, tables in the SYS schema are manipulated only by Oracle. They should never be modified by any user or database administrator, and no one should create any tables in the schema of user SYS. (However, you can change the storage parameters of the data dictionary settings if necessary.)

Ensure that most database users are never able to connect using the SYS account.

SYSTEM

When a database is created, the user SYSTEM is also automatically created and granted the DBA role.

The SYSTEM username is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle

options and tools. Never create in the `SYSTEM` schema tables of interest to individual users.

The DBA Role

A predefined role, named `DBA`, is automatically created with every Oracle database. This role contains most database system privileges. Therefore, it is very powerful and should be granted only to fully functional database administrators.

Note: The `DBA` role does not include the `SYSDBA` or `SYSOPER` system privileges. These are special administrative privileges that allow an administrator to perform basic database administration tasks, such as creating the database and instance startup and shutdown. These system privileges are discussed in "[Administrative Privileges](#)" on page 1-13.

Database Administrator Authentication

As a `DBA`, you often perform special operations such as shutting down or starting up a database. Because only a `DBA` should perform these operations, the database administrator usernames require a secure authentication scheme.

This section contains the following topics:

- [Administrative Privileges](#)
- [Selecting an Authentication Method](#)
- [Using Operating System \(OS\) Authentication](#)
- [Using Password File Authentication](#)

Administrative Privileges

Administrative privileges that are required for an administrator to perform basic database operations are granted through two special system privileges, `SYSDBA` and `SYSOPER`. You must have one of these privileges granted to you, depending upon the level of authorization you require.

Note: The `SYSDBA` and `SYSOPER` system privileges allow access to a database instance even when the database is not open. Control of these privileges is totally outside of the database itself.

While referred to as system privileges, `SYSDBA` and `SYSOPER`, can also be thought of as types of connections (for example, you specify: `CONNECT AS SYSDBA`) that enable you to perform certain database operations for which privileges cannot be granted in any other fashion.

SYSDBA and SYSOPER

The following are the operations that are authorized by the `SYSDBA` and `SYSOPER` system privileges:

System Privilege	Operations Authorized
<code>SYSDBA</code>	<ul style="list-style-type: none"> ■ Perform <code>STARTUP</code> and <code>SHUTDOWN</code> operations ■ <code>ALTER DATABASE: open, mount, back up, or change character set</code> ■ <code>CREATE DATABASE</code> ■ <code>CREATE SPFILE</code> ■ <code>ARCHIVELOG</code> and <code>RECOVERY</code> ■ Includes the <code>RESTRICTED SESSION</code> privilege <p>Effectively, this system privilege allows a user to connect as user <code>SYS</code>.</p>
<code>SYSOPER</code>	<ul style="list-style-type: none"> ■ Perform <code>STARTUP</code> and <code>SHUTDOWN</code> operations ■ <code>CREATE SPFILE</code> ■ <code>ALTER DATABASE OPEN/MOUNT/BACKUP</code> ■ <code>ARCHIVELOG</code> and <code>RECOVERY</code> ■ Includes the <code>RESTRICTED SESSION</code> privilege <p>This privilege allows a user to perform basic operational tasks, but without the ability to look at user data.</p>

The manor in which you are authorized to use these privileges depends upon the method of authentication that you choose to use.

When you connect with `SYSDBA` or `SYSOPER` privileges, you connect with a default schema, not with the schema that is generally associated with your username. For `SYSDBA` this schema is `SYS`; for `SYSOPER` the schema is `PUBLIC`.

Connecting with Administrative Privileges: Example

This example illustrates that a user is assigned another schema (`SYS`) when connecting with the `SYSDBA` system privilege.

Assume that user `scott` has issued the following statements:

```
CONNECT scott/password
CREATE TABLE admin_test(name VARCHAR2(20));
```

Later, `scott` issues these statements:

```
CONNECT scott/password AS SYSDBA
SELECT * FROM admin_test;
```

User `scott` now receives the following error:

```
ORA-00942: table or view does not exist
```

This is because `scott` now references the `SYS` schema by default. The table was created in the `scott` schema.

See Also:

- ["Using Operating System \(OS\) Authentication"](#) on page 1-17
- ["Using Password File Authentication"](#) on page 1-18

Selecting an Authentication Method

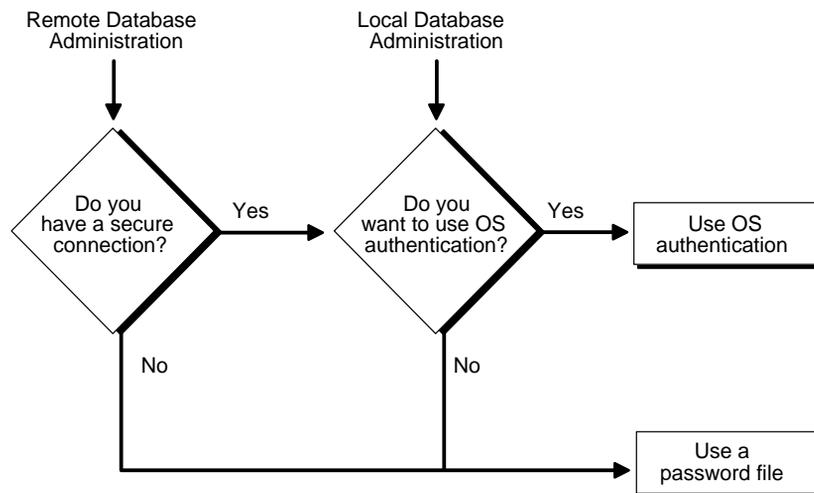
The following methods are available for authenticating database administrators:

- Operating system (OS) authentication
- Password files

Note: These methods replace the `CONNECT INTERNAL` syntax provided with earlier versions of Oracle. `CONNECT INTERNAL` is no longer allowed.

Your choice will be influenced by whether you intend to administer your database locally on the same machine where the database resides, or whether you intend to administer many different databases from a single remote client. [Figure 1-2](#) illustrates the choices you have for database administrator authentication schemes.

Figure 1-2 Database Administrator Authentication Methods



If you are performing remote database administration, you should consult your Oracle Net documentation to determine if you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure.

See Also:

- *Oracle9i Database Concepts* for additional information about user authentication
- ["User Authentication"](#) on page 23-2
- ["User Authentication Methods"](#) on page 24-8
- *Oracle9i Net Services Administrator's Guide*

Non-Secure Remote Connections

To connect to Oracle as a privileged user over a non-secure connection, you must be authenticated by a password file. When using password file authentication, the

database uses a password file to keep track of database usernames that have been granted the `SYSDBA` or `SYSOPER` system privilege.

This form of authentication is discussed in "[Using Password File Authentication](#)" on page 1-18.

Local Connections and Secure Remote Connections

To connect to Oracle as a privileged user over a local connection or a secure remote connection, you have the following options:

- You can connect and be authenticated by a password file, provided the database has a password file and you have been granted the `SYSDBA` or `SYSOPER` system privilege.
- If the server is not using a password file, or if you have not been granted `SYSDBA` or `SYSOPER` privileges and are therefore not in the password file, you can use OS authentication. On most operating systems, OS authentication for database administrators involves placing the OS username of the database administrator in a special group, generically referred to as `OSDBA`.

Using Operating System (OS) Authentication

This section describes how to authenticate an administrator using the operating system.

Preparing to Use OS Authentication

To enable authentication of an administrative user using the operating system you must do the following:

1. Create an operating system account for the user.
2. Add the user to the `OSDBA` or `OSOPER` operating system defined groups.
3. Ensure that the initialization parameter, `REMOTE_LOGIN_PASSWORDFILE`, is set to `NONE`. This is the default value for this parameter.

Connecting Using OS Authentication

A user can be authenticated, enabled as an administrative user, and connected to a local database by typing one of the following SQL*Plus commands:

```
CONNECT / AS SYSDBA  
CONNECT / AS SYSOPER
```

For a remote database connection over a secure connection, the user must also specify the net service name of the remote database:

```
CONNECT /@net_service_name AS SYSDBA
CONNECT /@net_service_name AS SYSOPER
```

See Also: *SQL*Plus User's Guide and Reference* for syntax of the `CONNECT` command

OSDBA and OSOPER

Two special operating system groups control database administrator connections when using OS authentication. These groups are generically referred to as OSDBA and OSOPER. The groups are created and assigned specific names as part of the database installation process. The specific names vary depending upon your operating system and are listed in the following table:

Operating System Group	UNIX	Windows
OSDBA	dba	ORA_DBA
OSOPER	oper	ORA_OPER

The default names assumed by the Oracle Universal Installer can be overridden. How you create the OSDBA and OSOPER groups is operating system specific.

The following describes how membership in the OSDBA or OSOPER group affects your connection to Oracle:

- If you are a member of the OSDBA group, and specify `AS SYSDBA` when you connect to the database, you are granted the `SYSDBA` system privilege.
- If you are a member of the OSOPER group, and specify `AS SYSOPER` when you connect to the database, you are granted the `SYSOPER` system privilege.
- If you are not a member of the associated operating system group for `SYSDBA` or `SYSOPER` system privileges, the `CONNECT` command fails.

See Also: Your operating system specific Oracle documentation for information about creating the OSDBA and OSOPER groups

Using Password File Authentication

This section describes how to authenticate an administrative user using password file authentication.

Preparing to Use Password File Authentication

To enable authentication of an administrative user using password file authentication you must do the following:

1. Create an operating system account for the user.
2. If not already created, Create the password file using the ORAPWD utility:

```
ORAPWD FILE=filename PASSWORD=password ENTRIES=max_users
```
3. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE.
4. Connect to the database as user SYS (or as another user with the administrative privilege).
5. If the user does not already exist in the database, create the user. Grant the SYSDBA or SYSOPER system privilege to the user:

```
GRANT SYSDBA to scott;
```

This statement adds the user to the password file, thereby enabling connection AS SYSDBA.

See Also: ["Creating and Maintaining a Password File"](#) on page 1-20 for instructions for creating and maintaining a password file

Connecting Using Password File Authentication

Administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus CONNECT command. They must connect using their username and password and with the AS SYSDBA or AS SYSOPER clause. For example, user scott has been granted the SYSDBA privilege, so he can connect as follows:

```
CONNECT scott/tiger AS SYSDBA
```

However, since scott has not been granted the SYSOPER privilege, the following command will fail:

```
CONNECT scott/tiger AS SYSOPER
```

Note: Operating system authentication takes precedence over password file authentication. Specifically, if you are a member of the OSDBA or OSOPER group for the operating system, and you connect as SYSDBA or SYSOPER, you will be connected with associated administrative privileges regardless of the *username/password* that you specify.

If you are not in the OSDBA or OSOPER groups, and you are not in the password file, then the connection will fail.

See Also: *SQL*Plus User's Guide and Reference* for syntax of the `CONNECT` command

Creating and Maintaining a Password File

You can create a password file using the password file creation utility, `ORAPWD`. For some operating systems, you can create this file as part of your standard installation.

This section contains the following topics:

- [Using ORAPWD](#)
- [Setting REMOTE_LOGIN_PASSWORDFILE](#)
- [Adding Users to a Password File](#)
- [Maintaining a Password File](#)

See Also: Your operating system specific Oracle documentation for information on using the installer utility to install the password file

Using ORAPWD

When you invoke the password file creation utility without supplying any parameters, you receive a message indicating the proper use of the command as shown in the following sample output:

```
orapwd
Usage: orapwd file=<fname> password=<password> entries=<users>
where
file - name of password file (mand),
password - password for SYS (mand),
```

entries - maximum number of distinct DBAs and OPERs (opt),
There are no spaces around the equal-to (=) character.

The following command creates a password file named `acct.pwd` that allows up to 30 privileged users with different passwords. In this example, the file is initially created with the password `secret` for users connecting as `SYS`.

```
ORAPWD FILE=acct.pwd PASSWORD=secret ENTRIES=30
```

Following are descriptions of the parameters in the `ORAPWD` utility.

FILE

This parameter sets the name of the password file being created. You must specify the full path name for the file. The contents of this file are encrypted, and the file cannot be read directly. This parameter is mandatory.

The types of filenames allowed for the password file are operating system specific. Some operating systems require the password file to be a specific format and located in a specific directory. Other operating systems allow the use of environment variables to specify the name and location of the password file. See your operating system specific Oracle documentation for the names and locations allowed on your platform.

If you are running multiple instances of Oracle using Oracle9i Real Application Clusters, the environment variable for each instance should point to the same password file.

Caution: It is critically important to the security of your system that you protect your password file and the environment variables that identify the location of the password file. Any user with access to these could potentially compromise the security of the connection.

PASSWORD

This parameter sets the password for user `SYS`. If you issue the `ALTER USER` statement to change the password for `SYS` after connecting to the database, both the password stored in the data dictionary and the password stored in the password file are updated. This parameter is mandatory.

ENTRIES

This parameter specifies the number of entries that you require the password file to accept. This number corresponds to the number of distinct users allowed to connect

to the database as `SYSDBA` or `SYSOPER`. The actual number of allowable entries can be higher than the number of users because the `ORAPWD` utility continues to assign password entries until an operating system block is filled. For example, if your operating system block size is 512 bytes, it holds four password entries. The number of password entries allocated is always multiple of four.

Entries can be reused as users are added to and removed from the password file. If you intend to specify `REMOTE_LOGON_PASSWORDFILE=EXCLUSIVE`, and to allow the granting of `SYSDBA` and `SYSOPER` privileges to users, this parameter is required.

Caution: When you exceed the allocated number of password entries, you must create a new password file. To avoid this necessity, allocate a number of entries that is larger than you think you will ever need.

Setting `REMOTE_LOGIN_PASSWORDFILE`

In addition to creating the password file, you must also set the initialization parameter `REMOTE_LOGIN_PASSWORDFILE` to the appropriate value. The values recognized are described as follows:

Value	Description
<code>NONE</code>	Setting this parameter to <code>NONE</code> causes Oracle to behave as if the password file does not exist. That is, no privileged connections are allowed over non-secure connections. <code>NONE</code> is the default value for this parameter.
<code>EXCLUSIVE</code>	An <code>EXCLUSIVE</code> password file can be used with only one database. Only an <code>EXCLUSIVE</code> file can contain the names of users other than <code>SYS</code> . Using an <code>EXCLUSIVE</code> password file allows you to grant <code>SYSDBA</code> and <code>SYSOPER</code> system privileges to individual users and have them connect as themselves.
<code>SHARED</code>	A <code>SHARED</code> password file can be used by multiple databases. However, the only user recognized by a <code>SHARED</code> password file is <code>SYS</code> . You cannot add users to a <code>SHARED</code> password file. All users needing <code>SYSDBA</code> or <code>SYSOPER</code> system privileges must connect using the same name, <code>SYS</code> , and password. This option is useful if you have a single DBA administering multiple databases.

Suggestion: To achieve the greatest level of security, you should set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE` immediately after creating the password file.

Adding Users to a Password File

When you grant `SYSDBA` or `SYSOPER` privileges to a user, that user's name and privilege information are added to the password file. If the server does not have an `EXCLUSIVE` password file (that is, if the initialization parameter `REMOTE_LOGIN_PASSWORDFILE` is `NONE` or `SHARED`) you receive an error message if you attempt to grant these privileges.

A user's name remains in the password file only as long as that user has at least one of these two privileges. If you revoke both of these privileges, the user is removed from the password file.

To Create a Password File and Add New Users to It

1. Follow the instructions for creating a password file as explained in "[Using ORAPWD](#)" on page 1-20.
2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE`.
3. Connect with `SYSDBA` privileges as shown in the following example:

```
CONNECT SYS/password AS SYSDBA
```
4. Start up the instance and create the database if necessary, or mount and open an existing database.
5. Create users as necessary. Grant `SYSDBA` or `SYSOPER` privileges to yourself and other users as appropriate. See "[Granting and Revoking SYSDBA and SYSOPER Privileges](#)".

Granting the `SYSDBA` or `SYSOPER` privilege to a user causes their username to be added to the password file. This enables the user to connect to the database as `SYSDBA` or `SYSOPER` by specifying username and password (instead of using `SYS`). The use of a password file does not prevent OS authenticated users from connecting if they meet the criteria for OS authentication.

Granting and Revoking SYSDBA and SYSOPER Privileges

If your server is using an `EXCLUSIVE` password file, use the `GRANT` statement to grant the `SYSDBA` or `SYSOPER` system privilege to a user, as shown in the following example:

```
GRANT SYSDBA TO scott;
```

Use the `REVOKE` statement to revoke the `SYSDBA` or `SYSOPER` system privilege from a user, as shown in the following example:

```
REVOKE SYSDBA FROM scott;
```

Because `SYSDBA` and `SYSOPER` are the most powerful database privileges, the `ADMIN OPTION` is not used. Only a user currently connected as `SYSDBA` (or `INTERNAL`) can grant or revoke another user's `SYSDBA` or `SYSOPER` system privileges. These privileges cannot be granted to roles, because roles are only available after database startup. Do not confuse the `SYSDBA` and `SYSOPER` database privileges with operating system roles, which are a completely independent feature.

See Also: [Chapter 25, "Managing User Privileges and Roles"](#) for more information on system privileges

Viewing Password File Members

Use the `V$PWFILE_USERS` view to see the users who have been granted `SYSDBA` or `SYSOPER` system privileges for a database. The columns displayed by this view are as follows:

Column	Description
<code>USERNAME</code>	This column contains the name of the user that is recognized by the password file.
<code>SYSDBA</code>	If the value of this column is <code>TRUE</code> , then the user can log on with <code>SYSDBA</code> system privileges.
<code>SYSOPER</code>	If the value of this column is <code>TRUE</code> , then the user can log on with <code>SYSOPER</code> system privileges.

Maintaining a Password File

This section describes how to:

- Expand the number of password file users if the password file becomes full
- Remove the password file

- Avoid changing the state of the password file

Expanding the Number of Password File Users

If you receive the file full error (ORA-1996) when you try to grant SYSDBA or SYSOPER system privileges to a user, you must create a larger password file and re-grant the privileges to the users.

To Replace a Password File

1. Note the users who have SYSDBA or SYSOPER privileges by querying the V\$PWFFILE_USERS view.
2. Shut down the database.
3. Delete the existing password file.
4. Follow the instructions for creating a new password file using the ORAPWD utility in "Using ORAPWD" on page 1-20. Ensure that the ENTRIES parameter is set to a number larger than you think you will ever need.
5. Follow the instructions in "Adding Users to a Password File" on page 1-23.

Removing a Password File

If you determine that you no longer require a password file to authenticate users, you can delete the password file and reset the REMOTE_LOGIN_PASSWORDFILE initialization parameter to NONE. After you remove this file, only those users who can be authenticated by the operating system can perform database administration operations.

Caution: Do not remove or modify the password file if you have a database or instance mounted using REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE (or SHARED). If you do, you will be unable to reconnect remotely using the password file. Even if you replace it, you cannot use the new password file, because the timestamps and checksums will be wrong.

Changing the Password File State

The password file state is stored in the password file. When you first create a password file, its default state is SHARED. You can change the state of the password file by setting the initialization parameter REMOTE_LOGIN_PASSWORDFILE. When you start up an instance, Oracle retrieves the value of this parameter from the

parameter file stored on your client machine. When you mount the database, Oracle compares the value of this parameter to the value stored in the password file. If the values do not match, Oracle overwrites the value stored in the file.

Caution: Use caution to ensure that an `EXCLUSIVE` password file is not accidentally changed to `SHARED`. If you plan to allow instance start up from multiple clients, each of those clients must have an initialization parameter file, and the value of the parameter `REMOTE_LOGIN_PASSWORDFILE` must be the same in each of these files. Otherwise, the state of the password file could change depending upon where the instance was started.

Database Administrator Utilities

Several utilities are available to help you maintain the data in your Oracle database. This section introduces two of these utilities:

- [SQL*Loader](#)
- [Export and Import](#)

See Also: *Oracle9i Database Utilities*

SQL*Loader

SQL*Loader is used both by database administrators and by other users of Oracle. It loads data from standard operating system files (such as, files in text or C data format) into Oracle database tables.

Export and Import

The Export and Import utilities enable you to move existing data in Oracle format to and from Oracle databases. For example, export files can archive database data or move data among different Oracle databases that run on the same or different operating systems.

Creating an Oracle Database

This chapter discusses the process of creating an Oracle database, and contains the following topics:

- [Considerations Before Creating a Database](#)
- [Using the Database Configuration Assistant](#)
- [Manually Creating an Oracle Database](#)
- [Understanding the CREATE DATABASE Statement](#)
- [Troubleshooting Database Creation](#)
- [Dropping a Database](#)
- [Considerations After Creating a Database](#)
- [Initialization Parameters and Database Creation](#)
- [Managing Initialization Parameters Using a Server Parameter File](#)

See Also:

- [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating a database whose underlying operating system files are automatically created and managed by the Oracle database server
- *Oracle9i Real Application Clusters Setup and Configuration* for additional information specific to an Oracle Real Application Clusters environment

Considerations Before Creating a Database

Database creation prepares several operating system files to work together as an Oracle database. You need only create a database once, regardless of how many datafiles it has or how many instances access it. Creating a database can also erase information in an existing database and create a new database with the same name and physical structure.

The following topics can help prepare you for database creation.

- [Planning for Database Creation](#)
- [Meeting Creation Prerequisites](#)
- [Deciding How to Create an Oracle Database](#)

Planning for Database Creation

Prepare to create the database by research and careful planning. The following are some recommended actions:

Action	For more information...
<ul style="list-style-type: none"> ■ Plan the database tables and indexes and estimate the amount of space they will require. 	<p>Part II, "Oracle Server Processes and Storage Structure"</p> <p>Part III, "Schema Objects"</p>
<ul style="list-style-type: none"> ■ Plan the layout of the underlying operating system files that are to comprise your database. Proper distribution of files can improve database performance dramatically by distributing the I/O for accessing the files. There are several ways to distribute I/O when you install Oracle and create your database. For example, placing redo log files on separate disks or striping; placing datafiles to reduce contention; and controlling density of data (number of rows to a data block). 	<p><i>Oracle9i Database Performance Tuning Guide and Reference</i></p> <p>Your Oracle operating system specific documentation</p>
<ul style="list-style-type: none"> ■ Consider using the Oracle Managed Files feature to create and manage the operating system files that comprise your database storage. This feature eases their administration. 	<p>Chapter 3, "Using Oracle-Managed Files"</p>
<ul style="list-style-type: none"> ■ Select the global database name, which is the name and location of the database within the network structure. Create the global database name by setting both the DB_NAME and DB_DOMAIN initialization parameters. 	<p>"Determining the Global Database Name" on page 2-35</p>

Action	For more information...
<ul style="list-style-type: none"> ■ Familiarize yourself with the initialization parameters that comprise the initialization parameter file. Become familiar with the concept and operation of a server parameter file. A server parameter file allows you to store and manage your initialization parameters persistently in a server-side disk file. 	<p>"Initialization Parameters and Database Creation" on page 2-34</p> <p>"What is a Server Parameter File?" on page 2-44</p> <p><i>Oracle9i Database Reference</i></p>
<ul style="list-style-type: none"> ■ Select the database character set. All character data, including data in the data dictionary, is stored in the database character set. You must specify the database character set when you create the database. If clients using different character sets will access the database, then choose a superset that includes all client character sets. Otherwise, character conversions may be necessary at the cost of increased overhead and potential data loss. You can also specify an alternate character set. 	<p><i>Oracle9i Database Globalization Support Guide</i></p>
<ul style="list-style-type: none"> ■ Consider what time zones your database must support. Oracle uses a time zone file, located in the Oracle home directory, as the source of valid time zones. If you determine that you need to use a time zone that is not in the default time zone file (<code>timezone.dat</code>), but that is present in the larger time zone file (<code>timezlg.dat</code>), then you must set the <code>ORA_TZFILE</code> environment variable to point to the larger file. 	<p>"Specifying the Database Time Zone File" on page 2-28</p> <p><i>Oracle9i Database Globalization Support Guide</i></p>
<ul style="list-style-type: none"> ■ Select the standard database block size. This is specified at database creation by the <code>DB_BLOCK_SIZE</code> initialization parameter and cannot be changed after the database is created. The <code>SYSTEM</code> tablespace and most other tablespaces use the standard block size. Additionally, you can specify up to four non-standard block sizes when creating tablespaces. 	<p>"Specifying Database Block Sizes" on page 2-37</p>
<ul style="list-style-type: none"> ■ Use an undo tablespace to manage your undo records, rather than rollback segments. 	<p>Chapter 13, "Managing Undo Space"</p>

Action	For more information...
<ul style="list-style-type: none"> Develop a backup and recovery strategy to protect the database from failure. It is important to protect the control file by multiplexing, to choose the appropriate backup mode, and to manage the online and archived redo logs. 	<p>Chapter 7, "Managing the Online Redo Log"</p> <p>Chapter 8, "Managing Archived Redo Logs"</p> <p>Chapter 6, "Managing Control Files"</p> <p><i>Oracle9i Backup and Recovery Concepts</i></p>
<ul style="list-style-type: none"> Familiarize yourself with the principles and options of starting up and shutting down an instance and mounting and opening a database. 	<p>Chapter 4, "Starting Up and Shutting Down"</p>

Meeting Creation Prerequisites

To create a new database, the following prerequisites must be met:

- The desired Oracle software is installed. This includes setting up various environment variables unique to your operating system and establishing the directory structure for software and database files.
- You have the operating system privileges associated with a fully operational database administrator. You must be specially authenticated by your operating system or through a password file, allowing you to start up and shut down an instance before the database is created or opened. This authentication is discussed in "[Database Administrator Authentication](#)" on page 1-13.
- There is sufficient memory available to start the Oracle instance.
- There is sufficient disk storage space for the planned database on the computer that executes Oracle.

All of these are discussed in the Oracle installation guide specific to your operating system. Additionally, the Oracle Universal Installer will guide you through your installation and provide help in setting up environment variables, directory structure, and authorizations.

Deciding How to Create an Oracle Database

Creating a database includes the following operations:

- Creating information structures, including the data dictionary, that Oracle requires to access and use the database

- Creating and initializing the control files and redo log files for the database
- Creating new datafiles or erasing data that existed in previous datafiles

You use the `CREATE DATABASE` statement to perform these operations, but other actions are necessary before you have an operational database. A few of these actions are creating users and temporary tablespaces, building views of the data dictionary tables, and installing Oracle built-in packages. This is why the database creation process involves executing prepared scripts. But, you do not necessarily have to prepare this script yourself.

You have the following options for creating your new Oracle database:

- Use the Database Configuration Assistant (DBCA).
DBCA can be launched by the Oracle Universal Installer, depending upon the type of install that you select, and provides a graphical user interface (GUI) that guides you through the creation of a database. You can choose not to use DBCA, or you can launch it as a standalone tool at any time in the future to create a database. See ["Using the Database Configuration Assistant"](#) on page 2-5.
- Create the database manually from a script.
If you already have existing scripts for creating your database, you can still create your database manually. However, consider editing your existing script to take advantage of new Oracle features. Oracle provides a sample database creation script and a sample initialization parameter file with the database software files it distributes, both of which can be edited to suit your needs. See ["Manually Creating an Oracle Database"](#) on page 2-14.
- Upgrade an existing database.
If you are already using a previous release of Oracle, database creation is required only if you want an entirely new database. You can upgrade your existing Oracle database and use it with the new release of the Oracle software. Database upgrades are not discussed in this book. The *Oracle9i Database Migration* manual contains information about upgrading an existing Oracle database.

Using the Database Configuration Assistant

The Database Configuration Assistant (DBCA) is an Oracle-supplied tool that enables you to create an Oracle database, configure database options for an existing Oracle database, delete an Oracle database, or manage database templates. DBCA is launched automatically by the Oracle Universal Installer, but it can be invoked

standalone from the Windows operating system start menu (under Configuration Assistants) or by entering the following on the UNIX command line:

```
dbca
```

DBCA can be run in three modes:

Mode	Description
Interactive	This is the default mode if you do not specify any parameters. This mode presents a wizard like GUI interface and provides complete DBCA functionality. Online help is provided.
Progress Only	This mode is typically used by other tools to create a database. For example, it is used by the Oracle Universal Installer, the Enterprise Manager Configuration Assistant, and the Oracle Internet Directory Configuration Assistant. In this mode, only a progress bar is displayed, and it is used when creating databases and templates.
Silent	This mode has only a command-line interface where parameters are specified. There is no other user interaction. Informational, error, and warning message are written to a log file. You specify the template of your choice, for customization or for the creation of a database.

DBCA can be used to create single instance databases, or it can be used to create or add instances in an Oracle Real Application Clusters environment.

This section primarily describes the use of DBCA in interactive mode. It contains the following topics:

- [Advantages of Using DBCA](#)
- [Creating a Database Using DBCA](#)
- [Configuring Database Options](#)
- [Deleting a Database Using DBCA](#)
- [Managing DBCA Templates](#)
- [Using DBCA Silent Mode](#)

Advantages of Using DBCA

These are a few of the advantages of using DBCA:

- You can use its wizards to guide you through a selection of options providing an easy means of creating and tailoring your database. It allows you to provide varying levels of detail. You can provide a minimum of input and allow Oracle to make decisions for you, eliminating the need to spend time deciding how best to set parameters or structure the database. Optionally, it allows you to be very specific about parameter settings and file allocations.
- It builds efficient and effective databases that take advantage of Oracle's new features.
- It uses Optimal Flexible Architecture (OFA), whereby database files and administrative files, including initialization files, follow standard naming and placement practices.

Creating a Database Using DBCA

DBCA enables you to create a database from predefined templates provided by Oracle or from templates that you or others have created. A template is a description of a database. Templates are described in more detail in ["Managing DBCA Templates"](#) on page 2-9.

Selecting the Template

DBCA displays the templates that are available, which includes templates that Oracle ships with the DBCA product. These templates are described in ["DBCA Templates Provided by Oracle"](#) on page 2-11. If you or others have created templates, those will be displayed also. You select the appropriate template for the database that you want to create. Clicking the "Show Details..." button displays specific information about the database defined by a template.

Including Datafiles

When you select a template, you also specify whether the database definition is to include datafiles. This determines whether you use a seed template (includes datafiles), or a non-seed template (does not include datafiles), to create your database.

Specifying Global Database Name and Database Features

The next page that DBCA displays enables you provide a global database name and SID.

Specifying Database Features

The "Database Features" page is presented only when you select a non-seed template. It enables you to include optional database features.

The following is a representative list of Oracle features that you can install in your database. Some of the listed options might already be included depending upon the database template that you selected. Those options that are already installed are noted as such (grayed out).

- Oracle Spatial
- Oracle Ultra Search
- Oracle Label Security
- Oracle Data Mining
- Oracle OLAP Services
- Sample Schemas

You can also display a list of standard database features. These are features that Oracle recommends you always install, but you have the option of excluding them. These include:

- Oracle JVM
- Oracle Text
- Oracle *interMedia*
- XDB Protocol

Specifying Mode, Initialization Parameters, and Datafiles

The next pages enable you to further define your database. You specify mode (dedicated server or shared server), set initialization parameters, and specify datafile locations. Oracle can determine specific values for you based upon your description of the database you are trying to create. For example, Oracle can choose appropriate settings for SGA memory sizing parameters depending upon whether you select a typical or custom database.

Completing Database Creation

After you have completed the specification of the parameters that define your database you can:

- Create the database now
- Save the description as a database template
- Generate database creation scripts

If you choose to generate scripts, you can use them to create the database later without using DBCA, or you can use them as a checklist

Configuring Database Options

When you elect to configure database options, you can add Oracle options that have not previously been configured for use with your database. This provides you the opportunity to add options and features that you did not include when you created the database. These options are discussed in ["Specifying Database Features"](#) on page 2-8.

Deleting a Database Using DBCA

DBCA enables you to delete a database. When you do so, you delete the database instance and its control file(s), redo log files, and datafiles. Any server parameter file (SPFILE) or initialization parameter file used by the database is also deleted.

Managing DBCA Templates

DBCA templates are XML files that contain information required to create a database. Templates are used in DBCA to create new databases and make clones of existing databases. The information in templates includes database options, initialization parameters, and storage attributes (for datafiles, tablespaces, control files and redo logs).

Templates can be used just like scripts, and they can be used in silent mode. But they are more powerful than scripts, because you have the option of cloning a database. This saves time in database creation, because copying an already created seed database's files to the correct locations takes less time than creating them as new.

Templates are stored in the following directory:

```
$ORACLE_HOME/assistants/dbsca/templates
```

Advantages of Using Templates

The following are some of the advantages of using templates:

- They save you time. If you use a template you do not have to define the database.
- By creating a template containing your database settings, you can easily create a duplicate database without specifying parameters twice.
- They are easily edited. You can quickly change database options from the template settings.
- Templates are easy to share. They can be copied from one machine to another.

Types of Templates

There are two types of templates:

- Seed templates
- Non-seed templates

The characteristics of each are shown in the following table:

Type	File Extension	Include Datafiles?	Database Structure
Seed	.dbc	Yes	<p>This type of template contains both the structure and the physical datafiles of an existing (seed) database. When you select a seed template, database creation is faster because the physical files and schema of the database have already been created. Your database starts as a copy of the seed database, rather than having to be built.</p> <p>You can change only the following:</p> <ul style="list-style-type: none"> ■ Name of the database ■ Destination of the datafiles ■ Number control files ■ Number redo log groups ■ Initialization parameters <p>Other changes can be made after database creation using custom scripts that can be invoked by DBCA, command line SQL statements, or the Oracle Enterprise Manager.</p> <p>The datafiles and redo logs for the seed database are stored in zipped format in another file with a .dfj extension. Usually the corresponding .dfj file of a .dbc file has the same file name, but this is not a requirement since the corresponding .dfj file's location is stored in the .dbc file.</p>
Non-seed	.dbt	No	<p>This type of template is used to create a new database from scratch. It contains the characteristics of the database to be created. Seed templates are more flexible than their seed counterparts because all datafiles and redo logs are created to your specification (not copied), and names, sizes, and other attributes can be changed as required.</p>

DBCA Templates Provided by Oracle

Oracle provides templates for the following environments:

Environment	Description of Environment
DSS (Data Warehousing)	<p>Users perform numerous, complex queries that process large volumes of data. Response time, accuracy, and availability are key issues.</p> <p>These queries (typically read-only) range from a simple fetch of a few records to numerous complex queries that sort thousands of records from many different tables.</p>
OLTP (Online Transaction Processing)	<p>Many concurrent users performing numerous transactions requiring rapid access to data. Availability, speed, concurrence, and recoverability are key issues.</p> <p>Transactions consist of reading (<code>SELECT</code> statements), writing (<code>INSERT</code> and <code>UPDATE</code> statements), and deleting (<code>DELETE</code> statements) data in database tables.</p>
General Purpose	<p>This template creates a database designed for general purpose usage. It combines features of both the DSS and OLTP database templates.</p>
New Database	<p>This template allows you maximum flexibility in defining a database.</p>

Creating Templates Using DBCA

The "Template Management" page provides you with three options that enable you to modify existing templates or to create your own custom templates. Your choices are:

- From an existing template

Using an existing template, you can create a new template based on the pre-defined template settings. You can add or change any template settings such as initialization parameters, storage parameters, or use custom scripts.

- From an existing database (structure only)

You can create a new template that contains structural information about an existing database, including database options, tablespaces, datafiles, and initialization parameters specified in the source database. User defined schema and their data *will not* be part of the created template. The source database can be either local or remote.

- From an existing database (structure as well as data--a seed database)

You can create a new template that has both the structural information and physical datafiles of an existing database. Databases created using such a

template are identical to the source database. User defined schema and their data *will* be part of the created template. The source database must be local.

Oracle saves templates as XML files.

While creating templates from existing databases, you can optionally choose to translate file paths into OFA (Optimal Flexible Architecture) or maintain existing file paths. OFA is recommended if the machine on which you plan to create the database using the template has a different directory structure. Non-OFA can be used if the target machine has a similar directory structure.

Deleting DBCA Templates

The "Template Management" page also allows you to delete existing templates.

Using DBCA Silent Mode

Silent mode does not have any user interface (other than what you initially input on the command line) or user interaction. It outputs all messages including information, errors, and warnings into a log file.

From the command line enter the following command to see all of the DBCA options that are available when using silent mode:

```
dbca -help
```

The following sections contain examples that illustrate the use of silent mode.

DBCA Silent Mode Example 1: Creating a Clone Database

To create a clone database, enter the following on the command line:

```
% dbca -silent -createDatabase -templateName Transaction_Processing.dbc
-gdbname ora9i -sid ora9i -datafileJarLocation
/private/oracle9i/ora9i/assistants/dbca/templates -datafileDestination
/private/oracle9i/ora9i/oradata -responseFile NO_VALUE
-characteraset WE8ISO8859P1
```

DBCA Silent Mode Example 2: Creating a Seed Template

To create a seed template, enter the following on the command line:

```
% dbca -silent -createCloneTemplate -sourceDB ora9I -sysDBAUserName
sys -sysDBAPassword change_on_install -templateName copy_of_ora9i.dbc
-datafileJarLocation /private/oracle/ora9i/assistants/dbca/templates
```

Manually Creating an Oracle Database

This section presents the steps involved when you create a database manually. These steps should be followed in the order presented. You will previously have created your environment for creating your Oracle database, including most operating system dependent environmental variables, as part of the Oracle software installation process.

[Step 1: Decide on Your Instance Identifier \(SID\)](#)

[Step 2: Establish the Database Administrator Authentication Method](#)

[Step 3: Create the Initialization Parameter File](#)

[Step 4: Connect to the Instance](#)

[Step 5: Start the Instance.](#)

[Step 6: Issue the CREATE DATABASE Statement](#)

[Step 7: Create Additional Tablespaces](#)

[Step 8: Run Scripts to Build Data Dictionary Views](#)

[Step 9: Run Scripts to Install Additional Options \(Optional\)](#)

[Step 10: Create a Server Parameter File \(Recommended\)](#)

[Step 11: Back Up the Database.](#)

The examples shown in these steps are to create the database `mynewdb`.

Note: At this point, you may not be familiar with all of the initialization parameters and database structures discussed in this section. These steps contain many cross references to other parts of this book to allow you to learn about and understand these parameters and structures.

Step 1: Decide on Your Instance Identifier (SID)

Decide on a unique Oracle system identifier (SID) for your instance and set the `ORACLE_SID` environment variable accordingly. This identifier is used to avoid confusion with other Oracle instances that you may create later and run concurrently on your system.

The following example sets the SID for the instance and database we are about to create:

```
% setenv ORACLE_SID mynewdb
```

The value of the `DB_NAME` initialization parameter should match the SID setting.

Step 2: Establish the Database Administrator Authentication Method

You must be authenticated and granted appropriate system privileges in order to create a database. You can use the password file or operating system authentication method. Database administrator authentication and authorization is discussed in the following sections of this book:

- ["Database Administrator Security and Privileges"](#) on page 1-10
- ["Database Administrator Authentication"](#) on page 1-13
- ["Creating and Maintaining a Password File"](#) on page 1-20

Step 3: Create the Initialization Parameter File

The instance (System Global Area and background processes) for any Oracle database is started using an initialization parameter file. One way of getting started on your initialization parameter file is to edit a copy of the sample initialization parameter file that Oracle provides on the distribution media, or the sample presented in this book.

For ease of operation, store your initialization parameter file in Oracle's default location, using the default name. That way, when you start your database, it is not necessary to specify the `PFIL` parameter because Oracle automatically looks in the default location for the initialization parameter file.

Default parameter file locations are shown in the following table:

Platform	Default Name	Default Location
UNIX	<code>init\$ORACLE_SID.ora</code> For example, the initialization parameter file for the <code>mynewdb</code> database is named: <code>initmynewdb.ora</code>	<code>\$ORACLE_HOME/dbs</code> For example, the initialization parameter file for the <code>mynewdb</code> database is stored in the following location: <code>/vobs/oracle/dbs/initmynewdb.ora</code>
Windows	<code>init\$ORACLE_SID.ora</code>	<code>\$ORACLE_HOME\database</code>

The following is the initialization parameter file used to create the `mynewdb` database.

Sample Initialization Parameter File

```
# Cache and I/O
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=20971520

# Cursors and Library Cache
CURSOR_SHARING=SIMILAR
OPEN_CURSORS=300

# Diagnostics and Statistics
BACKGROUND_DUMP_DEST=/vobs/oracle/admin/mynewdb/bdumpp
CORE_DUMP_DEST=/vobs/oracle/admin/mynewdb/cdump
TIMED_STATISTICS=TRUE
USER_DUMP_DEST=/vobs/oracle/admin/mynewdb/udump

# Control File Configuration
CONTROL_FILES=( "/vobs/oracle/oradata/mynewdb/control01.ctl",
                "/vobs/oracle/oradata/mynewdb/control02.ctl",
                "/vobs/oracle/oradata/mynewdb/control03.ctl" )

# Archive
LOG_ARCHIVE_DEST_1='LOCATION=/vobs/oracle/oradata/mynewdb/archive'
LOG_ARCHIVE_FORMAT=%t_%s.dbf
LOG_ARCHIVE_START=TRUE

# Shared Server
# Uncomment and use first DISPATCHES parameter below when your listener is
# configured for SSL
# (listener.ora and sqlnet.ora)
# DISPATCHERS = "(PROTOCOL=TCPS)(SER=MODESE)",
#               "(PROTOCOL=TCPS)(PRE=oracle.aurora.server.SGiopServer)"
DISPATCHERS="(PROTOCOL=TCP)(SER=MODESE)",
              "(PROTOCOL=TCP)(PRE=oracle.aurora.server.SGiopServer)",
              (PROTOCOL=TCP)

# Miscellaneous
COMPATIBLE=9.2.0
DB_NAME=mynewdb

# Distributed, Replication and Snapshot
DB_DOMAIN=us.oracle.com
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

# Network Registration
INSTANCE_NAME=mynewdb
```

```
# Pools
JAVA_POOL_SIZE=31457280
LARGE_POOL_SIZE=1048576
SHARED_POOL_SIZE=52428800

# Processes and Sessions
PROCESSES=150

# Redo Log and Recovery
FAST_START_MTR_TARGET=300

# Resource Manager
RESOURCE_MANAGER_PLAN=SYSTEM_PLAN

# Sort, Hash Joins, Bitmap Indexes
SORT_AREA_SIZE=524288

# Automatic Undo Management
UNDO_MANAGEMENT=AUTO
UNDO_TABLESPACE=undotbs
```

See Also:

- ["Initialization Parameters and Database Creation"](#) on page 2-34 for more information on some of these parameters and other initialization parameters that you decide to include

Step 4: Connect to the Instance

Start SQL*Plus and connect to your Oracle instance AS SYSDBA.

```
$ SQLPLUS /nolog
CONNECT SYS/password AS SYSDBA
```

Step 5: Start the Instance.

Start an instance without mounting a database. Typically, you do this only during database creation or while performing maintenance on the database. Use the `STARTUP` command with the `NOMOUNT` option. In this example, because the initialization parameter file is stored in the default location, you are not required to specify the `PFILE` clause:

```
STARTUP NOMOUNT
```

At this point, there is no database. Only the SGA is created and background processes are started in preparation for the creation of a new database.

See Also:

- ["Managing Initialization Parameters Using a Server Parameter File" on page 2-44](#)
- [Chapter 4, "Starting Up and Shutting Down"](#) to learn how to use the `STARTUP` command

Step 6: Issue the `CREATE DATABASE` Statement

To create the new database, use the `CREATE DATABASE` statement. The following statement creates database `mynewdb`:

```
CREATE DATABASE mynewdb
  USER SYS IDENTIFIED BY pz6r58
  USER SYSTEM IDENTIFIED BY yltz5p
  LOGFILE GROUP 1 ('/vobs/oracle/oradata/mynewdb/redo01.log') SIZE 100M,
           GROUP 2 ('/vobs/oracle/oradata/mynewdb/redo02.log') SIZE 100M,
           GROUP 3 ('/vobs/oracle/oradata/mynewdb/redo03.log') SIZE 100M
  MAXLOGFILES 5
  MAXLOGMEMBERS 5
  MAXLOGHISTORY 1
  MAXDATAFILES 100
  MAXINSTANCES 1
  CHARACTER SET US7ASCII
  NATIONAL CHARACTER SET ALL16UTF16
  DATAFILE '/vobs/oracle/oradata/mynewdb/system01.dbf' SIZE 325M REUSE
  EXTENT MANAGEMENT LOCAL
  DEFAULT TEMPORARY TABLESPACE tempts1
     DATAFILE '/vobs/oracle/oradata/mynewdb/temp01.dbf'
     SIZE 20M REUSE
  UNDO TABLESPACE undotbs
     DATAFILE '/vobs/oracle/oradata/mynewdb/undotbs01.dbf'
     SIZE 200M REUSE AUTOEXTEND ON NEXT 5120K MAXSIZE UNLIMITED;
```

A database is created with the following characteristics:

- The database is named `mynewdb`. Its global database name is `mynewdb.us.oracle.com`. See ["DB_NAME Initialization Parameter"](#) and ["DB_DOMAIN Initialization Parameter"](#) on page 2-36.
- Three control files are created as specified by the `CONTROL_FILES` initialization parameter. See ["Specifying Control Files"](#) on page 2-36.

- The password for user SYS is pz6r58 and the password for SYSTEM is y1tz5p. These two clauses that specify the passwords for SYS and SYSTEM are not mandatory in this release of Oracle9i. However, if you specify either clause, you must specify both clauses. For further information about the use of these clauses, see "[Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM](#)" on page 2-23.
- The new database has three online redo log files as specified in the LOGFILE clause. MAXLOGFILES, MAXLOGMEMBERS, and MAXLOGHISTORY define limits for the redo log. See [Chapter 7, "Managing the Online Redo Log"](#).
- MAXDATAFILES specifies the maximum number of datafiles that can be open in the database. This number affects the initial sizing of the control file.

Note: You can set several limits during database creation. Some of these limits are also subject to superseding limits of the operating system and can be affected by them. For example, if you set MAXDATAFILES, Oracle allocates enough space in the control file to store MAXDATAFILES filenames, even if the database has only one datafile initially. However, because the maximum control file size is limited and operating system dependent, you might not be able to set all CREATE DATABASE parameters at their theoretical maximums.

For more information about setting limits during database creation, see the *Oracle9i SQL Reference* and your operating system specific Oracle documentation.

- MAXINSTANCES specifies that only one instance can have this database mounted and open.
- The US7ASCII character set is used to store data in this database.
- The AL16UTF16 character set is specified as the NATIONAL CHARACTER SET, used to store data in columns specifically defined as NCHAR, NCLOB, or NVARCHAR2.
- The SYSTEM tablespace, consisting of the operating system file /vobs/oracle/oradata/mynewdb/system01.dbf, is created as specified by the DATAFILE clause. If the file already exists, it is overwritten.
- The SYSTEM tablespace is a locally managed tablespace. See "[Creating a Locally Managed SYSTEM Tablespace](#)" on page 2-26.

- The `DEFAULT_TEMPORARY_TABLESPACE` clause creates and names a default temporary tablespace for this database. See ["Creating a Default Temporary Tablespace"](#) on page 2-24.
- The `UNDO_TABLESPACE` clause creates and names an undo tablespace to be used to store undo records for this database if you have specified `UNDO_MANAGEMENT=AUTO` in the initialization parameter file. See ["Using Automatic Undo Management: Creating an Undo Tablespace"](#) on page 2-24.
- Because the `ARCHIVELOG` clause is not specified in this `CREATE DATABASE` statement, redo log files will not initially be archived. This is customary during database creation and an `ALTER DATABASE` statement can be used later to switch to `ARCHIVELOG` mode. The initialization parameters in the initialization parameter file for `mynewdb` relating to archiving are `LOG_ARCHIVE_DEST_1`, `LOG_ARCHIVE_FORMAT`, and `LOG_ARCHIVE_START`. See [Chapter 8, "Managing Archived Redo Logs"](#).

See Also:

- ["Understanding the CREATE DATABASE Statement"](#) on page 2-22
- *Oracle9i SQL Reference* for more information about specifying the clauses and parameter values for the `CREATE DATABASE` statement

Step 7: Create Additional Tablespaces

To make the database functional, you need to create additional files and tablespaces for users. The following sample script creates some additional tablespaces:

```
CONNECT SYS/password AS SYSDBA
-- create a user tablespace to be assigned as the default tablespace for users
CREATE TABLESPACE users LOGGING
    DATAFILE '/vobs/oracle/oradata/mynewdb/users01.dbf'
    SIZE 25M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL;
-- create a tablespace for indexes, separate from user tablespace
CREATE TABLESPACE indx LOGGING
    DATAFILE '/vobs/oracle/oradata/mynewdb/indx01.dbf'
    SIZE 25M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL;
EXIT
```

For information about creating tablespaces, see [Chapter 11, "Managing Tablespaces"](#).

Step 8: Run Scripts to Build Data Dictionary Views

Run the scripts necessary to build views, synonyms, and PL/SQL packages:

```
CONNECT SYS/password AS SYSDBA
@/vobs/oracle/rdbms/admin/catalog.sql
@/vobs/oracle/rdbms/admin/catproc.sql
EXIT
```

The following table contains descriptions of the scripts:

Script	Description
CATALOG.SQL	Creates the views of the data dictionary tables, the dynamic performance views, and public synonyms for many of the views. Grants PUBLIC access to the synonyms.
CATPROC.SQL	Runs all scripts required for or used with PL/SQL.

You may want to run other scripts. The scripts that you run are determined by the features and options you choose to use or install. Many of the scripts available to you are described in the *Oracle9i Database Reference*.

See your Oracle installation guide for your operating system for the location of these scripts.

Step 9: Run Scripts to Install Additional Options (Optional)

If you plan to install other Oracle products to work with this database, see the installation instructions for those products. Some products require you to create additional data dictionary tables. Usually, command files are provided to create and load these tables into the database's data dictionary.

See your Oracle documentation for the specific products that you plan to install for installation and administration instructions.

Step 10: Create a Server Parameter File (Recommended)

Oracle recommends you create a server parameter file as a dynamic means of maintaining initialization parameters. The server parameter file is discussed in ["Managing Initialization Parameters Using a Server Parameter File"](#) on page 2-44.

The following script creates a server parameter file from the text initialization parameter file and writes it to the default location. The instance is shut down, then restarted using the server parameter file (in the default location).

```
CONNECT SYS/password AS SYSDBA
-- create the server parameter file
CREATE SPFILE='/vobs/oracle/dbs/spfilemynewdb.ora' FROM
    PFILE='/vobs/oracle/admin/mynewdb/scripts/init.ora';
SHUTDOWN
-- this time you will start up using the server parameter file
CONNECT SYS/password AS SYSDBA
STARTUP
EXIT
```

Step 11: Back Up the Database.

You should make a full backup of the database to ensure that you have a complete set of files from which to recover if a media failure occurs. For information on backing up a database, see *Oracle9i Backup and Recovery Concepts*.

Understanding the CREATE DATABASE Statement

When you execute a `CREATE DATABASE` statement, Oracle performs (at least) the following operations. The actual operations performed are in large part determined by the clauses that you specify in the `CREATE DATABASE` statement or initialization parameters that you have set.

- Creates the datafiles for the database
- Creates the control files for the database
- Creates the redo log files for the database and establishes the `ARCHIVELOG` mode.
- Creates the `SYSTEM` tablespace and the `SYSTEM` rollback segment
- Creates the data dictionary
- Sets the character set that stores data in the database
- Sets the database time zone
- Mounts and opens the database for use

This section discusses several of the clauses of the `CREATE DATABASE` statement. It expands upon some of the clauses discussed in ["Step 6: Issue the CREATE DATABASE Statement"](#) on page 2-18 and introduces additional ones.

The following topics are contained in this section:

- [Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM](#)
- [Clauses that Simplify Database Creation and Management](#)
- [Creating a Locally Managed SYSTEM Tablespace](#)
- [Specifying the Database Time Zone and Time Zone File](#)
- [Specifying FORCE LOGGING Mode](#)

Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM

The clauses of the `CREATE DATABASE` statement used for specifying the passwords for users `SYS` and `SYSTEM` are:

- `USER SYS IDENTIFIED BY password`
- `USER SYSTEM IDENTIFIED BY password`

If not specified, these users are assigned the default passwords `change_on_install` and `manager`, respectively. A record is written to the alert file indicating that the default passwords were used. To protect your database, you should change these passwords using the `ALTER USER` statement after database creation.

While these clauses are optional in this Oracle release, Oracle strongly recommends that you specify them. The default passwords are commonly known, and if you neglect to change them later, you leave yourself vulnerable to attack by malicious users.

See Also: ["Some Security Considerations"](#) on page 2-32

Clauses that Simplify Database Creation and Management

In addition to using the Database Configuration Assistant for creating your database, Oracle9i offers you other options that can simplify the creation, operation, and management of your database. These options, and their associated `CREATE DATABASE` clauses, are discussed briefly in the following sections, and in more detail in later sections of this book:

- [Using Automatic Undo Management: Creating an Undo Tablespace](#)
- [Creating a Default Temporary Tablespace](#)
- [Using Oracle-Managed Files](#)

Using Automatic Undo Management: Creating an Undo Tablespace

Oracle recommends that instead of using rollback segments in your database, you use an undo tablespace. This requires the use of a different set of initialization parameters, and optionally, the inclusion of the `UNDO TABLESPACE` clause in your `CREATE DATABASE` statement.

You must include the following initialization parameter if you want to operate your database in automatic undo management mode:

```
UNDO_MANAGEMENT=AUTO
```

In this mode, rollback information, referred to as **undo**, is stored in an undo tablespace rather than rollback segments and is managed by Oracle. If you want to create and name a specific tablespace for the undo tablespace, you can include the `UNDO TABLESPACE` clause at database creation time. If you omit this clause, and automatic undo management is specified, Oracle creates a default undo tablespace named `SYS_UNDOTBS`.

See Also:

- ["Specifying the Method of Undo Space Management"](#) on page 2-42
- [Chapter 13, "Managing Undo Space"](#) for information about the creation and use of undo tablespaces

Creating a Default Temporary Tablespace

The `DEFAULT TEMPORARY TABLESPACE` clause of the `CREATE DATABASE` statement specifies that a temporary tablespace is to be created at database creation time. This tablespace is used as the default temporary tablespace for users who are not otherwise assigned a temporary tablespace.

Users can be explicitly assigned a default temporary tablespace in the `CREATE USER` statement. But, if no temporary tablespace is specified, they default to using the `SYSTEM` tablespace. It is not good practice to store temporary data in the `SYSTEM` tablespace. To avoid this problem, and to avoid the need to assign every user a default temporary tablespace at `CREATE USER` time, you can use the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE`.

If you decide later to change the default temporary tablespace, or to create an initial one after database creation, you can do so. You do this by creating a new temporary tablespace (`CREATE TEMPORARY TABLESPACE`), then assign it as the temporary tablespace using the `ALTER DATABASE DEFAULT TEMPORARY TABLESPACE`

statement. Users will automatically be switched (or assigned) to the new temporary default tablespace.

The following statement assigns a new default temporary tablespace:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tempts2;
```

The new default temporary tablespace must be an existing temporary tablespace. When using a locally managed SYSTEM tablespace, the new default temporary tablespace must also be locally managed.

You cannot drop a default temporary tablespace, but you can assign a new default temporary tablespace, then drop the former one. You are not allowed to change a default temporary tablespace to a permanent tablespace, nor can you take a default temporary tablespace offline.

Users can obtain the name of the current default temporary tablespace using the DATABASE_PROPERTIES view. The PROPERTY_NAME column contains the value "DEFAULT_TEMP_TABLESPACE" and the PROPERTY_VALUE column contains the default temporary tablespace name.

See Also:

- *Oracle9i SQL Reference* for the syntax of the DEFAULT TEMPORARY TABLESPACE clause of CREATE DATABASE and ALTER DATABASE
- "[Temporary Tablespaces](#)" on page 11-12 for information about creating and using temporary tablespaces

Using Oracle-Managed Files

You can minimize the number of clauses and parameters that you specify in your CREATE DATABASE statement by using the Oracle Managed Files feature.

If you include the DB_CREATE_FILE_DEST or DB_CREATE_ONLINE_LOG_DEST_n initialization parameters in your initialization parameter file, you enable Oracle to create and manage the underlying operating system files of your database. Oracle will automatically create and manage the operating system files for the following database structures, dependent upon the initialization parameters you specify and how you specify clauses in your CREATE DATABASE statement:

- Tablespaces
- Temporary tablespaces
- Control files

- Online redo log files

Briefly, this is how the Oracle Managed Files feature works, using the following CREATE DATABASE statement as an example:

```
CREATE DATABASE rdb1
  USER SYS IDENTIFIED BY pz6r58
  USER SYSTEM IDENTIFIED BY y1tz5p
  UNDO TABLESPACE undotbs
  DEFAULT TEMPORARY TABLESPACE tempts1;
```

- No DATAFILE clause is specified, therefore Oracle creates an Oracle-managed datafile for the SYSTEM tablespace.
- No LOGFILE clauses are included, therefore Oracle creates two online redo log file groups that are Oracle managed.
- No DATAFILE subclause is specified for the UNDO TABLESPACE clause, therefore Oracle creates an Oracle-managed datafile for the undo tablespace.
- No TEMPFILE subclause is specified for the DEFAULT TEMPORARY TABLESPACE clause, therefore Oracle creates an Oracle-managed tempfile.
- Additionally, if no CONTROL_FILES initialization parameter is specified in the initialization parameter file, Oracle creates an Oracle-managed control file.
- If using a server parameter file (see ["Managing Initialization Parameters Using a Server Parameter File"](#) on page 2-44) the initialization parameters are set accordingly and automatically.

See Also: [Chapter 3, "Using Oracle-Managed Files"](#) for a complete description of the Oracle Managed Files feature and how to use it

Creating a Locally Managed SYSTEM Tablespace

When you specify the EXTENT MANAGEMENT LOCAL clause in the CREATE DATABASE statement, you cause Oracle to create a locally managed SYSTEM tablespace wherein Oracle determines extent sizes. The COMPATIBLE initialization parameter must be set to 9.2 or higher for this statement to be successful. If you do not specify the EXTENT MANAGEMENT LOCAL clause, the default is to create a dictionary-managed SYSTEM tablespace.

Locally managed tablespaces provide better performance and greater ease of management over dictionary-managed tablespaces. A locally managed SYSTEM tablespace is created AUTOALLOCATE by default, meaning that it is system managed with extent sizes determined and controlled by Oracle. You may notice an increase

in the initial size of objects created in a locally managed `SYSTEM` tablespace because of the autoallocate policy. It is not possible to create a locally managed `SYSTEM` tablespace and specify `UNIFORM` extent size.

When you create your database with a locally managed `SYSTEM` tablespace, ensure the following conditions are met:

- There must be a default temporary tablespace, and that tablespace cannot be the `SYSTEM` tablespace.
- You must not create rollback segments in dictionary-managed tablespaces. Attempting to create a rollback segment in a dictionary-managed tablespace will fail if the `SYSTEM` tablespace is locally managed.

To meet the first condition, you can specify the `DEFAULT TEMPORARY TABLESPACE` clause in the `CREATE DATABASE` statement, or you cannot include the clause and allow Oracle to create the tablespace for you using a default name and in a default location.

For fulfilling the second condition, Oracle recommends that instead of using rollback segments to manage the database's undo records, that you use automatic undo management. You can include the `UNDO TABLESPACE` clause in the `CREATE DATABASE` statement to create a specific undo tablespace, or if you do not include the clause, Oracle creates a locally managed undo tablespace for you using the default name and in a default location.

Note: When your `SYSTEM` tablespace is locally managed, there are restrictions on other tablespaces in the database. These restrictions are:

- You cannot create any dictionary-managed tablespaces in the database.
 - You cannot migrate a locally managed tablespace to a dictionary-managed tablespace.
 - You can transport dictionary-managed tablespaces into the database, but you are not allowed to alter them to read-write.
 - Preexisting dictionary-managed tablespaces are allowed to remain in the database, but only in `READ ONLY` mode. They cannot be altered to `READ WRITE`.
-
-

Oracle also allows you to migrate an existing dictionary-managed `SYSTEM` tablespace to a locally managed tablespace, using the `DBMS_SPACE_ADMIN` package. However, there is no procedure for backward migration.

See Also:

- *Oracle9i SQL Reference* for more specific information about the use of the `DEFAULT TEMPORARY TABLESPACE` and `UNDO TABLESPACE` clauses when `EXTENT MANAGEMENT LOCAL` is specified for the `SYSTEM` tablespace
- ["Locally Managed Tablespaces"](#) on page 11-5
- ["Migrating the SYSTEM Tablespace to a Locally Managed Tablespace"](#) on page 11-34

Specifying the Database Time Zone and Time Zone File

Oracle allows you to specify the database's default time zone, and provides you with the option of choosing the size of the supporting time zone file.

Specifying the Database Time Zone

You set the database's default time zone by specifying the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If omitted, the default database time zone is the operating system time zone. The database time zone can be changed for a session with an `ALTER SESSION` statement.

See Also: *Oracle9i Database Globalization Support Guide* for more information about setting the database time zone

Specifying the Database Time Zone File

Oracle9i enables you to specify the default time zone for your database using the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. This section provides information on the time zone files used to support this feature, specifically on Solaris platforms. Names of directories, filenames, and environment variables may differ for each platform but will probably be the same for all UNIX platforms.

The time zone files contain the valid time zone names. The following information is included for each zone (note that abbreviations are only used in conjunction with the zone names):

- Offset from UTC
- Transition times for daylight savings

- Abbreviation for standard time
- Abbreviation for daylight savings time

There are 2 time zone files under the Oracle installation directory:

- `$ORACLE_HOME/oracore/zoneinfo/timezone.dat`
This is the default. It contains the most commonly used time zones and is smaller, thus enabling better database performance.
- `$ORACLE_HOME/oracore/zoneinfo/timezlg.dat`
This file contains the larger set of defined time zones and should be used by users who require zones that are not defined in the default `timezone.dat` file. Note that this larger set of zone information may affect performance.

To enable the use of the larger time zone datafile, do the following:

1. Shut down the database.
2. Set the environment variable `ORA_TZFILE` to the full path name of the location for the `timezlg.dat` file.
3. Restart the database.

Once the larger `timezlg.dat` is used, it must continue to be used unless you are sure that none of the nondefault zones are used for data that is stored in the database. Also, all databases that share information must use the same time zone datafile.

To view the time zone names, use the following query:

```
SELECT * FROM V$TIMEZONE_NAMES;
```

Specifying FORCE LOGGING Mode

Through the use of the `NOLOGGING` clause allowed in some DDL statements (for example, `CREATE TABLE`), certain database operations will not generate redo records to the database redo log. Specifying `NOLOGGING` can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it causes problems for media recovery and for a standby database.

Oracle provides a means of forcing the writing of redo records for changes against the database, even where `NOLOGGING` has been specified in DDL statements. Oracle never generates redo records for temporary tablespaces and temporary segments, so forced logging has no affect for these.

See Also:

- *Oracle9i Database Concepts* for additional information about NOLOGGING mode
- *Oracle9i SQL Reference* for information about operations that can be done in NOLOGGING mode

Using the FORCE LOGGING Clause

To put the database into FORCE LOGGING mode, use the FORCE LOGGING clause in the CREATE DATABASE statement. If you do not specify this clause, the database is not placed into FORCE LOGGING mode.

Use the ALTER DATABASE statement to place the database into FORCE LOGGING mode after database creation. This statement can potentially wait a considerable time for completion because it waits for all unlogged direct writes to complete.

You can cancel FORCE LOGGING mode using the following SQL statement:

```
ALTER DATABASE NO FORCE LOGGING
```

Independent of specifying FORCE LOGGING for the database, you can selectively specify FORCE LOGGING or NO FORCE LOGGING at the tablespace level. However, if FORCE LOGGING mode is in effect for the database, it takes precedence over the tablespace mode setting. If it is not in effect for the database, then the individual tablespace settings are enforced. Oracle recommends that either the entire database is placed into FORCE LOGGING mode, or individual tablespaces be placed into FORCE LOGGING mode, but not both.

The FORCE LOGGING mode is a persistent attribute of the database. That is, if the database is shut down and restarted, it remains in the same logging mode state. However, if you re-create the control file, the database not restarted in the FORCE LOGGING mode unless you specify the FORCE LOGGING clause in the CREATE CONTROL FILE statement.

See Also: ["Controlling the Writing of Redo Records"](#) on page 11-20 for information about using the FORCE LOGGING clause for tablespace creation.

Performance Considerations of FORCE LOGGING Mode

There is a performance degradation for FORCE LOGGING mode. If there is no standby database active, but the primary reason for specifying FORCE LOGGING is to ensure complete media recovery, then consider the following:

- How many media failures are likely to happen?
- How serious is the damage if unlogged direct writes cannot be recovered?
- Is the performance degradation caused by forced logging tolerable?

If the database is running in `NOARCHIVELOG` mode, then generally there is no benefit to placing the database in `FORCE LOGGING` mode. This is because media recovery is not possible in this mode, thus there is performance degradation with little benefit.

Troubleshooting Database Creation

If for any reason database creation fails, shut down the instance and delete any files created by the `CREATE DATABASE` statement before you attempt to create it once again. After correcting the error that caused the failure of the database creation, try running the script again.

Dropping a Database

To drop a database, you must remove its datafiles, redo log files, and all other associated files (control files, initialization parameter files, archived log files). To view the names of the database's datafiles, redo log files, and control files, query the data dictionary views `V$DATAFILE`, `V$LOGFILE`, and `V$CONTROLFILE`, respectively.

If the database is in archive log mode, locate the archive log destinations by inspecting the initialization parameters `LOG_ARCHIVE_DEST_n`, or `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST`.

If you used the Database Configuration Assistant to create your database, you can use that tool to delete your database and clean up the files.

See Also: *Oracle9i Database Reference* for more information about these views and initialization parameters

Considerations After Creating a Database

After you create a database, the instance is left running, and the database is open and available for normal database use. You may want to perform other actions, some of which are discussed in this section.

Some Security Considerations

A newly created database has least three users that are useful for administering your database: SYS, SYSTEM and OUTLN (owner of schema where stored outlines are stored).

Caution: To prevent unauthorized access and protect the integrity of your database, the default passwords for SYS and SYSTEM should be changed immediately after the database is created.

Depending on the features and options installed, other users can also be present. Some of these users are:

- MDSYS (*interMedia Spatial*)
- ORDSYS (*interMedia Audio*)
- ORDPLUGINS (*interMedia Audio*)
- CTXSYS (Oracle Text)
- DBSNMP (Enterprise Manager Intelligent Agent)

To change the password for user DBSNMP refer to *Oracle Intelligent Agent User's Guide*.

Note Regarding Security Enhancements: In this release of Oracle and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts.

- During initial installation with the Database Configuration Assistant (DCBA), all default database user accounts except SYS, SYSTEM, SCOTT, DBSNMP, OUTLN, AURORA\$JIS\$UTILITY\$, AURORA\$ORB\$UNAUTHENTICATED and OSE\$HTTP\$ADMIN are locked and expired. To activate a locked account, the DBA must manually unlock it and reassign it a new password.
 - In addition, the DBCA prompts for passwords for users SYS and SYSTEM during initial installation of the database rather than assigning default passwords to them. A CREATE DATABASE statement issued manually also lets you supply passwords for these two users.
-
-

See Also:

- ["A Security Checklist"](#) on page 23-20 for guidance on configuring your Oracle database in a secure manner
- ["Database Administrator Usernames"](#) on page 1-11 for more information about the users `SYS` and `SYSTEM`
- ["Altering Users"](#) on page 24-6 to learn how to add new users and change passwords
- *Oracle9i SQL Reference* for the syntax of the `ALTER USER` statement used for unlocking user accounts

Installing Oracle's Sample Schemas

The Oracle database server distribution media can include various SQL files that let you experiment with the system, learn SQL, or create additional tables, views, or synonyms.

Starting with Oracle9i, Oracle provides sample schemas that enable you to become familiar with Oracle functionality. Some Oracle documents and books use these sample schemas for presenting examples. There is an ongoing effort for most Oracle books to convert to the use of Sample Schemas based examples.

The following table briefly describes the sample schemas:

Schema	Description
Human Resources	The Human Resources (HR) schema is a basic relational database schema. There are six tables in the HR schema: Employees, Departments, Locations, Countries, Jobs, and Job_History. The Order Entry (OE) schema has links into HR schema
Order Entry	The Order Entry (OE) schema builds on the purely relational Human Relations (HR) schema with some object-relational and object-oriented features. The OE schema contains seven tables: Customers, Product_Descriptions, Product_Information, Order_Items, Orders, Inventories, and Warehouses. The OE schema has links into the HR schema and PM schema. This schema also has synonyms defined on HR objects to make access transparent to users.

Schema	Description
Product Media	Product Media (PM) schema includes two tables, <code>online_media</code> and <code>print_media</code> , one object type, <code>adheader_typ</code> , and one nested table, <code>textdoc_typ</code> . The PM schema includes <i>interMedia</i> and LOB column types. Note: To use Oracle Text you must create an Oracle Text index.
Sales History	The Sales History (SH) schema is an example of a relational star schema. It consists of one big range partitioned fact table <code>sales</code> and five dimension tables: <code>times</code> , <code>promotions</code> , <code>channels</code> , <code>products</code> and <code>customers</code> . The additional <code>countries</code> table linked to <code>customers</code> shows a simple snowflake.
Queued Shipping	The Queued Shipping (QS) schema is actually multiple schemas that contain message queues.

Sample Schemas can be installed automatically for you by the Database Configuration Assistant or you can install it manually. The schemas and installation instructions are described in detail in *Oracle9i Sample Schemas*.

Initialization Parameters and Database Creation

Oracle has attempted to provide appropriate values in the starter initialization parameter file provided with your database software, or as created for you by the Database Configuration Assistant. You can edit these Oracle-supplied initialization parameters and add others, depending upon your configuration and options and how you plan to tune the database. For any relevant initialization parameters not specifically included in the initialization parameter file, Oracle supplies defaults.

If you are creating an Oracle database for the first time, it is suggested that you minimize the number of parameter values that you alter. As you become more familiar with your database and environment, you can dynamically tune many initialization parameters using the `ALTER SYSTEM` statement. If you are using a traditional text initialization parameter file, your changes are only for the current instance. To make them permanent, you must update them manually in the initialization parameter file, otherwise they will be lost over the next shutdown and startup of the database.

If you are using a server parameter file, initialization parameter file changes made by the `ALTER SYSTEM` statement can persist across shutdown and startup. This is discussed in "[Managing Initialization Parameters Using a Server Parameter File](#)" on page 2-44.

This section introduced you to some of the initialization parameters you may choose to add or edit before you create your new database.

The following topics are contained in this section:

- [Determining the Global Database Name](#)
- [Specifying Control Files](#)
- [Specifying Database Block Sizes](#)
- [Setting Initialization Parameters that Affect the Size of the SGA](#)
- [Specifying the Maximum Number of Processes](#)
- [Specifying the Method of Undo Space Management](#)
- [Setting License Parameters](#)

See Also: *Oracle9i Database Reference* for descriptions of all initialization parameters including their default settings

Determining the Global Database Name

A database's global database name consists of the local database name that you assign and its location within a network structure. The `DB_NAME` initialization parameter determines the local name component of the database's name, while the `DB_DOMAIN` parameter indicates the domain (logical location) within a network structure. The combination of the settings for these two parameters must form a database name that is unique within a network.

For example, to create a database with a global database name of `test.us.acme.com`, edit the parameters of the new parameter file as follows:

```
DB_NAME = test
DB_DOMAIN = us.acme.com
```

You can rename the `GLOBAL_NAME` of your database using the `ALTER DATABASE RENAME GLOBAL_NAME` statement. However, you must also shut down and restart the database after first changing the `DB_NAME` and `DB_DOMAIN` initialization parameters and re-creating the control file.

See Also: *Oracle9i Database Utilities* for information about using the `DBNEWID` utility, which is another means of changing a database name

DB_NAME Initialization Parameter

`DB_NAME` must be set to a text string of no more than eight characters. During database creation, the name provided for `DB_NAME` is recorded in the datafiles, redo log files, and control file of the database. If during database instance startup the value of the `DB_NAME` parameter (in the parameter file) and the database name in the control file are not the same, the database does not start.

DB_DOMAIN Initialization Parameter

`DB_DOMAIN` is a text string that specifies the network domain where the database is created. This is typically the name of the organization that owns the database. If the database you are about to create will ever be part of a distributed database system, pay special attention to this initialization parameter before database creation.

See Also: Part VI, "[Distributed Database Management](#)" for more information about distributed databases

Specifying Control Files

Include the `CONTROL_FILES` initialization parameter in your new parameter file and set its value to a list of control filenames to use for the new database. When you execute the `CREATE DATABASE` statement, the control files listed in the `CONTROL_FILES` parameter are created. If no filenames are listed for the `CONTROL_FILES` parameter, Oracle uses a default operating system dependent filename.

If you want Oracle to create new operating system files when creating your database's control files, the filenames listed in the `CONTROL_FILES` parameter must not match any filenames that currently exist on your system. If you want Oracle to reuse or overwrite existing files when creating your database's control files, ensure that the filenames listed in the `CONTROL_FILES` parameter match the filenames that are to be reused.

Caution: Use extreme caution when setting this option. If you inadvertently specify a file that you did not intend and execute the `CREATE DATABASE` statement, the previous contents of that file will be overwritten.

Oracle Corporation strongly recommends you use at least two control files stored on separate physical disk drives for each database.

See Also: [Chapter 6, "Managing Control Files"](#)

Specifying Database Block Sizes

The `DB_BLOCK_SIZE` initialization parameter specifies the standard block size for the database. This block size is used for the `SYSTEM` tablespace and by default in other tablespaces. Oracle can support up to four additional non-standard block sizes.

DB_BLOCK_SIZE Initialization Parameter

The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you need to specify. Typically, `DB_BLOCK_SIZE` is set to either 4K or 8K. If not specified, the default data block size is operating system specific, and is generally adequate.

The block size *cannot* be changed after database creation, except by re-creating the database. If a database's block size is different from the operating system block size, make the database block size a multiple of the operating system's block size.

For example, if your operating system's block size is 2K (2048 bytes), the following setting for the `DB_BLOCK_SIZE` initialization parameter is valid:

```
DB_BLOCK_SIZE=4096
```

You may want to specify a block size larger than your operating system block size. A larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Such cases include the following scenarios:

- Oracle is on a large computer system with a large amount of memory and fast disk drives. For example, databases controlled by mainframe computers with vast hardware resources typically use a data block size of 4K or greater.
- The operating system that runs Oracle uses a small operating system block size. For example, if the operating system block size is 1K and the default data block size matches this, Oracle may be performing an excessive amount of disk I/O during normal operation. For best performance in this case, a database block should consist of multiple operating system blocks.

See Also: Your operating system specific Oracle documentation for details about the default block size.

Non-Standard Block Sizes

Tablespaces of non-standard block sizes can be created using the `CREATE TABLESPACE` statement and specifying the `BLOCKSIZE` clause. These non-standard block sizes can have any power-of-two value between 2K and 32K: specifically, 2K,

4K, 8K, 16K or 32K. Platform-specific restrictions regarding the maximum block size apply, so some of these sizes may not be allowed on some platforms.

To use non-standard block sizes, you must configure sub-caches within the buffer cache area of the SGA memory for all of the non-standard block sizes that you intend to use. The initialization parameters used for configuring these sub-caches are described in the next section, "[Setting Initialization Parameters that Affect the Size of the SGA](#)".

The ability to specify multiple block sizes for your database is especially useful if you are transporting tablespaces between databases. You can, for example, transport a tablespace that uses a 4K block size from an OLTP environment to a datawarehouse environment that uses a standard block size of 8K.

See Also:

- "[Creating Tablespaces](#)" on page 11-3
- "[Transporting Tablespaces Between Databases](#)" on page 11-34

Setting Initialization Parameters that Affect the Size of the SGA

The initialization parameters discussed in this section affect the amount of memory that is allocated to the System Global Area. Except for the `SGA_MAX_SIZE` initialization parameter, they are dynamic parameters which values can be changed by the `ALTER SYSTEM` statement. The size of the SGA is dynamic, and can grow or shrink by dynamically altering these parameters.

Note: The memory for dynamic components in the SGA is allocated in the unit of granules. Granule size is determined by total SGA size. Generally speaking, on most platforms, if the total SGA size is equal to or less than 128 MB, then granule size is 4 MB. Otherwise, granule size is 16 MB.

However, there may be some platform dependency. For example, on 32-bit Windows NT, the granule size is 8 MB for SGAs larger than 128 MB. Consult your operating system specific documentation for more details.

You can query the `V$SGA_DYNAMIC_COMPONENTS` view to see the granule size that is being used by an instance. The same granule size is used for all dynamic components in the SGA.

If you specify a size for a component that is not a multiple of granule size, Oracle will round the specified size up to the nearest multiple. For example, if the granule size is 4 MB and you specify `DB_CACHE_SIZE` as 10 MB, you will actually be allocated 12 MB.

You can see a summary of information about the dynamic components of the SGA in the `V$SGA_DYNAMIC_COMPONENTS` view. Information about on-going SGA resize operations can be found in the `V$SGA_CURRENT_RESIZE_OPS` view, and information about the last 100 completed SGA resize operations can be found in the `V$SGA_RESIZE_OPS` view. To find the amount of SGA memory available for future dynamic SGA resize operations, query the `V$SGA_DYNAMIC_FREE_MEMORY` view.

See Also:

- *Oracle9i Database Performance Tuning Guide and Reference* for information about the monitoring and tuning of SGA components
- *Oracle9i Database Reference* for descriptions of the dynamic performance views used for monitoring the size of the SGA
- *Oracle9i Database Concepts* for a conceptual discussion of the SGA

Limiting the Size of the SGA

The `SGA_MAX_SIZE` initialization parameter specifies the maximum size of the System Global Area for the lifetime of the instance. You can dynamically alter the initialization parameters affecting the size of the buffer caches, shared pool, and large pool, but only to the extent that the sum of these sizes and the sizes of the other components of the SGA (fixed SGA, variable SGA, and redo log buffers) does not exceed the value specified by `SGA_MAX_SIZE`.

If you do not specify `SGA_MAX_SIZE`, then Oracle selects a default value that is the sum of all components specified or defaulted at initialization time.

Setting the Buffer Cache Initialization Parameters

The buffer cache initialization parameters determine the size of the buffer cache component of the SGA. You use them to specify the sizes of caches for the various block sizes used by the database. These initialization parameters are all dynamic.

If you intend to use multiple block sizes in your database, you must have the `DB_CACHE_SIZE` and at least one `DB_nK_CACHE_SIZE` parameter set. Oracle assigns an appropriate default value to the `DB_CACHE_SIZE` parameter, but the `DB_nK_CACHE_SIZE` parameters default to 0, and no additional block size caches are configured.

The size of a buffer cache affects performance. Larger cache sizes generally reduce the number of disk reads and writes. However, a large cache may take up too much memory and induce memory paging or swapping.

DB_CACHE_SIZE Initialization Parameter The `DB_CACHE_SIZE` initialization parameter replaces the `DB_BLOCK_BUFFERS` initialization parameter that was used in previous releases. The `DB_CACHE_SIZE` parameter specifies the size of the cache of standard block size buffers, where the standard block size is specified by `DB_BLOCK_SIZE`.

For backward compatibility the `DB_BLOCK_BUFFERS` parameter will still work, but it remains a static parameter and cannot be combined with any of the dynamic sizing parameters.

DB_nK_CACHE_SIZE Initialization Parameters The sizes and numbers of non-standard block size buffers are specified by the following initialization parameters:

- `DB_2K_CACHE_SIZE`
- `DB_4K_CACHE_SIZE`
- `DB_8K_CACHE_SIZE`

- `DB_16K_CACHE_SIZE`
- `DB_32K_CACHE_SIZE`.

Each parameter specifies the size of the buffer cache for the corresponding block size. For example:

```
DB_BLOCK_SIZE=4096
```

```
DB_CACHE_SIZE=12M
```

```
DB_2K_CACHE_SIZE=8M
```

```
DB_8K_CACHE_SIZE=4M
```

In the above example, the parameters specify that the standard block size of the database will be 4K. The size of the cache of standard block size buffers will be 12M. Additionally, 2K and 8K caches will be configured with sizes of 8M and 4M respectively.

Note: These parameters cannot be used to size the cache for the standard block size. For example, if the value of `DB_BLOCK_SIZE` is 2K, it is illegal to set `DB_2K_CACHE_SIZE`. The size of the cache for the standard block size is always determined from the value of `DB_CACHE_SIZE`.

Adjusting the Size of the Shared Pool

The `SHARED_POOL_SIZE` initialization parameter is a dynamic parameter that allows you to specify or adjust the size of the shared pool component of the SGA. Oracle selects an appropriate default value.

Adjusting the Size of the Large Pool

The `LARGE_POOL_SIZE` initialization parameter is a dynamic parameter that allows you to specify or adjust the size of the large pool component of the SGA. Oracle selects an appropriate default value.

Specifying the Maximum Number of Processes

The `PROCESSES` initialization parameter determines the maximum number of operating system processes that can be connected to Oracle concurrently. The value of this parameter must be 6 or greater (5 for the background processes plus 1 for each user process). For example, if you plan to have 50 concurrent users, set this parameter to at least 55.

Specifying the Method of Undo Space Management

Every Oracle database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Oracle refers to these records collectively as **undo**. Oracle allows you to store undo in an undo tablespace or in rollback segments.

See Also: [Chapter 13, "Managing Undo Space"](#)

UNDO_MANAGEMENT Initialization Parameter

The `UNDO_MANAGEMENT` initialization parameter determines whether an instance will start up in automatic undo management mode, where undo is stored in an undo tablespace, or manual undo management mode, where undo is stored in rollback segments. A value of `AUTO` enables automatic undo management mode, `MANUAL` enables manual undo management mode. For backward compatibility, the default is `MANUAL`.

UNDO_TABLESPACE Initialization Parameter

When the instance starts up in automatic undo management mode, it selects the first available undo tablespace in the instance for storing undo. A default undo tablespace named `SYS_UNDOTBS` is automatically created when you execute a `CREATE DATABASE` statement and the `UNDO_MANAGEMENT` initialization parameter is set to `AUTO`. This is the undo tablespace that Oracle normally selects whenever you start up the database.

Optionally, you can specify the `UNDO_TABLESPACE` initialization parameter. This causes the instance to use the undo tablespace specified by the parameter. The `UNDO_TABLESPACE` parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

If there is no undo tablespace available, the instance will start, but uses the `SYSTEM` rollback segment. This is not recommended in normal circumstances, and an alert message is written to the alert file to warn that the system is running without an undo tablespace.

Oracle recommends using an undo tablespace rather than rollback segments. An undo tablespace is easier to administer and enables you to explicitly set an undo retention time.

ROLLBACK_SEGMENTS Initialization Parameter

The `ROLLBACK_SEGMENTS` parameter is a list of the non-system rollback segments an Oracle instance acquires at database startup if the database is to operate in manual undo management mode. List your rollback segments as the value of this parameter. If no rollback segments are specified, the system rollback segment is used.

The `ROLLBACK_SEGMENTS` initialization parameter is supported for backward compatibility. Oracle recommends using an undo tablespace rather than rollback segments.

Setting License Parameters

Note: Oracle no longer offers licensing by the number of concurrent sessions. Therefore the `LICENSE_MAX_SESSIONS` and `LICENSE_SESSIONS_WARNING` initialization parameters have been deprecated and are no longer discussed in this book.

If you use named user licensing, Oracle can help you enforce this form of licensing. You can set a limit on the number of users created in the database. Once this limit is reached, you cannot create more users.

Note: This mechanism assumes that each person accessing the database has a unique user name and that no people share a user name. Therefore, so that named user licensing can help you ensure compliance with your Oracle license agreement, do not allow multiple users to log in using the same user name.

To limit the number of users created in a database, set the `LICENSE_MAX_USERS` initialization parameter in the database's initialization parameter file, as shown in the following example:

```
LICENSE_MAX_USERS = 200
```

Managing Initialization Parameters Using a Server Parameter File

Oracle has traditionally stored initialization parameters in a text initialization parameter file. Starting with Oracle9i, you can choose to maintain initialization parameters in a binary server parameter file.

This section introduces the server parameter file, and explains how to manage initialization parameters using either method of storing the parameters. The following topics are contained in this section.

- [What is a Server Parameter File?](#)
- [Migrating to a Server Parameter File](#)
- [Creating a Server Parameter File](#)
- [The SPFILE Initialization Parameter](#)
- [Using ALTER SYSTEM to Change Initialization Parameter Values](#)
- [Exporting the Server Parameter File](#)
- [Backing Up the Server Parameter File](#)
- [Errors and Recovery for the Server Parameter File](#)
- [Viewing Parameter Settings](#)

What is a Server Parameter File?

A **server parameter file** (SPFILE) can be thought of as a repository for initialization parameters that is maintained on the machine where the Oracle database server executes. It is, by design, a server-side initialization parameter file. Initialization parameters stored in a server parameter file are persistent, in that any changes made to the parameters while an instance is running can persist across instance shutdown and startup. This eliminates the need to manually update initialization parameters to make changes effected by `ALTER SYSTEM` statements persistent. It also provides a basis for self tuning by the Oracle database server.

A server parameter file is initially built from a traditional text initialization parameter file using the `CREATE SPFILE` statement. It is a binary file that cannot be browsed or edited using a text editor. Oracle provides other interfaces for viewing and modifying parameter settings.

Caution: Although you can open the binary server parameter file with a text editor and view its text, *do not* manually edit it. Doing so will corrupt the file. You will not be able to start you instance, and if the instance is running, it could fail.

At system startup, the default behavior of the `STARTUP` command is to read a server parameter file to obtain initialization parameter settings. The `STARTUP` command with no `PFILE` clause, reads the server parameter file from an operating system specific location. If you choose to use the traditional text initialization parameter file, you must specify the `PFILE` clause when issuing the `STARTUP` command. Explicit instructions for starting an instance using a server parameter file are contained in [Starting Up a Database](#) on page 4-2.

Migrating to a Server Parameter File

If you are currently using a traditional initialization parameter file, use the following steps to migrate to a server parameter file.

1. If the initialization parameter file is located on a client machine, transfer the file (for example, FTP) from the client machine to the server machine.

Note: If you are using Oracle9i Real Application Clusters, you must combine all of your instance specific initialization parameter files into a single initialization parameter file. Instructions for doing this, and other actions unique to using a server parameter file for Oracle Real Application Cluster instances, are discussed in:

- *Oracle9i Real Application Clusters Setup and Configuration*
 - *Oracle9i Real Application Clusters Administration*
-
-

2. Create a server parameter file using the `CREATE SPFILE` statement. This statement reads the initialization parameter file to create a server parameter file. The database does not have to be started to issue a `CREATE SPFILE` statement.
3. Start up the instance using the newly created server parameter file.

Creating a Server Parameter File

The server parameter file must initially be created from a traditional text initialization parameter file. It must be created prior to its use in the `STARTUP` command. The `CREATE SPFILE` statement is used to create a server parameter file. You must have the `SYSDBA` or the `SYSOPER` system privilege to execute this statement.

The following example creates a server parameter file from initialization parameter file `/u01/oracle/dbs/init.ora`. In this example no `SPFILE` name is specified, so the file is created in a platform-specific default location and is named `spfile$ORACLE_SID.ora`.

```
CREATE SPFILE FROM PFILE='/u01/oracle/dbs/init.ora';
```

Another example, below, illustrates creating a server parameter file and supplying a name.

```
CREATE SPFILE='/u01/oracle/dbs/test_spfile.ora'  
FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

The server parameter file is always created on the machine running the database server. If a server parameter file of the same name already exists on the server, it is overwritten with the new information.

Oracle recommends that you allow the database server to default the name and location of the server parameter file. This will ease administration of your database. For example, the `STARTUP` command assumes this default location to read the parameter file.

When the server parameter file is created from the initialization parameter file, comments specified on the same lines as a parameter setting in the initialization parameter file are maintained in the server parameter file. All other comments are ignored.

The `CREATE SPFILE` statement can be executed before or after instance startup. However, if the instance has been started using a server parameter file, an error is raised if you attempt to re-create the same server parameter file that is currently being used by the instance.

Note: When you use the Database Configuration Assistant to create a database, it can automatically create a server parameter file for you.

The SPFILE Initialization Parameter

The `SPFILE` initialization parameter contains the name of the current server parameter file. When the default server parameter file is used by the server (that is, you issue a `STARTUP` command and do not specify a `PFILE`), the value of `SPFILE` is internally set by the server. The SQL*Plus command `SHOW PARAMETERS SPFILE` (or any other method of querying the value of a parameter) displays the name of the server parameter file that is currently in use.

The `SPFILE` parameter can also be set in a traditional parameter file to indicate the server parameter file to use. You use the `SPFILE` parameter to specify a server parameter file located in a nondefault location. *Do not* use an `IFILE` initialization parameter within a traditional initialization parameter file to point to a server parameter file; instead, use the `SPFILE` parameter. See "[Starting Up a Database](#)" on page 4-2 for details about:

- Starting up a database that uses a server parameter file
- Using the `SPFILE` parameter to specify the name of a server parameter file to use at instance startup

Using ALTER SYSTEM to Change Initialization Parameter Values

The `ALTER SYSTEM` statement allows you to set, change, or delete (restore to default value) initialization parameter values. When the `ALTER SYSTEM` statement is used to alter a parameter setting in a traditional initialization parameter file, the change affects only the current instance, since there is no mechanism for automatically updating initialization parameters on disk. They must be manually updated in order to be passed to a future instance. Using a server parameter file overcomes this limitation.

Setting or Changing Initialization Parameter Values

Use the `SET` clause of the `ALTER SYSTEM` statement to set or change initialization parameter values. Additionally, the `SCOPE` clause specifies the scope of a change as described in the following table:

SCOPE Clause	Description
<code>SCOPE = SPFILE</code>	The change is applied in the server parameter file only. The effect is as follows: <ul style="list-style-type: none"> ■ For dynamic parameters, the change is effective at the next startup and is persistent. ■ For static parameters, the behavior is the same as for dynamic parameters. This is the only <code>SCOPE</code> specification allowed for static parameters.
<code>SCOPE = MEMORY</code>	The change is applied in memory only. The effect is as follows: <ul style="list-style-type: none"> ■ For dynamic parameters, the effect is immediate, but it is not persistent because the server parameter file is not updated. ■ For static parameters, this specification is not allowed.
<code>SCOPE = BOTH</code>	The change is applied in both the server parameter file and memory. The effect is as follows: <ul style="list-style-type: none"> ■ For dynamic parameters, the effect is immediate and persistent. ■ For static parameters, this specification is not allowed.

It is an error to specify `SCOPE=SPFILE` or `SCOPE=BOTH` if the server is not using a server parameter file. The default is `SCOPE=BOTH` if a server parameter file was used to start up the instance, and `MEMORY` if a traditional initialization parameter file was used to start up the instance.

For dynamic parameters, you can also specify the `DEFERRED` keyword. When specified, the change is effective only for future sessions.

A `COMMENT` clause allows a comment string to be associated with the parameter update. When you specify `SCOPE` as `SPFILE` or `BOTH`, the comment is written to the server parameter file.

The following statement changes the maximum number of job queue processes allowed for the instance. It also specifies a comment, and explicitly states that the change is to be made only in memory (that is, it is not persistent across instance shutdown and startup).

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=50
                COMMENT='temporary change on Nov 29'
                SCOPE=MEMORY;
```

Another example illustrates setting a complex initialization parameter that takes a list of strings. Specifically, the parameter value being set is the `LOG_ARCHIVE_DEST_n` initialization parameter. The case could be that either the parameter is being changed to a new value or a new archive destination is being added.

```
ALTER SYSTEM
  SET LOG_ARCHIVE_DEST_4='LOCATION=/u02/oracle/rbdb1/' ,MANDATORY, 'REOPEN=2'
  COMMENT='Add new destination on Nov 29'
  SCOPE=SPFILE;
```

Note that when a value consists of a list of strings, the syntax of the `ALTER SYSTEM SET` statement does not support editing each element of the list of values by the position or ordinal number. You must specify the complete list of values each time the parameter is updated, and the new list completely replaces the old list.

Deleting Initialization Parameter Values

For initialization parameters whose values are string values you can restore a parameter to its default value (effectively deleting it), by using the following syntax:

```
ALTER SYSTEM SET parameter = '';
```

For numeric and boolean value parameters, you must specifically set the parameter back to its original default value.

Exporting the Server Parameter File

You can export a server parameter file to create a traditional text initialization parameter file. Reasons for doing this include:

- Creating backups of the server parameter file
- For diagnostic purposes, listing all of the parameter values currently used by an instance. This is analogous to the `SQL*Plus SHOW PARAMETERS` command or selecting from the `V$PARAMETER` or `V$PARAMETER2` views.
- Modifying of the server parameter file by first exporting it, editing the output file, and then re-creating it

The exported file can also be used to start up an instance using the `PFILE` option.

The `CREATE PFILE` statement is used to export a server parameter file. You must have the `SYSDBA` or the `SYSOPER` system privilege to execute this statement. The exported file is created on the database server machine. It contains any comments associated with the parameter in the same line as the parameter setting.

The following example creates a text initialization parameter file from the server parameter file:

```
CREATE PFILE FROM SPFILE;
```

Because no names were specified for the files, a platform-specific name is used for the initialization parameter file, and it is created from the platform-specific default server parameter file.

The following example creates a text initialization parameter file from a server parameter file where the names of the files are specified:

```
CREATE PFILE='/u01/oracle/dbs/test_init.ora'  
FROM SPFILE='/u01/oracle/dbs/test_spfile.ora';
```

Backing Up the Server Parameter File

You can create a backup of your server parameter file by exporting it, as described in "[Exporting the Server Parameter File](#)" on page 2-49. If the backup and recovery strategy for your database is implemented using Recovery Manager (RMAN), then you can use RMAN to create a backup. The server parameter file is backed up automatically by RMAN when you back up your database, but RMAN also allows you to specifically create a backup of the currently active server parameter file.

See Also: Oracle9i Recovery Manager User's Guide

Errors and Recovery for the Server Parameter File

If an error occurs while reading the server parameter file (during startup or an export operation), or while writing the server parameter file during its creation, the operation terminates with an error reported to the user.

If an error occurs while reading or writing the server parameter file during a parameter update, the error is reported in the alert file and all subsequent parameter updates to the server parameter file are ignored. At this point, you have the following options:

- Shut down the instance, recover the server parameter file, then restart the instance
- Continue to run without caring that subsequent parameter updates will not be persistent

Viewing Parameter Settings

You have several options for viewing parameter settings.

Method	Description
SHOW PARAMETERS	This SQL*Plus command displays the currently in use parameter values.
CREATE PFILE	This SQL statement creates a text initialization parameter file from the binary server parameter file.
V\$PARAMETER	This view displays the currently in effect parameter values.
V\$PARAMETER2	This view displays the currently in effect parameter values. It is easier to distinguish list parameter values in this view because each list parameter value appears as a row.
V\$SPPARAMETER	This view displays the current contents of the server parameter file. The view returns NULL values if a server parameter file is not being used by the instance.

See Also: *Oracle9i Database Reference* for a complete description of views

Using Oracle-Managed Files

This chapter discusses the use of the Oracle-managed files and contains the following topics:

- [What are Oracle-Managed Files?](#)
- [Enabling the Creation and Use of Oracle-Managed Files](#)
- [Creating Oracle-Managed Files](#)
- [Behavior of Oracle-Managed Files](#)
- [Scenarios for Using Oracle-Managed Files](#)

What are Oracle-Managed Files?

Using Oracle-managed files simplifies the administration of an Oracle database. Oracle-managed files eliminate the need for you, the DBA, to directly manage the operating system files comprising an Oracle database. You specify operations in terms of database objects rather than filenames. Oracle internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Online redo log files
- Control files

Through initialization parameters, you specify the file system directory to be used for a particular type of file. Oracle then ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed.

This feature does not affect the creation or naming of administrative files such as trace files, audit files, alert files, and core files.

Who Can Use Oracle-Managed Files?

Oracle-managed files are most useful for the following types of databases:

- Databases that are supported by the following:
 - A logical volume manager that supports striping/RAID and dynamically extensible logical volumes
 - A file system that provides large, extensible files
- Low end or test databases

The Oracle Managed Files feature is not intended to ease administration of systems that use raw disks. This feature provides better integration with operating system functionality for disk space allocation. Since there is no operating system support for allocation of raw disks (it is done manually), this feature cannot help. On the other hand, because Oracle-managed files require that you use the operating system file system (unlike raw disks), you lose control over how files are laid out on the disks and thus, you lose some I/O tuning ability.

What is a Logical Volume Manager?

A logical volume manager (LVM) is a software package available with most operating systems. Sometimes it is called a logical disk manager (LDM). It allows

pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software. An LVM can make the logical volume have better capacity, performance, reliability, and availability characteristics than any of the underlying physical disks. It uses techniques such as mirroring, striping, concatenation, and RAID 5 to implement these characteristics.

Some LVMs allow the characteristics of a logical volume to be changed after it is created, even while it is in use. The volume may be resized or mirrored, or it may be relocated to different physical disks.

What is a File System?

A file system is a data structure built inside a contiguous disk address space. A file manager (FM) is a software package that manipulates file systems, but it is sometimes called the file system. All operating systems have file managers. The primary task of a file manager is to allocate and deallocate disk space into files within a file system.

A file system allows the disk space to be allocated to a large number of files. Each file is made to appear as a contiguous address space to applications such as Oracle. The files may not actually be contiguous within the disk space of the file system. Files can be created, read, written, resized, and deleted. Each file has a name associated with it that is used to refer to the file.

A file system is commonly built on top of a logical volume constructed by an LVM. Thus all the files in a particular file system have the same performance, reliability, and availability characteristics inherited from the underlying logical volume. A file system is a single pool of storage that is shared by all the files in the file system. If a file system is out of space, then none of the files in that file system can grow. Space available in one file system does not affect space in another file system. However some LVM/FM combinations allow space to be added or removed from a file system.

An operating system can support multiple file systems. Multiple file systems are constructed to give different storage characteristics to different files as well as to divide the available disk space into pools that do not affect each other.

Benefits of Using Oracle-Managed Files

Consider the following benefits of using Oracle-managed files:

- They make the administration of the database easier.

There is no need to invent filenames and define specific storage requirements. A consistent set of rules is used to name all relevant files. The file system defines the characteristics of the storage and the pool where it is allocated.

- They reduce corruption caused by administrators specifying the wrong file.

Each Oracle-managed file and filename is unique. Using the same file in two different databases is a common mistake that can cause very large down times and loss of committed transactions. Using two different names that refer to the same file is another mistake that causes major corruptions.

- They reduce wasted disk space consumed by obsolete files.

Oracle automatically removes old Oracle-managed files when they are no longer needed. Much disk space is wasted in large systems simply because no one is sure if a particular file is still required. This also simplifies the administrative task of removing files that are no longer required on disk and prevents the mistake of deleting the wrong file.

- They simplify creation of test and development databases.

You can minimize the time spent making decisions regarding file structure and naming, and you have fewer file management tasks. You can focus better on meeting the actual requirements of your test or development database.

- Oracle-managed files make development of portable third-party tools easier.

Oracle-managed files eliminate the need to put operating system specific file names in SQL scripts.

Oracle-Managed Files and Existing Functionality

Using Oracle-managed files does not eliminate any existing functionality. Existing databases are able to operate as they always have. New files can be created as managed files while old ones are administered in the old way. Thus, a database can have a mixture of Oracle-managed and unmanaged files.

Enabling the Creation and Use of Oracle-Managed Files

The following initialization parameters allow the database server to use the Oracle Managed Files feature:

Parameter	Description
DB_CREATE_FILE_DEST	Defines the location of the default file system directory where Oracle creates datafiles or tempfiles when no file specification is given in the creation operation. Also used as the default file system directory for online redo log and control files if DB_CREATE_ONLINE_LOG_DEST_1 is not specified.
DB_CREATE_ONLINE_LOG_DEST_n	Defines the location of the default file system directory for online redo log files and control file creation when no file specification is given in the creation operation. You can use this initialization parameter multiple times, where <i>n</i> specifies a multiplexed copy of the online redo log or control file. You can specify up to five multiplexed copies.

The file system directory specified by either of these parameters must already exist: Oracle does not create it. The directory must also have permissions to allow Oracle to create the files in it.

The default location is used whenever a location is not explicitly specified for the operation creating the file. Oracle creates the filename, and a file thus created is an Oracle-managed file.

Both of these initialization parameters are dynamic, and can be set using the ALTER SYSTEM or ALTER SESSION statement.

See Also:

- *Oracle9i Database Reference* for additional information about initialization parameters
- ["How Oracle-Managed Files are Named"](#) on page 3-7

Setting the DB_CREATE_FILE_DEST Initialization Parameter

Include the DB_CREATE_FILE_DEST initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Datafiles
- Tempfiles
- Online redo log files
- Control files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. The following example sets `/u01/oradata/payroll` as the default directory to use when creating Oracle-managed files:

```
DB_CREATE_FILE_DEST = '/u01/oradata/payroll'
```

Setting the `DB_CREATE_ONLINE_LOG_DEST_n` Initialization Parameter

Include the `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameter in your initialization parameter file to identify the default location for the database server to create:

- Online redo log files
- Control files

You specify the name of a file system directory that becomes the default location for the creation of the operating system files for these entities. You can specify up to five multiplexed locations.

For the creation of online redo log files and control files only, this parameter overrides any default location specified in the `DB_CREATE_FILE_DEST` initialization parameter. If you do not specify a `DB_CREATE_FILE_DEST` parameter, but you do specify this parameter, then only online redo log files and control files can be created as Oracle-managed files.

It is recommended that you specify at least two parameters. For example:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/payroll'  
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/payroll'
```

This allows multiplexing, which provides greater fault-tolerance for the online redo log and control file if one of the destinations fails.

Creating Oracle-Managed Files

If you have met any of the following conditions, then Oracle creates Oracle-managed files for you, as appropriate, when no file specification is given in the creation operation:

- You have included either or both of the `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters in your initialization parameter file.

- You have issued the `ALTER SYSTEM` or `ALTER SESSION` statement to dynamically set either or both the `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters.

If a statement that creates an Oracle-managed file finds an error or does not complete due to some failure, then any Oracle-managed files created by the statement are automatically deleted as part of the recovery of the error or failure. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands. When an Oracle-managed file is created, its filename is written to the alert file. This information can be used to find the file if it is necessary to manually remove the file.

The following topics are discussed in this section:

- [How Oracle-Managed Files are Named](#)
- [Creating Oracle-Managed Files at Database Creation](#)
- [Creating Datafiles for Tablespaces](#)
- [Creating Tempfiles for Temporary Tablespaces](#)
- [Creating Control Files](#)
- [Creating Online Redo Log Files](#)

See Also: *Oracle9i SQL Reference*

How Oracle-Managed Files are Named

The filenames of Oracle-managed files comply with the Oracle Flexible Architecture (OFA) standard for file naming. The assigned names are intended to meet the following requirements:

- Database files are easily distinguishable from all other files.
- Control files, online redo log files, and datafiles are identifiable as such.
- The association of datafile to tablespace is clearly indicated.

No two Oracle-managed files are given the same name. The name that is used for creation of an Oracle-managed file is constructed from three sources.

- The default file system directory location
- A port-specific file name template that is chosen based on the type of file

- A unique string created by the Oracle database server or the operating system. This ensures that file creation does not damage an existing file and that the file cannot be mistaken for some other file.

As a specific example, filenames for Oracle-managed files have the following format on Solaris:

File Type	Format	Example
Datafile	o1_mf_%t_%u_.dbf	/u01/oradata/payroll/o1_mf_tbs1_2ixfh90q_.dbf
Tempfile	o1_mf_%t_%u_.tmp	/u01/oradata/payroll/o1_mf_temp1_6dygh80r_.tmp
Redo log file	o1_mf_%g_%u_.log	/u01/oradata/payroll/o1_mf_1_wo94n2xi_.log
Control file	o1_mf_%u_.ctl	/u01/oradata/payroll/o1_mf_cmr7t30p_.ctl

where:

- %t is the tablespace name. At most, eight characters of the tablespace name are used. If eight characters causes the name to be too long, then the tablespace name is truncated. Placing the tablespace name before the uniqueness string means that all the datafiles for a tablespace appear next to each other in an alphabetic file listing.
- %u is an eight character string that guarantees uniqueness
- %g is the online redo log file group number

On other platforms the names are similar, subject to the constraints of the platform's naming rules.

Caution: Do not rename an Oracle-managed file. Oracle identifies an Oracle-managed file based on its name. If you rename the file, Oracle is no longer able to recognize it as an Oracle-managed file and will not manage the file accordingly.

Creating Oracle-Managed Files at Database Creation

The behavior of the `CREATE DATABASE` statement for creating database structures when using Oracle-managed files is discussed in this section.

Specifying Control Files at Database Creation

At database creation, the control file is created in the files specified by the `CONTROL_FILES` initialization parameter. If the `CONTROL_FILES` parameter is not

set and at least one of the initialization parameters required for the creation of Oracle-managed files is set, then an Oracle-managed control file is created in the default control file destinations. In order of precedence, the default destination is defined as follows:

- If `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed control file copy is created in each directory specified. The file in the first directory is the primary control file.
- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, and no `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed control file is created in the directory specified.

If the `CONTROL_FILES` parameter is not set and none of the above initialization parameters are set, then Oracle's default behavior is operating system dependent. At least one copy of a control file is created in an operating system dependent default location. Any copies of control files created in this fashion are not Oracle-managed files, and you must add a `CONTROL_FILES` initialization parameter to any initialization parameter file.

If Oracle creates an Oracle-managed control file, and if there is a server parameter file, then Oracle creates a `CONTROL_FILES` initialization parameter entry in the server parameter file. If there is no server parameter file, then you must manually include a `CONTROL_FILES` initialization parameter entry in the text initialization parameter file.

See Also: [Chapter 6, "Managing Control Files"](#)

Specifying Online Redo Log Files at Database Creation

The `LOGFILE` clause is not required in the `CREATE DATABASE` statement, and omitting it provides a simple means of creating Oracle-managed online redo log files. If the `LOGFILE` clause is omitted, then online redo log files are created in the default online redo log file destinations. In order of precedence, the default destination is defined as follows:

- If `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then Oracle creates two online redo files in each directory specified. More specifically, Oracle creates two online redo groups with corresponding members in each directory specified. These online redo log files are Oracle-managed files.
- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, but no `DB_CREATE_ONLINE_LOG_DEST_n` parameters are specified, then two online redo log files (two groups with one member each) are created in the directory specified. These online redo log files are Oracle-managed files.

- If the `LOGFILE` clause is omitted and neither of the above initialization parameters are specified, then two online redo log files are created in operating system dependent default locations. Any online redo log files created in this fashion are not Oracle-managed files.

The default size of an Oracle-managed online redo log file is 100 (MB).

Optionally, you can create Oracle-managed online redo log files, and override default attributes, by including the `LOGFILE` clause but omitting a filename. Online redo log files are created as above, except for the following: If no filename is provided in the `LOGFILE` clause of `CREATE DATABASE`, and none of the initialization parameters required for creating Oracle-managed files are provided, then the `CREATE DATABASE` statement fails.

See Also: [Chapter 7, "Managing the Online Redo Log"](#)

Specifying the `SYSTEM` Tablespace Datafile at Database Creation

The `DATAFILE` clause is not required in the `CREATE DATABASE` statement, and omitting it provides a simple means of creating an Oracle-managed datafile for the `SYSTEM` tablespace. If the `DATAFILE` clause is omitted, then one of the following actions occurs:

- If `DB_CREATE_FILE_DEST` is set, then an Oracle-managed datafile for the `SYSTEM` tablespace is created in the `DB_CREATE_FILE_DEST` directory.
- If `DB_CREATE_FILE_DEST` is not set, then Oracle creates one `SYSTEM` tablespace datafile whose name and size are operating system dependent. Any `SYSTEM` tablespace datafile created in this manner is not an Oracle-managed file.

The default size for an Oracle-managed datafile is 100 MB and the file is autoextensible with an unlimited maximum size.

Optionally, you can create an Oracle-managed datafile for the `SYSTEM` tablespace and override default attributes. This is done by including the `DATAFILE` clause, omitting a filename, but specifying overriding attributes. When a filename is not supplied and the `DB_CREATE_FILE_DEST` parameter is set, an Oracle-managed datafile for the `SYSTEM` tablespace is created in the `DB_CREATE_FILE_DEST` directory with the specified attributes being overridden. However, if a filename is not supplied and the `DB_CREATE_FILE_DEST` parameter is not set, then the `CREATE DATABASE` statement fails.

When overriding the default attributes of an Oracle-managed file, if a `SIZE` value is specified but no `AUTOEXTEND` clause is specified, then the datafile is *not* autoextensible.

Specifying the Undo Tablespace Datafile at Database Creation

The `DATAFILE` subclause of the `UNDO TABLESPACE` clause is optional and a filename is not required in the file specification. If a filename is not supplied and the `DB_CREATE_FILE_DEST` parameter is set, then an Oracle-managed datafile is created in the `DB_CREATE_FILE_DEST` directory. If `DB_CREATE_FILE_DEST` is not set, then the statement fails with a syntax error.

The `UNDO TABLESPACE` clause itself is optional in the `CREATE DATABASE` statement. If it is not supplied, and automatic undo management mode is enabled, then a default undo tablespace named `SYS_UNDOTBS` is created and a 10 MB datafile that is autoextensible is allocated as follows:

- If `DB_CREATE_FILE_DEST` is set, then an Oracle-managed datafile is created in the indicated directory.
- If `DB_CREATE_FILE_DEST` is not set, then the datafile location is operating system specific.

See Also: [Chapter 13, "Managing Undo Space"](#)

Specifying the Default Temporary Tablespace Tempfile at Database Creation

The `TEMPFILE` subclause is optional for the `DEFAULT TEMPORARY TABLESPACE` clause and a filename is not required in the file specification. If a filename is not supplied and the `DB_CREATE_FILE_DEST` parameter set, then an Oracle-managed tempfile is created in the `DB_CREATE_FILE_DEST` directory. If `DB_CREATE_FILE_DEST` is not set, then the `CREATE DATABASE` statement fails with a syntax error.

The `DEFAULT TEMPORARY TABLESPACE` clause itself is optional. If it is not specified, then no default temporary tablespace is created.

The default size for an Oracle-managed tempfile is 100 MB and the file is autoextensible with an unlimited maximum size.

CREATE DATABASE Statement Using Oracle-Managed Files: Examples

This section contains examples of the `CREATE DATABASE` statement when using the Oracle Managed Files feature.

CREATE DATABASE: Example 1

This example creates a database with the following Oracle-managed files:

- A **SYSTEM** tablespace datafile in directory `/u01/oradata/sample` that is **100 MB** and autoextensible up to an unlimited size
- Two online log groups with two members of **100 MB** each, one each in `/u02/oradata/sample` and `/u03/oradata/sample`
- If automatic undo management mode is enabled, then an undo tablespace datafile in directory `/u01/oradata/sample2` that is **10 MB** and autoextensible up to an unlimited size. An undo tablespace named `SYS_UNDOTBS` is created.
- If no `CONTROL_FILES` initialization parameter is specified, then two control files, one each in `/u02/oradata/sample` and `/u03/oradata/sample`. The control file in `/u02/oradata/sample` is the primary control file.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'  
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample'  
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample;
```

CREATE DATABASE: Example 2

This example creates a database with the following Oracle-managed files:

- A **100 MB** **SYSTEM** tablespace datafile in directory `/u01/oradata/sample2` that is autoextensible up to an unlimited size.
- Two online redo log files of **100 MB** each in directory `/u01/oradata/sample2`. They are not multiplexed.
- An undo tablespace datafile in directory `/u01/oradata/sample2` that is **10 MB** and autoextensible up to an unlimited size. An undo tablespace named `SYS_UNDOTBS` is created.
- A control file in `/u01/oradata/sample2`

In this example, it is assumed that:

- No `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified in the initialization parameter file.

- No `CONTROL_FILES` initialization parameter was specified in the initialization parameter file.
- Automatic undo management mode is enabled.

The following statements are issued at the SQL prompt:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample2';
SQL> CREATE DATABASE sample2;
```

This database configuration is not recommended for a production database. The example illustrates how a very low-end database or simple test database can easily be created. To better protect this database from failures, at least one more control file should be created and the online redo log should be multiplexed.

CREATE DATABASE: Example 3

In this example, the file size for the Oracle-managed files for the default temporary tablespace and undo tablespace are specified. A database with the following Oracle-managed files is created:

- A 400 MB `SYSTEM` tablespace datafile in directory `/u01/oradata/sample3`. Because `SIZE` is specified, the file is not autoextensible.
- Two online redo log groups with two members of 100 MB each, one each in directories `/u02/oradata/sample3` and `/u03/oradata/sample3`
- For the default temporary tablespace `dflt_ts`, a 10 MB tempfile in directory `/u01/oradata/sample3`. Because `SIZE` is specified, the file is not autoextensible.
- For the undo tablespace `undo_ts`, a 10 MB datafile in directory `/u01/oradata/sample3`. Because `SIZE` is specified, the file is not autoextensible.
- If no `CONTROL_FILES` initialization parameter was specified, then two control files, one each in directories `/u02/oradata/sample3` and `/u03/oradata/sample3`. The control file in `/u02/oradata/sample3` is the primary control file.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample3'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample3'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample3'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE DATABASE sample3 DATAFILE SIZE 400M
2>   DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
3>   UNDO TABLESPACE undo_ts DATAFILE SIZE 10M;
```

Creating Datafiles for Tablespaces

The following statements that can create datafiles are relevant to the discussion in this section:

- CREATE TABLESPACE
- CREATE UNDO TABLESPACE
- ALTER TABLESPACE ... ADD DATAFILE

When creating a tablespace, either a regular tablespace or an undo tablespace, the `DATAFILE` clause is optional. When you include the `DATAFILE` clause the filename is optional. If the `DATAFILE` clause or filename is not provided, then the following rules apply:

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, then an Oracle-managed datafile is created in the location specified by the parameter.
- If the `DB_CREATE_FILE_DEST` initialization parameter is not specified, then the statement creating the datafile fails.

When you add a datafile to a tablespace with the `ALTER TABLESPACE ... ADD DATAFILE` statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph.

By default, an Oracle-managed datafile for a regular tablespace is 100 MB and is autoextensible with an unlimited maximum size. However, if in your `DATAFILE` clause you override these defaults by specifying a `SIZE` value (and no `AUTOEXTEND` clause), then the datafile is *not* autoextensible.

See Also:

- ["Specifying the SYSTEM Tablespace Datafile at Database Creation" on page 3-10](#)
- ["Specifying the Undo Tablespace Datafile at Database Creation" on page 3-11](#)
- [Chapter 11, "Managing Tablespaces"](#)

CREATE TABLESPACE: Examples

The following are some examples of creating tablespaces with Oracle-managed files.

CREATE TABLESPACE: Example 1

The following example sets the default location for datafile creations to `/u01/oradata/sample` and then creates a tablespace `tbs_1` with a datafile in that location. The datafile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample';
SQL> CREATE TABLESPACE tbs_1;
```

CREATE TABLESPACE: Example 2

This example creates a tablespace named `tbs_2` with a datafile in the directory `/u01/oradata/sample2`. The datafile's initial size is 400 MB and it is not autoextensible.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample2'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M AUTOEXTEND OFF;
```

CREATE TABLESPACE: Example 3

This example creates a tablespace named `tbs_3` with an autoextensible datafile in the directory `/u01/oradata/sample3` with a maximum size of 800 MB and an initial size of 100 MB:

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample3'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

CREATE TABLESPACE: Example 4

The following example sets the default location for datafile creations to `/u01/oradata/sample4` and then creates a tablespace named `tbs_4` in that directory with two datafiles. Both datafiles have an initial size of 200 MB, and because a `SIZE` value is specified, they are not autoextensible

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample4';
SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M, SIZE 200M;
```

CREATE UNDO TABLESPACE: Example

The following example creates an undo tablespace named `undotbs_1` with a datafile in the directory `/u01/oradata/sample`. The datafile for the undo tablespace is 100 MB and is autoextensible with an unlimited maximum size.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE UNDO TABLESPACE undotbs_1;
```

ALTER TABLESPACE: Example

This example adds an Oracle-managed autoextensible datafile to the `tbs_1` tablespace. The datafile has an initial size of 100 MB and a maximum size of 800 MB.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
```

The following statement is entered at the SQL prompt:

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

Creating Tempfiles for Temporary Tablespaces

The following statements that create tempfiles are relevant to the discussion in this section:

- `CREATE TEMPORARY TABLESPACE`
- `ALTER TABLESPACE ... ADD TEMPFILE`

When creating a temporary tablespace the `TEMPFILE` clause is optional. If you include the `TEMPFILE` clause, then the filename is optional. If the `TEMPFILE` clause or filename is not provided, then the following rules apply:

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, then an Oracle-managed tempfile is created in the location specified by the parameter.
- If the `DB_CREATE_FILE_DEST` initialization parameter is not specified, then the statement creating the tempfile fails.

When you add a tempfile to a tablespace with the `ALTER TABLESPACE ... ADD TEMPFILE` statement the filename is optional. If the filename is not specified, then the same rules apply as discussed in the previous paragraph.

When overriding the default attributes of an Oracle-managed file, if a `SIZE` value is specified but no `AUTOEXTEND` clause is specified, then the datafile is *not* autoextensible.

See Also: ["Specifying the Default Temporary Tablespace Tempfile at Database Creation"](#) on page 3-11

CREATE TEMPORARY TABLESPACE: Example

The following example sets the default location for datafile creations to `/u01/oradata/sample` and then creates a tablespace named `temptbs_1` with a tempfile in that location. The tempfile is 100 MB and is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

ALTER TABLESPACE ... ADD TEMPFILE: Example

The following example sets the default location for datafile creations to `/u03/oradata/sample` and then adds a tempfile in the default location to a tablespace named `temptbs_1`. The tempfile's initial size is 100 MB. It is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata/sample';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

Creating Control Files

When you issue the `CREATE CONTROLFILE` statement, a control file is created (or reused, if `REUSE` is specified) in the files specified by the `CONTROL_FILES` initialization parameter. If the `CONTROL_FILES` parameter is not set, then the control file is created in the default control file destinations. In order of precedence, the default destination is defined as follows:

- If `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed control file copy is created in each directory specified. The file in the first directory is the primary control file.

- If the `DB_CREATE_FILE_DEST` initialization parameter is specified, and no `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed control file is created in the directory specified.
- If neither `DB_CREATE_ONLINE_LOG_DEST_n` or `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then a control file is created in an operating system specific default location. This control file is not an Oracle-managed file.

If Oracle creates an Oracle-managed control file, and there is a server parameter file, then Oracle creates a `CONTROL_FILES` initialization parameter for the server parameter file. If there is no server parameter file, then you must create a `CONTROL_FILES` initialization parameter manually and include it in the initialization parameter file.

If the datafiles in the database are Oracle-managed files, then the Oracle generated filenames for the files must be supplied in the `DATAFILE` clause of the statement.

If the online redo log files are Oracle-managed files, then the `NORESETLOGS` or `RESETLOGS` keyword determines what can be supplied in the `LOGFILE` clause:

- If the `NORESETLOGS` keyword is used, then the Oracle generated filenames for the Oracle-managed online redo log files must be supplied in the `LOGFILE` clause.
- If the `RESETLOGS` keyword is used, then the online redo log file names can be supplied as with the `CREATE DATABASE` statement. See ["Specifying Online Redo Log Files at Database Creation"](#) on page 3-9.

The sections that follow contain examples of using the `CREATE CONTROLFILE` statement with Oracle-managed files.

See Also: ["Specifying Control Files at Database Creation"](#) on page 3-8

CREATE CONTROLFILE Using NORESETLOGS Keyword: Example

The following `CREATE CONTROLFILE` statement is generated by an `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` statement for a database with Oracle-managed datafiles and online redo log files:

```
CREATE CONTROLFILE
  DATABASE sample
  LOGFILE GROUP 1 ('/u01/oradata/sample/ora_1_o220rtt9.log',
                 '/u02/oradata/sample/ora_1_v2o0b2i3.log') SIZE 100M,
  GROUP 2 ('/u01/oradata/sample/ora_2_p22056iw.log',
```

```

                                '/u02/oradata/sample/ora_2_p02rcyg3.log') SIZE 100M
NORESETLOGS
DATAFILE '/u01/oradata/sample/ora_system_xu34ybm2.dbf' SIZE 100M
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;

```

CREATE CONTROLFILE Using RESETLOGS Keyword: Example

The following is an example of a CREATE CONTROLFILE statement with the RESETLOGS option. Either DB_CREATE_ONLINE_LOG_DEST_1 or DB_CREATE_FILE_DEST must be set.

```

CREATE CONTROLFILE
  DATABASE sample
  RESETLOGS
  DATAFILE '/u01/oradata/sample/ora_system_aawbmz51.dbf' SIZE 100M
  MAXLOGFILES 5
  MAXLOGHISTORY 100
  MAXDATAFILES 10
  MAXINSTANCES 2
  ARCHIVELOG;

```

Later, you must issue the ALTER DATABASE OPEN RESETLOGS statement to re-create the online redo log files. This is discussed in "[Using the ALTER DATABASE OPEN RESETLOGS Statement](#)" on page 3-20. If the previous log files are Oracle-managed files, then they are not deleted.

Creating Online Redo Log Files

Online redo log files are created at database creation time. They can also be created when you issue either of the following statements:

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE OPEN RESETLOGS

Using the ALTER DATABASE ADD LOGFILE Statement

The ALTER DATABASE ADD LOGFILE statement allows you to later add a new group to your current online redo log. The filename in the ADD LOGFILE clause is optional if you are using Oracle-managed files. If a filename is not provided, then a

redo log file is created in the default log file destination. In order of precedence, the default destination is defined as follows:

- If `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed log file member is created in each directory specified in the parameters (up to `MAXLOGMEMBERS` for the database).
- If the `DB_CREATE_FILE_DEST` initialization parameter is specified and no `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed log file member is created in the directory specified in the parameter.

If a filename is not provided and you have not provided one of the initialization parameters required for creating Oracle-managed files, then the statement returns an error.

The default size for an Oracle-managed log file is 100 MB.

You continue to add and drop online redo log file members by specifying complete filenames.

See Also:

- ["Specifying Online Redo Log Files at Database Creation"](#) on page 3-9
- ["Creating Control Files"](#) on page 3-17

Adding New Online Redo Log Files: Example

The following example creates a log file with a member in `/u01/oradata/sample` and another member in `/u02/oradata/sample`. The size of the log file is 100 MB.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata/sample'  
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata/sample'
```

The following statement is issued at the SQL prompt:

```
SQL> ALTER DATABASE ADD LOGFILE;
```

Using the ALTER DATABASE OPEN RESETLOGS Statement

If you previously created a control file specifying `RESETLOGS` and either did not specify filenames or specified non-existent filenames, then Oracle creates online redo log files for you when you issue the `ALTER DATABASE OPEN RESETLOGS`

statement. The rules for determining the directories in which to store redo log files, when none are specified in the control file, are the same as those discussed in ["Specifying Online Redo Log Files at Database Creation"](#) on page 3-9.

Behavior of Oracle-Managed Files

The filenames of Oracle-managed files are accepted in SQL statements wherever a filename is used to identify an existing file. These filenames, like other filenames, are stored in the control file and, if using Recovery Manager (RMAN) for backup and recovery, in the RMAN catalog. They are visible in all of the usual fixed and dynamic performance views that are available for monitoring datafiles and tempfiles (for example, `V$DATAFILE` or `DBA_DATA_FILES`).

The following are some examples of statements using Oracle generated filenames:

```
SQL> ALTER DATABASE RENAME FILE 'ora_tbs01_ziw3bopb.dbf' TO 'tbs0101.dbf';
```

```
SQL> ALTER DATABASE DROP LOGFILE 'ora_1_wo94n2xi.log';
```

```
SQL> ALTER TABLE emp ALLOCATE EXTENT ( DATAFILE 'ora_tbs1_2ixfh90q.dbf' );
```

You can backup and restore Oracle-managed datafiles, tempfiles, and control files as you would corresponding non Oracle-managed files. Using Oracle generated filenames does not impact the use of logical backup files such as export files. This is particularly important for tablespace point-in-time recovery (TSPITR) and transportable tablespace export files.

There are some cases where Oracle-managed files behave differently. These are discussed in the sections that follow.

Dropping Datafiles and Tempfiles

Unlike files that are not Oracle managed, when an Oracle-managed datafile or tempfile is dropped, the filename is removed from the control file and the file is automatically deleted from the file system. The statements that delete Oracle-managed files when they are dropped are:

- `DROP TABLESPACE`
- `ALTER DATABASE TEMPFILE ... DROP`

Dropping Online Redo Log Files

When an Oracle-managed online redo log file is dropped its Oracle-managed files are deleted. You specify the group or members to be dropped. The following statements drop and delete online redo log files:

- ALTER DATABASE DROP LOGFILE
- ALTER DATABASE DROP LOGFILE MEMBER

Renaming Files

The following statements are used to rename files:

- ALTER DATABASE RENAME FILE
- ALTER TABLESPACE ... RENAME DATAFILE

These statements do not actually rename the files on the operating system, but rather, the names in the control file are changed. If the old file is an Oracle-managed file and it exists, then it is deleted. You must specify each filename using the conventions for filenames on your operating system when you issue this statement.

Managing Standby Databases

The datafiles, control files, and online redo log files in a standby database can be Oracle managed. This is independent of whether Oracle-managed files are used on the primary database.

When recovery of a standby database encounters redo for the creation of a datafile, if the datafile is an Oracle-managed file, then the recovery process creates an empty file in the local default file system location. This allows the redo for the new file to be applied immediately without any human intervention.

When recovery of a standby database encounters redo for the deletion of a tablespace, it deletes any Oracle-managed datafiles in the local file system. Note that this is independent of the `INCLUDING DATAFILES` option issued at the primary database.

Scenarios for Using Oracle-Managed Files

This section further demonstrates the use of Oracle-managed files by presenting scenarios of their use.

Scenario 1: Create and Manage a Database with Multiplexed Online Redo Logs

In this scenario, a DBA creates a database where the datafiles and online redo log files are created in separate directories. The online redo log files and control files are multiplexed. The database uses an undo tablespace, and has a default temporary tablespace. The following are tasks involved with creating and maintaining this database.

1. Setting the initialization parameters

The DBA includes three generic file creation defaults in the initialization parameter file before creating the database. Automatic undo management mode is also specified.

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'  
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample'  
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample'  
UNDO_MANAGEMENT = AUTO
```

The `DB_CREATE_FILE_DEST` parameter sets the default file system directory for the datafiles and tempfiles.

The `DB_CREATE_ONLINE_LOG_DEST_1` and `DB_CREATE_ONLINE_LOG_DEST_2` parameters set the default file system directories for online redo log file and control file creation. Each online redo log file and control file is multiplexed across the two directories.

2. Creating a database

Once the initialization parameters are set, the database can be created by using this statement:

```
SQL> CREATE DATABASE sample  
2>   DEFAULT TEMPORARY TABLESPACE df1t_tmp;
```

Because a `DATAFILE` clause is not present and the `DB_CREATE_FILE_DEST` initialization parameter is set, the `SYSTEM` tablespace datafile is created in the default file system (`/u01/oradata/sample` in this scenario). The filename is uniquely generated by Oracle. The file is autoextensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle-managed file.

Because a `LOGFILE` clause is not present, two online redo log groups are created. Each log group has two members, with one member in the `DB_CREATE_ONLINE_LOG_DEST_1` location and the other member in the `DB_CREATE_ONLINE_LOG_DEST_2` location. The filenames are uniquely generated

by Oracle. The log files are created with a size of 100 MB. The log file members are Oracle-managed files.

Similarly, because the `CONTROL_FILES` initialization parameter is not present, and two `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, two control files are created. The control file located in the `DB_CREATE_ONLINE_LOG_DEST_1` location is the primary control file; the control file located in the `DB_CREATE_ONLINE_LOG_DEST_2` location is a multiplexed copy. The filenames are uniquely generated by Oracle. They are Oracle-managed files. Assuming there is a server parameter file, a `CONTROL_FILES` initialization parameter is generated.

Automatic undo management mode is specified, but because an undo tablespace is not specified and the `DB_CREATE_FILE_DEST` initialization parameter is set, a default undo tablespace named `SYS_UNDOTBS` is created in the directory specified by `DB_CREATE_FILE_DEST`. The datafile is a 10 MB datafile that is autoextensible. It is an Oracle-managed file.

Lastly, a default temporary tablespace named `df1t_tmp` is specified. Because `DB_CREATE_FILE_DEST` is included in the parameter file, the tempfile for `df1t_tmp` is created in the directory specified by that parameter. The tempfile is 100 MB and is autoextensible with an unlimited maximum size. It is an Oracle-managed file.

The resultant file tree, with generated filenames, is as follows:

```

/u01
  /oradata
    /sample
      /ora_system_cmr7t30p.dbf
      /ora_sys_undo_2ixfh90q.dbf
      /ora_df1t_tmp_157se6ff.tmp
/u02
  /oradata
    /sample
      /ora_1_0orrm31z.log
      /ora_2_2xyz16am.log
      /ora_cmr7t30p.ctl
/u03
  /oradata
    /sample
      /ora_1_ixfvm8w9.log
      /ora_2_q89tmp28.log
      /ora_xlsr8t36.ctl

```

The internally generated filenames can be seen when selecting from the usual views. For example:

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
-----
/u01/oradata/sample/ora_system_cmr7t30p.dbf
/u01/oradata/sample/ora_sys_undo_2ixfh90q.dbf

2 rows selected
```

The name is also printed to the alert file when the file is created.

3. Managing control files

The control file was created when generating the database, and a `CONTROL_FILES` initialization parameter was added to the parameter file. If needed, then the DBA can re-create the control file or build a new one for the database using the `CREATE CONTROLFILE` statement.

The correct Oracle-managed filenames must be used in the `DATAFILE` and `LOGFILE` clauses. The `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` statement generates a script with the correct filenames. Alternatively, the filenames can be found by selecting from the `V$DATAFILE`, `V$TEMPFILE`, and `V$LOGFILE` views. The following example re-creates the control file for the sample database:

```
SQL> CREATE CONTROLFILE REUSE
2>     DATABASE sample
3>     LOGFILE GROUP 1('/u02/oradata/sample/ora_1_0orm31z.log',
4>                   '/u03/oradata/sample/ora_1_ixfvm8w9.log'),
5>     GROUP 2('/u02/oradata/sample/ora_2_2xyzl6am.log',
6>            '/u03/oradata/sample/ora_2_q89tmp28.log')
7>     NORESETLOGS
8>     DATAFILE '/u01/oradata/sample/ora_system_cmr7t30p.dbf',
9>             '/u01/oradata/sample/ora_sys_undo_2ixfh90q.dbf',
10>            '/u01/oradata/sample/ora_dflt_tmp_157se6ff.tmp'
11>     MAXLOGFILES 5
12>     MAXLOGHISTORY 100
13>     MAXDATAFILES 10
14>     MAXINSTANCES 2
15>     ARCHIVELOG;
```

The control file created by this statement is located as specified by the `CONTROL_FILES` initialization parameter that was generated when the

database was created. The `REUSE` clause causes any existing files to be overwritten.

4. Managing the online redo log

To create a new group of online redo log files, the DBA can use the `ALTER DATABASE ADD LOGFILE` statement. The following statement adds a log file with a member in the `DB_CREATE_ONLINE_LOG_DEST_1` location and a member in the `DB_CREATE_ONLINE_LOG_DEST_2` location. These files are Oracle-managed files.

```
SQL> ALTER DATABASE ADD LOGFILE;
```

Log file members continue to be added and dropped by specifying complete filenames.

The `GROUP` clause can be used to drop a log file. In the following example the operating system file associated with each Oracle-managed log file member is automatically deleted.

```
SQL> ALTER DATABASE DROP LOGFILE GROUP 3;
```

5. Managing tablespaces

The default storage for all datafiles for future tablespace creations in the sample database is the location specified by the `DB_CREATE_FILE_DEST` initialization parameter (`/u01/oradata/sample` in this scenario). Any datafiles for which no filename is specified, are created in the file system specified by the initialization parameter `DB_CREATE_FILE_DEST`. For example:

```
SQL> CREATE TABLESPACE tbs_1;
```

The preceding statement creates a tablespace whose storage is in `/u01/oradata/sample`. A datafile is created with an initial size of 100 MB and it is autoextensible with an unlimited maximum size. The datafile is an Oracle-managed file.

When the tablespace is dropped, the Oracle-managed files for the tablespace are automatically removed. The following statement drops the tablespace and all the Oracle-managed files used for its storage:

```
SQL> DROP TABLESPACE tbs_1;
```

Once the first datafile is full, Oracle does not automatically create a new datafile. More space can be added to the tablespace by adding another

Oracle-managed datafile. The following statement adds another datafile in the location specified by `DB_CREATE_FILE_DEST`:

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;
```

The default file system can be changed by changing the initialization parameter. This does not change any existing datafiles. It only affects future creations. This can be done dynamically using the following statement:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata/sample';
```

6. Archiving redo information

Archiving of online redo log files is no different for Oracle-managed files, than it is for unmanaged files. A file system location for the archived log files can be specified using the `LOG_ARCHIVE_DEST_n` initialization parameters. The filenames are formed based on the `LOG_ARCHIVE_FORMAT` parameter or its default. The archived logs are not Oracle-managed files

7. Backup, restore, and recover

Since an Oracle-managed file is compatible with standard operating system files, you can use operating system utilities to backup or restore Oracle-managed files. All existing methods for backing up, restoring, and recovering the database work for Oracle-managed files.

Scenario 2: Add Oracle-Managed Files to an Existing Database

Assume in this case that an existing database does not have any Oracle-managed files, but the DBA would like to create new tablespaces with Oracle-managed files and locate them in directory `/u03/oradata/sample2`.

1. Setting the initialization parameters

To allow automatic datafile creation, set the `DB_CREATE_FILE_DEST` initialization parameter to the file system directory in which to create the datafiles. This can be done dynamically as follows:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata/sample2';
```

2. Creating tablespaces

Once `DB_CREATE_FILE_DEST` is set, the `DATAFILE` clause can be omitted from a `CREATE TABLESPACE` statement. The datafile is created in the location specified by `DB_CREATE_FILE_DEST` by default. For example:

```
SQL> CREATE TABLESPACE tbs_2;
```

When the `tbls_2` tablespace is dropped, its datafiles are automatically deleted.

Starting Up and Shutting Down

This chapter describes the procedures for starting up and shutting down an Oracle database, and contains the following topics:

- [Starting Up a Database](#)
- [Altering Database Availability](#)
- [Shutting Down a Database](#)
- [Quiescing a Database](#)
- [Suspending and Resuming a Database](#)

See Also: For additional information specific to an Oracle Real Application Clusters environment:

- *Oracle9i Real Application Clusters Administration*
- *Oracle9i Real Application Clusters Setup and Configuration*

Starting Up a Database

When you start up a database, you create an instance of that database, and you choose the state in which the database starts. Normally, you would start up an instance by mounting and opening the database, thus making it available for any valid user to connect to and perform typical data access operations. However, there are other options and these are also discussed in this section.

This section contains the following topics relating to starting up an instance of a database:

- [Options for Starting Up a Database](#)
- [Preparing to Start an Instance](#)
- [Using SQL*Plus to Start Up a Database](#)
- [Starting an Instance: Scenarios](#)

Options for Starting Up a Database

There are options as to the method you use for starting up (and administering) an instance of your database.

Using SQL*Plus

To start up a database use SQL*Plus to connect to Oracle with administrator privileges and then issue the `STARTUP` command. While three methods are presented, using SQL*Plus is the only method that is within the scope of this book.

Using Recovery Manager

You can also use Recovery Manager (RMAN) to execute `STARTUP` (and `SHUTDOWN`) commands. You may prefer to do this if you are within the RMAN environment and do not want to invoke SQL*Plus.

See Also: *Oracle9i Recovery Manager User's Guide*

Using Oracle Enterprise Manager

You can choose to use the Oracle Enterprise Manager for administering your database, including starting it up and shutting it down. The Oracle Enterprise Manager is a separate Oracle product, that combines a graphical console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products. It enables you to perform the functions discussed in this book using a GUI interface, rather than command lines.

See Also:

- *Oracle Enterprise Manager Concepts Guide*
- *Oracle Enterprise Manager Administrator's Guide*

Preparing to Start an Instance

You must perform some preliminary steps before attempting to start an instance of your database using SQL*Plus.

1. Start SQL*Plus without connecting to the database:

```
SQLPLUS /NOLOG
```

2. Connect to Oracle as SYSDBA:

```
CONNECT username/password AS SYSDBA
```

Now you are connected to Oracle and ready to start up an instance of your database.

See Also: *SQL*Plus User's Guide and Reference* for descriptions and syntax for the `CONNECT`, `STARTUP`, and `SHUTDOWN` commands. These commands are SQL*Plus commands.

Using SQL*Plus to Start Up a Database

You use the `STARTUP` command to start up a database instance. To start an instance, Oracle must read instance configuration parameters (the initialization parameters) from either a server parameter file or a traditional text initialization parameter file.

When you issue the `STARTUP` command with no `PFILE` clause, Oracle reads the initialization parameters from a server parameter file (`SPFILE`) in a platform-specific default location.

Note: For UNIX, the platform-specific default location (directory) for the server parameter file (or text initialization parameter file) is:

```
$ORACLE_HOME/dbs
```

For Windows NT and Windows 2000 the location is:

```
$ORACLE_HOME\database
```

In the platform-specific default location, Oracle locates your initialization parameter file by examining filenames in the following order:

1. `spfile$ORACLE_SID.ora`
2. `spfile.ora`
3. `init$ORACLE_SID.ora`

Note: The `spfile.ora` file is included in this search path because in a Real Application Clusters environment one server parameter file is used to store the initialization parameter settings for all instances. There is no instance specific location for storing a server parameter file.

For more information about the server parameter file for a Real Application Clusters environment, see *Oracle9i Real Application Clusters Administration*.

You can direct Oracle to read initialization parameters from a traditional text initialization parameter file, by using the `PFILE` clause of the `STARTUP` command. For example:

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

Further, you can use this `PFILE` clause to start an instance with a nondefault server parameter file as follows:

1. Create a one line text initialization parameter file that contains only the `SPFILE` parameter. The value of the parameter is the nondefault server parameter file location.

For example, create a text initialization parameter file `/u01/oracle/dbs/spf_init.ora` that contains only the following parameter:

```
SPFILE = /u01/oracle/dbs/test_spfile.ora
```

Note: You cannot use the `IFILE` initialization parameter within a text initialization parameter file to point to a server parameter file. In this context, you must use the `SPFILE` initialization parameter.

2. Start up the instance pointing to this initialization parameter file.

```
STARTUP PFILE = /u01/oracle/dbs/spf_init.ora
```

Since the server parameter file must reside on the machine running the database server, the above method also provides a means for a client machine to start a database that uses a server parameter file. It also eliminates the need for a client machine to maintain a client-side initialization parameter file. When the client machine reads the initialization parameter file containing the `SPFILE` parameter, it passes the value to the server where the specified server parameter file is read.

You can start an instance in various modes:

- Start the instance without mounting a database. This does not allow access to the database and usually would be done only for database creation or the re-creation of control files.
- Start the instance and mount the database, but leave it closed. This state allows for certain DBA activities, but does not allow general access to the database.
- Start the instance, and mount and open the database. This can be done in unrestricted mode, allowing access to all users, or in restricted mode, allowing access for database administrators only.

Note: You cannot start a database instance if you are connected to the database through a shared server process.

In addition, you can force the instance to start, or start the instance and have complete media recovery begin immediately. The `STARTUP` command options that you specify to achieve these states are illustrated in the following section.

See Also: [Chapter 2, "Creating an Oracle Database"](#) for more information about initialization parameters, initialization parameter files, and server parameter files

Starting an Instance: Scenarios

The following scenarios describe and illustrate the various states in which you can start up an instance. Some restrictions apply when combining options of the `STARTUP` command.

Note: It is possible to encounter problems starting up an instance if control files, database files, or redo log files are not available. If one or more of the files specified by the `CONTROL_FILES` initialization parameter does not exist or cannot be opened when you attempt to mount a database, Oracle returns a warning message and does not mount the database. If one or more of the datafiles or redo log files is not available or cannot be opened when attempting to open a database, Oracle returns a warning message and does not open the database.

See Also: *SQL*Plus User's Guide and Reference* for information about the restrictions that apply when combining options of the `STARTUP` command

Starting an Instance, and Mounting and Opening a Database

Normal database operation means that an instance is started and the database is mounted and open. This mode allows any valid user to connect to the database and perform typical data access operations.

Start an instance, read the initialization parameters from the default server parameter file location, and then mount and open the database by using the `STARTUP` command by itself (you can, of course, optionally specify a `PFILE` or `SPFILE` clause):

```
STARTUP
```

Starting an Instance Without Mounting a Database

You can start an instance without mounting a database. Typically, you do so only during database creation. Use the `STARTUP` command with the `NOMOUNT` option:

```
STARTUP NOMOUNT
```

Starting an Instance and Mounting a Database

You can start an instance and mount a database without opening it, allowing you to perform specific maintenance operations. For example, the database must be mounted but not open during the following tasks:

Task	For more information...
Renaming datafiles	Chapter 12, "Managing Datafiles"
Adding, dropping, or renaming redo log files	Chapter 7, "Managing the Online Redo Log"
Enabling and disabling redo log archiving options	Chapter 8, "Managing Archived Redo Logs"
Performing full database recovery	<i>Oracle9i User-Managed Backup and Recovery Guide</i> <i>Oracle9i Recovery Manager User's Guide</i>

Start an instance and mount the database, but leave it closed by using the `STARTUP` command with the `MOUNT` option:

```
STARTUP MOUNT
```

Restricting Access to a Database at Startup

You can start an instance and mount and open a database in restricted mode so that the database is available only to administrative personnel (not general database users). Use this mode of database startup when you need to accomplish one of the following tasks:

- Perform an export or import of database data
- Perform a data load (with SQL*Loader)
- Temporarily prevent typical users from using data
- During certain migration and upgrade operations

Typically, all users with the `CREATE SESSION` system privilege can connect to an open database. Opening a database in restricted mode allows database access only to users with both the `CREATE SESSION` and `RESTRICTED SESSION` system privilege. Only database administrators should have the `RESTRICTED SESSION` system privilege.

Start an instance (and, optionally, mount and open the database) in restricted mode by using the `STARTUP` command with the `RESTRICT` option:

```
STARTUP RESTRICT
```

Later, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION` feature:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

If you open the database in nonrestricted mode and later find you need to restrict access, you can use the `ALTER SYSTEM` statement to do so, as described in ["Restricting Access to an Open Database"](#) on page 4-11.

See Also: *Oracle9i SQL Reference* for more information on the `ALTER SYSTEM` statement

Forcing an Instance to Start

In unusual circumstances, you might experience problems when attempting to start a database instance. You should not force a database to start unless you are faced with the following:

- You cannot shut down the current instance with the `SHUTDOWN NORMAL`, `SHUTDOWN IMMEDIATE`, or `SHUTDOWN TRANSACTIONAL` commands.
- You experience problems when starting an instance.

If one of these situations arises, you can usually solve the problem by starting a new instance (and optionally mounting and opening the database) using the `STARTUP` command with the `FORCE` option:

```
STARTUP FORCE
```

If an instance is running, `STARTUP FORCE` shuts it down with mode `ABORT` before restarting it.

See Also: ["Shutting Down with the ABORT Option"](#) on page 4-13 to understand the side effects of aborting the current instance

Starting an Instance, Mounting a Database, and Starting Complete Media Recovery

If you know that media recovery is required, you can start an instance, mount a database to the instance, and have the recovery process automatically start by using the `STARTUP` command with the `RECOVER` option:

```
STARTUP OPEN RECOVER
```

If you attempt to perform recovery when no recovery is required, Oracle issues an error message.

Automatic Database Startup at Operating System Start

Many sites use procedures to enable automatic startup of one or more Oracle instances and databases immediately following a system start. The procedures for performing this task are specific to each operating system. For information about automatic startup, see your operating system specific Oracle documentation.

Starting Remote Instances

If your local Oracle server is part of a distributed database, you might want to start a remote instance and database. Procedures for starting and stopping remote instances vary widely depending on communication protocol and operating system.

Altering Database Availability

You can alter the availability of a database. You may want to do this in order to restrict access for maintenance reasons or to make the database read only. The following sections explain how to alter a database's availability:

- [Mounting a Database to an Instance](#)
- [Opening a Closed Database](#)
- [Opening a Database in Read-Only Mode](#)
- [Restricting Access to an Open Database](#)

Mounting a Database to an Instance

When you need to perform specific administrative operations, the database must be started and mounted to an instance, but closed. You can achieve this scenario by starting the instance and mounting the database.

To mount a database to a previously started, but not opened instance, use the SQL statement `ALTER DATABASE` with the `MOUNT` option as follows:

```
ALTER DATABASE MOUNT
```

See Also: ["Starting an Instance and Mounting a Database"](#) on page 4-6 for a list of operations that require the database to be mounted and closed (and procedures to start an instance and mount a database in one step)

Opening a Closed Database

You can make a mounted but closed database available for general use by opening the database. To open a mounted database, use the `ALTER DATABASE` statement with the `OPEN` option:

```
ALTER DATABASE OPEN
```

After executing this statement, any valid Oracle user with the `CREATE SESSION` system privilege can connect to the database.

Opening a Database in Read-Only Mode

Opening a database in read-only mode enables you to query an open database while eliminating any potential for online data content changes. While opening a database in read-only mode guarantees that datafile and redo log files are not written to, it does not restrict database recovery or operations that change the state of the database without generating redo. For example, you can take datafiles offline or bring them online since these operations do not effect data content.

If a query against a database in read-only mode uses temporary tablespace, for example to do disk sorts, then the issuer of the query must have a locally managed tablespace assigned as the default temporary tablespace. Otherwise, the query will fail. This is explained in "[Creating a Locally Managed Temporary Tablespace](#)" on page 11-13.

Ideally, you open a database in read-only mode when you alternate a standby database between read-only and recovery mode. Be aware that these are mutually exclusive modes.

The following statement opens a database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

You can also open a database in read-write mode as follows:

```
ALTER DATABASE OPEN READ WRITE;
```

However, read-write is the default mode.

Note: You cannot use the `RESETLOGS` clause with a `READ ONLY` clause.

See Also: *Oracle9i SQL Reference* for more information about the `ALTER DATABASE` statement

Restricting Access to an Open Database

To place an instance in restricted mode, use the SQL statement `ALTER SYSTEM` with the `ENABLE RESTRICTED SESSION` clause. After placing an instance in restricted mode, you should consider killing all current user sessions before performing any administrative tasks. To lift an instance from restricted mode, use `ALTER SYSTEM` with the `DISABLE RESTRICTED SESSION` option.

See Also: ["Restricting Access to a Database at Startup"](#) on page 4-7 to learn some reasons for placing an instance in restricted mode

Shutting Down a Database

To initiate database shutdown, use the SQL*Plus `SHUTDOWN` command. Control is not returned to the session that initiates a database shutdown until shutdown is complete. Users who attempt connections while a shutdown is in progress receive a message like the following:

```
ORA-01090: shutdown in progress - connection is not permitted
```

Note: You cannot shut down a database if you are connected to the database through a shared server process.

To shut down a database and instance, you must first connect as `SYSOPER` or `SYSDBA`. There are several modes for shutting down a database. These are discussed in the following sections:

- [Shutting Down with the NORMAL Option](#)
- [Shutting Down with the IMMEDIATE Option](#)
- [Shutting Down with the TRANSACTIONAL Option](#)
- [Shutting Down with the ABORT Option](#)

Shutting Down with the NORMAL Option

To shut down a database in normal situations, use the `SHUTDOWN` command with the `NORMAL` option:

```
SHUTDOWN NORMAL
```

Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, Oracle waits for all currently connected users to disconnect from the database.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the IMMEDIATE Option

Use immediate database shutdown only in the following situations:

- To initiate an automated and unattended backup
- When a power shutdown is going to occur soon
- When the database or one of its applications is functioning irregularly and you cannot contact users to ask them to log off or they are unable to log off

To shut down a database immediately, use the `SHUTDOWN` command with the `IMMEDIATE` option:

```
SHUTDOWN IMMEDIATE
```

Immediate database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Any uncommitted transactions are rolled back. (If long uncommitted transactions exist, this method of shutdown might not complete quickly, despite its name.)
- Oracle does not wait for users currently connected to the database to disconnect. Oracle implicitly rolls back active transactions and disconnects all connected users.

The next startup of the database will not require any instance recovery procedures.

Shutting Down with the TRANSACTIONAL Option

When you want to perform a planned shutdown of an instance while allowing active transactions to complete first, use the `SHUTDOWN` command with the `TRANSACTIONAL` option:

```
SHUTDOWN TRANSACTIONAL
```

Transactional database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- After all transactions have completed, any client still connected to the instance is disconnected.
- At this point, the instance shuts down just as it would when a `SHUTDOWN IMMEDIATE` statement is submitted.

The next startup of the database will not require any instance recovery procedures.

A transactional shutdown prevents clients from losing work, and at the same time, does not require all users to log off.

Shutting Down with the ABORT Option

You can shut down a database instantaneously by aborting the database's instance. If possible, perform this type of shutdown *only* in the following situations:

- The database or one of its applications is functioning irregularly *and* none of the other types of shutdown works.
- You need to shut down the database instantaneously (for example, if you know a power shutdown is going to occur in one minute).
- You experience problems when starting a database instance.

When you must do a database shutdown by aborting transactions and user connections, issue the `SHUTDOWN` command with the `ABORT` option:

```
SHUTDOWN ABORT
```

An aborted database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.

- Current client SQL statements being processed by Oracle are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle does not wait for users currently connected to the database to disconnect. Oracle implicitly disconnects all connected users.

The next startup of the database *will* require instance recovery procedures.

Quiescing a Database

There are times when there is a need to put a database into a state where only DBA transactions, queries, fetches, or PL/SQL statements are allowed. This is called a quiesced state, in the sense that there are no ongoing non-DBA transactions, queries, fetches, or PL/SQL statements in the system. This quiesced state allows you or other administrators to perform actions that cannot safely be done otherwise. These actions are categorized as follows:

- Actions that can fail if concurrent user transactions access the same object. For example, changing the schema of a database table or adding a column to an existing table where a no-wait lock is required.
- Actions whose undesirable intermediate effect can be seen by concurrent user transactions. For example, a multistep procedure for reorganizing a table where the table is first exported, then dropped, and finally imported. A concurrent user who attempted to access the table after it was dropped, but before import, would see disturbing results.

Without the ability to quiesce the database, you would be required to shut down the database and reopen it in restricted mode. This is a serious restriction, especially for systems requiring 24 x 7 availability. Quiescing a database is much less of a restriction because it eliminates the disruption to users and downtime associated with shutting down and restarting the database.

Note: For this release of Oracle9i, in the quiesce database context a DBA is defined as user `SYS` or `SYSTEM`. Other users, including those with the DBA role are not allowed to issue the `ALTER SYSTEM QUIESCE DATABASE` statement or proceed after the database is quiesced.

Placing a Database into a Quiesced State

To place a database into a quiesced state, issue the following statement:

```
ALTER SYSTEM QUIESCE RESTRICTED;
```

Any non-DBA active sessions will proceed until they become inactive. An active session is defined as a session that is currently inside of a transaction, a query, a fetch, or a PL/SQL statement; or a session that is currently holding any shared resources (for example, enqueues). No inactive sessions are allowed to become active. If a user, for example, issues a SQL query in an attempt to force an inactive session to become active, the query will appear to be hung. When the database is later unquiesced, the session is resumed, and the blocked action (for example, the previously mentioned SQL query) will be processed.

Once all non-DBA sessions become inactive, the `ALTER SYSTEM QUIESCE RESTRICTED` statement finishes, and the database is considered as in a quiesced state. In an Oracle Real Application Clusters environment, this statement affects all instances, not just the one that issues the statement.

Note: You must have the Database Resource Manager feature activated, and it must have been activated since instance startup (all instances in an Oracle Real Application Clusters environment) to successfully issue the `ALTER SYSTEM QUIESCE RESTRICTED` statement. It is through the facilities of the Database Resource Manager that non-DBA sessions are prevented from becoming active. Also, while this statement is in effect, any attempt to change the current resource plan will be queued until after the system is unquiesced.

For information about the Database Resource Manager, see [Chapter 27, "Using the Database Resource Manager"](#).

The `ALTER SYSTEM QUIESCE RESTRICTED` statement may wait a long time for active sessions to become inactive. If you interrupt the request, or if your session abnormally terminates for some reason before all active sessions are quiesced, Oracle will automatically undo any partial effects of the statement.

If a query is carried out by successive multiple Oracle Call Interface (OCI) fetches, the `ALTER SYSTEM QUIESCE RESTRICTED` statement does not wait for all fetches to finish; it only waits for the current fetch to finish.

For both dedicated and shared server connections, all non-DBA logins after this statement is issued are queued by the Database Resource Manager, and are not allowed to proceed. To the user, it appears as if the login is hung. The login will resume when the database is unquiesced.

The database remains in the quiesced state even if the session that issued the statement exits. A DBA must log in to the database to issue the statement that specifically unquiesces the database.

While in the quiesced state, you cannot use file system copy to backup the database's datafiles as cold backups, even if you do a checkpoint on every instance. The reason for this is that in the quiesced state the file headers of online datafiles continue to look like they are being accessed. They do not look the same as if a clean shutdown were done. Similarly, to perform a hot backup of the datafiles of any online tablespace while the database is in a quiesced state, you are still required to first place the tablespace into backup mode using the `ALTER TABLESPACE . . . BEGIN BACKUP` statement.

Restoring the System to Normal Operation

The following statement restores the database to normal operation:

```
ALTER SYSTEM UNQUIESCE;
```

All non-DBA activity is allowed to proceed. In an Oracle Real Application Clusters environment, this statement is not required to be issued from the same session, or even the same instance, as that which imposed the quiesce state. If the session issuing the `ALTER SYSTEM UNQUIESCE` statement should terminate abnormally, the Oracle database server ensures that the unquiesce operation finishes.

Viewing the Quiesce State of an Instance

The `V$INSTANCE` view can be queried to see the current state of an instance. It contains a column named `ACTIVE_STATE`, whose values are shown in the following table:

<code>ACTIVE_STATE</code>	Description
<code>NORMAL</code>	Normal unquiesced state
<code>QUIESCING</code>	Being quiesced, but there are still active non-DBA sessions running
<code>QUIESCED</code>	Quiesced, no active non-DBA sessions are active or allowed

Suspending and Resuming a Database

The `ALTER SYSTEM SUSPEND` statement suspends a database by halting all input and output (I/O) to datafiles (file header and file data) and control files, thus allowing a database to be backed up without I/O interference. When the database is suspended all preexisting I/O operations are allowed to complete and any new database accesses are placed in a queued state.

The suspend command suspends the database, and is not specific to an instance. Therefore, in an Oracle Real Application Clusters environment, if the suspend command is entered on one system, then internal locking mechanisms will propagate the halt request across instances, thereby quiescing all active instances in a given cluster. However, do not start a new instance while you suspend another instance, since the new instance will not be suspended.

Use the `ALTER SYSTEM RESUME` statement to resume normal database operations. You can specify the `SUSPEND` and `RESUME` from different instances. For example, if instances 1, 2, and 3 are running, and you issue an `ALTER SYSTEM SUSPEND` statement from instance 1, then you can issue a `RESUME` from instance 1, 2, or 3 with the same effect.

The suspend/resume feature is useful in systems that allow you to mirror a disk or file and then split the mirror, providing an alternative backup and restore solution. If you use a system that is unable to split a mirrored disk from an existing database while writes are occurring, then you can use the suspend/resume feature to facilitate the split.

The suspend/resume feature is not a suitable substitute for normal shutdown operations, however, since copies of a suspended database can contain uncommitted updates.

Caution: Do not use the `ALTER SYSTEM SUSPEND` statement as a substitute for placing a tablespace in hot backup mode. Precede any database suspend operation by an `ALTER TABLESPACE BEGIN BACKUP` statement.

The following statements illustrate `ALTER SYSTEM SUSPEND/RESUME` usage. The `V$INSTANCE` view is queried to confirm database status.

```
SQL> ALTER SYSTEM SUSPEND;
System altered
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;
DATABASE_STATUS
```

```
-----  
SUSPENDED  
  
SQL> ALTER SYSTEM RESUME;  
System altered  
SQL> SELECT DATABASE_STATUS FROM V$INSTANCE;  
DATABASE_STATUS  
-----  
ACTIVE
```

See Also: *Oracle9i User-Managed Backup and Recovery Guide* for details about backing up a database using the database suspend/resume feature

Part II

Oracle Server Processes and Storage Structure

Part II presents the Oracle database server processes and underlying database storage structures that support its operation. It contains the following chapters:

- [Chapter 5, "Managing Oracle Processes"](#)
- [Chapter 6, "Managing Control Files"](#)
- [Chapter 7, "Managing the Online Redo Log"](#)
- [Chapter 8, "Managing Archived Redo Logs"](#)
- [Chapter 9, "Using LogMiner to Analyze Redo Logs"](#)
- [Chapter 10, "Managing Job Queues"](#)
- [Chapter 11, "Managing Tablespaces"](#)
- [Chapter 12, "Managing Datafiles"](#)
- [Chapter 13, "Managing Undo Space"](#)

Managing Oracle Processes

This chapter describes how to manage the processes of an Oracle instance, and contains the following topics:

- [Server Processes](#)
- [Configuring Oracle for the Shared Server](#)
- [About Oracle Background Processes](#)
- [Monitoring the Processes of an Oracle Instance](#)
- [Managing Processes for Parallel Execution](#)
- [Managing Processes for External Procedures](#)
- [Terminating Sessions](#)

Server Processes

Oracle creates server processes to handle the requests of user processes connected to an instance. A server process can be either a **dedicated server process**, where one server process services only one user process, or if your database server is configured for **shared server**, it can be a **shared server process**, where a server process can service multiple user processes.

See Also: *Oracle9i Database Concepts*

Dedicated Server Processes

Figure 5–1, "Oracle Dedicated Server Processes" illustrates how dedicated server processes work. In this diagram two user processes are connected to Oracle through dedicated server processes.

In general, it is better to be connected through a dispatcher and use a shared server process. This is illustrated in Figure 5–2, "Oracle Shared Server Processes". A shared server process can be more efficient because it keeps the number of processes required for the running instance low.

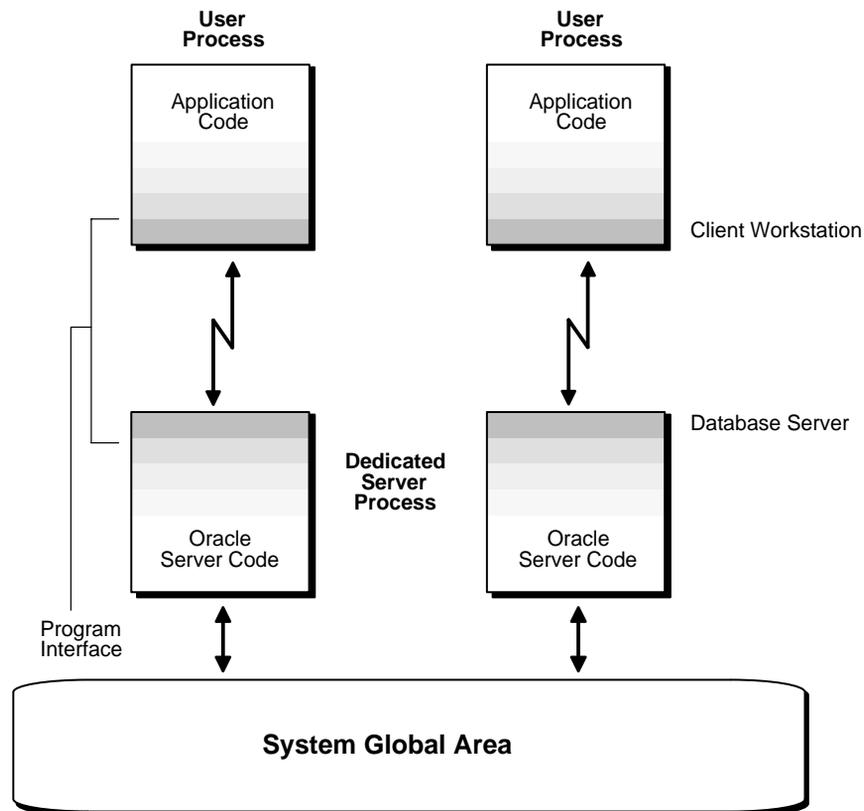
In the following situations, however, users and administrators should explicitly connect to an instance using a dedicated server process:

- To submit a batch job (for example, when a job can allow little or no idle time for the server process)
- To use Recovery Manager to back up, restore, or recover a database

To request a dedicated server connection when Oracle is configured for shared server, users must connect using a net service name that is configured to use a dedicated server. Specifically, the net service name value should include the `SERVER=DEDICATED` clause in the connect descriptor.

See Also: *Oracle9i Net Services Administrator's Guide* for more information about requesting a dedicated server connection

Figure 5–1 Oracle Dedicated Server Processes

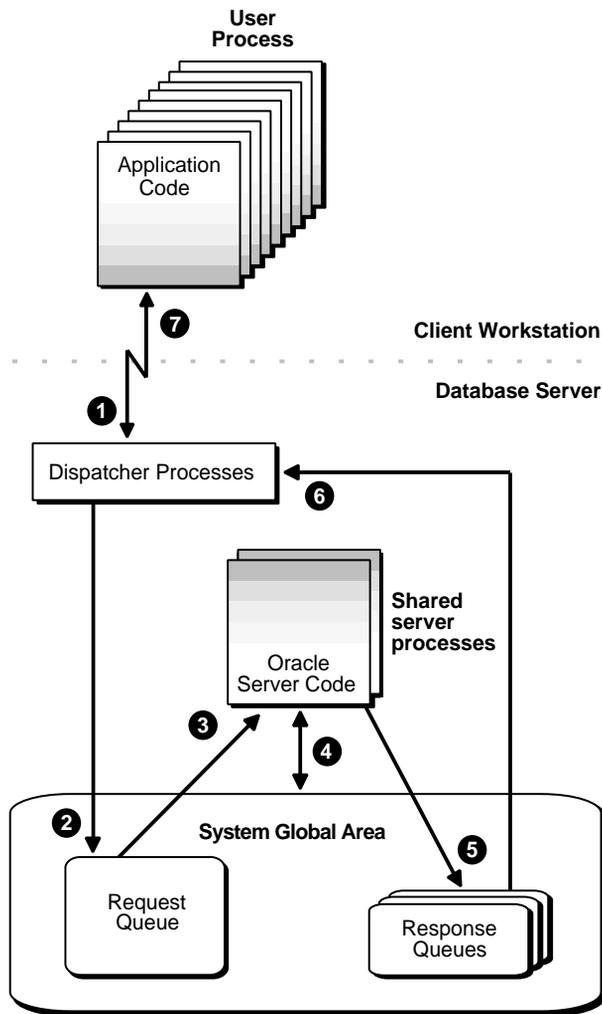


Shared Server Processes

Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer and the server process dedicated to the clerk's user process remains idle. The server process is not needed during most of the transaction, and the system is slower for other clerks entering orders because the idle server process is holding system resources.

The shared server architecture eliminates the need for a dedicated server process for each connection (see [Figure 5–2](#)).

Figure 5-2 Oracle Shared Server Processes



In a shared server configuration, client user processes connect to a dispatcher. A dispatcher can support multiple client connections concurrently. Each client connection is bound to a virtual circuit. A virtual circuit is a piece of shared memory used by the dispatcher for client database connection requests and replies. The dispatcher places a virtual circuit on a common queue when a request arrives.

An idle shared server picks up the virtual circuit from the common queue, services the request, and relinquishes the virtual circuit before attempting to retrieve another virtual circuit from the common queue. This approach enables a small pool of server processes to serve a large number of clients. A significant advantage of shared server architecture over the dedicated server model is the reduction of system resources, enabling the support of an increased number of users.

The shared server architecture requires Oracle Net Services. User processes targeting the shared server must connect through Oracle Net Services, even if they are on the same machine as the Oracle instance.

There are several things that must be done to configure your system for shared server. These are discussed in the next section.

See Also: *Oracle9i Net Services Administrator's Guide* for more information about shared server, including additional features such as connection pooling

Configuring Oracle for the Shared Server

You activate shared server by setting database initialization parameters. Shared server requires that an Oracle Net Services listener process be active. This section discusses setting shared server initialization parameters and how to alter them.

This section contains the following topics:

- [Initialization Parameters for Shared Server](#)
- [Setting the Initial Number of Dispatchers \(DISPATCHERS\)](#)
- [Setting the Initial Number of Shared Servers \(SHARED_SERVERS\)](#)
- [Modifying Dispatcher and Server Processes](#)
- [Monitoring Shared Server](#)

Initialization Parameters for Shared Server

The initialization parameters controlling shared server are:

Parameter	Description
The following parameter is required by shared server:	
DISPATCHERS	Configures dispatcher processes in the shared server architecture.

Parameter	Description
The following parameters are optional (if not specified, Oracle selects defaults):	
MAX_DISPATCHERS	Specifies the maximum number of dispatcher processes that can run simultaneously.
SHARED_SERVERS	Specifies the number of shared server processes created when an instance is started up.
MAX_SHARED_SERVERS	Specifies the maximum number of shared server processes that can run simultaneously.
CIRCUITS	Specifies the total number of virtual circuits that are available for inbound and outbound network sessions.
SHARED_SERVER_SESSIONS	Specifies the total number of shared server user sessions to allow. Setting this parameter enables you to reserve user sessions for dedicated servers.
Other parameters affected by shared server that may require adjustment:	
LARGE_POOL_SIZE	Specifies the size in bytes of the large pool allocation heap. Shared server may force the default value to be set too high, causing performance problems or problems starting the database.
SESSIONS	Specifies the maximum number of sessions that can be created in the system. May need to be adjusted for shared server.

See Also:

- *Oracle9i Net Services Reference Guide*
- *Oracle9i Database Reference*

Setting the Initial Number of Dispatchers (DISPATCHERS)

The number of dispatcher processes started at instance startup is controlled by the DISPATCHERS initialization parameter. You can specify multiple DISPATCHERS parameters in the initialization file, but they must be adjacent to each other. Internally, Oracle will assign an INDEX value to each DISPATCHERS parameter, so that you can later specifically refer to that DISPATCHERS parameter in an ALTER SYSTEM statement.

The appropriate number of dispatcher processes for each instance depends upon the performance you want from your database, the host operating system's limit on the number of connections for each process (which is operating system dependent),

and the number of connections required for each network protocol. The instance must be able to provide as many connections as there are concurrent users on the database system. After instance startup, you can start more dispatcher processes if needed. This is discussed in ["Adding and Removing Dispatcher Processes"](#) on page 5-8.

A ratio of 1 dispatcher for every 1000 connections works well for typical systems, but round up to the next integer. For example, if you anticipate 1500 connections at peak time, then you may want to configure 2 dispatchers. Being too aggressive in your estimates is not beneficial, because configuring too many dispatchers can degrade performance. Use this ratio as your guide, but tune according to your particular circumstances.

The following are some examples of setting the `DISPATCHERS` initialization parameter.

Example: Typical

This is a typical example of setting the `DISPATCHERS` initialization parameter.

```
DISPATCHERS=" (PROTOCOL=TCP) "
```

Example: Forcing the IP Address Used for Dispatchers

To force the IP address used for the dispatchers, enter the following:

```
DISPATCHERS=" (ADDRESS=(PROTOCOL=TCP)\
                (HOST=144.25.16.201)) (DISPATCHERS=2) "
```

This will start two dispatchers that will listen in on the IP address, which must be a valid IP address for the host that the instance is on.

Example: Forcing the Port Used by Dispatchers

To force the exact location of dispatchers, add the `PORT` as follows:

```
DISPATCHERS=" (ADDRESS=(PROTOCOL=TCP) (PORT=5000)) "
DISPATCHERS=" (ADDRESS=(PROTOCOL=TCP) (PORT=5001)) "
```

Setting the Initial Number of Shared Servers (`SHARED_SERVERS`)

The `SHARED_SERVERS` initialization parameter specifies the number of shared server processes that you want to create when an instance is started up. Oracle dynamically adjusts the number of shared server processes based on the length of the request queue. The number of shared server processes that can be created

ranges between the values of the initialization parameters `SHARED_SERVERS` and `MAX_SHARED_SERVERS`. Typical systems seem to stabilize at a ratio of one shared server for every ten connections.

For OLTP applications, the connections-to-servers ratio could be higher. This could happen when the rate of requests is low, or when the ratio of server usage to request is low. On the other hand, in applications where the rate of requests is high, or the server usage-to-request ratio is high, the connections-to-server ratio could be lower.

Set `MAX_SHARED_SERVERS` to a reasonable value based on your application. Oracle provides good defaults for `SHARED_SERVERS` and `MAX_SHARED_SERVERS` for a typical configuration, but the optimal values for these settings can be different depending upon your application.

Note: On Windows NT, take care when setting `MAX_SHARED_SERVERS` to a high value because each server is a thread in a common process.

`MAX_SHARED_SERVERS` is a static initialization parameter, so you cannot change it without shutting down your database. However, `SHARED_SERVERS` is a dynamic initialization parameter and can be changed using an `ALTER SYSTEM` statement.

Modifying Dispatcher and Server Processes

You can modify the settings for `DISPATCHERS` and `SHARED_SERVERS` dynamically when an instance is running. If you have the `ALTER SYSTEM` privilege, you can use the `ALTER SYSTEM` statement to make such changes.

See Also: *Oracle9i SQL Reference* for information about the `ALTER SYSTEM` statement

Adding and Removing Dispatcher Processes

You can control the number of dispatcher processes in the instance. If monitoring the `V$QUEUE`, `V$DISPATCHER` and `V$DISPATCHER_RATE` views indicates that the load on the dispatcher processes is consistently high, starting additional dispatcher processes to route user requests may improve performance. In contrast, if the load on dispatchers is consistently low, reducing the number of dispatchers may improve performance.

To change the number of dispatcher processes, use the SQL statement `ALTER SYSTEM`. You can start new dispatcher processes for an existing `DISPATCHERS`

value, or you can add new `DISPATCHERS` values. Dispatchers can be added up to the limit specified by `MAX_DISPATCHERS`.

If you reduce the number of dispatchers for a particular shared server dispatcher value, the dispatchers are not immediately removed. Rather, as users disconnect, Oracle is eventually able to terminate dispatchers down to the limit you specify in `DISPATCHERS`.

The following statement dynamically changes the number of dispatcher processes for the TCP/IP protocol to 5, and adds dispatcher processes for the TCP/IP with SSL (TCPS) protocol. There was no `DISPATCHERS` initialization parameter for the TCPS protocol (the only `DISPATCHERS` parameter was the one for the TCP protocol), so this statement effectively adds one.

```
ALTER SYSTEM
SET DISPATCHERS =
'(PROTOCOL=TCP)(DISPATCHERS=5)(INDEX=0)',
'(PROTOCOL=TCPS)(DISPATCHERS=2)(INDEX=1)';
```

If there are currently fewer than five dispatcher processes for TCP, Oracle creates new ones. If there are currently more than five, Oracle terminates some of them after the connected users disconnect.

Note: The `INDEX` keyword can be used to identify which `DISPATCHERS` parameter to modify. The `INDEX` value can range from 0 to n , where n is one less than the defined number of `DISPATCHERS` parameters. If your `ALTER SYSTEM` statement specifies an `INDEX` value equal to $n+1$, where n is the current number of dispatchers, a new `DISPATCHERS` parameter is added. To identify the index number assigned to an `DISPATCHERS` parameter, query the `CONF_INDXX` value in the `V$DISPATCHER` view.

Shutting Down Specific Dispatcher Processes

It is possible to shut down specific dispatcher processes. To identify the name of the specific dispatcher process to shut down, use the `V$DISPATCHER` dynamic performance view.

```
SELECT NAME, NETWORK FROM V$DISPATCHER;
```

```
NAME NETWORK
-----
```

```
D000 (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3499)
D001 (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3531)
D002 (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3532)
```

Each dispatcher is uniquely identified by a name of the form *Dnnn*.

To shut down dispatcher D002, issue the following statement:

```
ALTER SYSTEM SHUTDOWN IMMEDIATE 'D002';
```

The `IMMEDIATE` keyword stops the dispatcher from accepting new connections and Oracle immediately terminates all existing connections through that dispatcher. After all sessions are cleaned up, the dispatcher process shuts down. If `IMMEDIATE` were not specified, the dispatcher would wait until all of its users disconnected and all of its connections terminated before shutting down.

Changing the Minimum Number of Shared Server Processes

After starting an instance, you can change the minimum number of shared server processes by using the SQL statement `ALTER SYSTEM`. Oracle will eventually terminate servers that are idle when there are more shared servers than the minimum limit you specify.

If you set `SHARED_SERVERS` to 0, Oracle terminates all current servers when they become idle and does not start any new servers until you increase `SHARED_SERVERS`. Thus, setting `SHARED_SERVERS` to 0 may be used to effectively disable shared server.

The following statement dynamically sets the minimum number of shared server processes to two:

```
ALTER SYSTEM SET SHARED_SERVERS = 2;
```

Monitoring Shared Server

The following are useful views for obtaining information about your shared server configuration and for monitoring performance.

View	Description
V\$DISPATCHER	Provides information on the dispatcher processes, including name, network address, status, various usage statistics, and index number.
V\$DISPATCHER_RATE	Provides rate statistics for the dispatcher processes.

View	Description
V\$QUEUE	Contains information on the shared server message queues.
V\$SHARED_SERVER	Contains information on the shared server processes.
V\$CIRCUIT	Contains information about virtual circuits, which are user connections to the database through dispatchers and servers.
V\$SHARED_SERVER_MONITOR	Contains information for tuning shared server.
V\$SGA	Contains size information about various system global area (SGA) groups. May be useful when tuning shared server.
V\$SGASTAT	Detailed statistical information about the SGA, useful for tuning.
V\$SHARED_POOL_RESERVED	Lists statistics to help tune the reserved pool and space within the shared pool.

See Also:

- *Oracle9i Database Reference* for a detailed description of these views
- *Oracle9i Database Performance Tuning Guide and Reference* for specific information about monitoring and tuning shared server

About Oracle Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle system uses some additional processes called background processes. Background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. Background processes asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

The following are some basic Oracle background processes, many of which are discussed in more detail elsewhere in this book. The use of additional Oracle database server features or options can cause more background processes to be present. For example, if you use Advanced Queuing, the queue monitor (QMn) background process is present, or if you specified the `FILE_MAPPING` initialization parameter for mapping datafiles to physical devices on a storage subsystem, then the LMON process is present.

Process Name	Description
Database writer (DBWn)	<p>The database writer writes modified blocks from the database buffer cache to the datafiles. Oracle allows a maximum of 20 database writer processes (DBW0-DBW9 and DBWa-DBWj). The initialization parameter <code>DB_WRITER_PROCESSES</code> specifies the number of DBWn processes. Oracle selects an appropriate default setting for this initialization parameter (or might adjust a user specified setting) based upon the number of CPUs and the number of processor groups.</p> <p>For more information about setting the <code>DB_WRITER_PROCESSES</code> initialization parameter, see the <i>Oracle9i Database Performance Tuning Guide and Reference</i>.</p>
Log writer (LGWR)	<p>The log writer process writes redo log entries to disk. Redo log entries are generated in the redo log buffer of the system global area (SGA), and LGWR writes the redo log entries sequentially into an online redo log file. If the database has a multiplexed redo log, LGWR writes the redo log entries to a group of online redo log files. See Chapter 7, "Managing the Online Redo Log" for information about the log writer process.</p>
Checkpoint (CKPT)	<p>At specific times, all modified database buffers in the system global area are written to the datafiles by DBWn. This event is called a checkpoint. The checkpoint process is responsible for signalling DBWn at checkpoints and updating all the datafiles and control files of the database to indicate the most recent checkpoint.</p>
System monitor (SMON)	<p>The system monitor performs crash recovery when a failed instance starts up again. In a cluster database (Oracle9i Real Application Clusters), the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers dead transactions skipped during crash and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online.</p> <p>SMON also coalesces free extents within the database's dictionary-managed tablespaces to make free space contiguous and easier to allocate (see "Coalescing Free Space in Dictionary-Managed Tablespaces" on page 11-16).</p>
Process monitor (PMON)	<p>The process monitor performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on the dispatcher processes (see below) and server processes and restarts them if they have failed.</p>
Archiver (ARCn)	<p>One or more archiver processes copy the online redo log files to archival storage when they are full or a log switch occurs. Archiver processes are the subject of Chapter 8, "Managing Archived Redo Logs".</p>

Process Name	Description
Recoverer (RECO)	The recoverer process is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions. For information about this process and how to start it, see Chapter 32, "Managing Distributed Transactions" .
Dispatcher (Dnnn)	Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server was discussed previously in "Configuring Oracle for the Shared Server" on page 5-5.
Global Cache Service (LMS)	In an Oracle Real Application Clusters environment, this process manages resources and provides inter-instance resource control. See: <ul style="list-style-type: none"> ■ <i>Oracle9i Real Application Clusters Concepts</i> ■ <i>Oracle9i Real Application Clusters Setup and Configuration</i>.
Coordinator job queue process (CJQ0)	This is the coordinator of job queue processes for an instance. It monitors the JOBS table (table of jobs in the job queue) and starts job queue processes (Jnnn) as needed to execute jobs. The Jnnn processes execute job requests created by the DBMS_JOBS package. This is the subject of Chapter 10, "Managing Job Queues" . Additionally, up to 1000 Jnnn processes can automatically refresh materialized views. They wake up periodically and refresh any materialized views that are scheduled to be refreshed. For information about creating and refreshing materialized views, see: <ul style="list-style-type: none"> ■ <i>Oracle9i Replication</i> ■ <i>Oracle9i Replication Management API Reference</i>. <p>Yet another function of the Jnnn processes is to propagate queued messages to queues on other databases. See <i>Oracle9i Application Developer's Guide - Advanced Queuing</i> for information on propagating queued messages.</p> <p>Unlike many Oracle background processes, if a job queue process or the coordinator (CJQ0) fails, it does not cause instance failure.</p>

See Also: *Oracle9i Database Concepts* for more information about Oracle's background processes

Monitoring the Processes of an Oracle Instance

This section lists some of the data dictionary views that you can use to monitor an Oracle instance. These views are more general in their scope. There are other views, more specific to a process, that are discussed in the section of this book where the

process is described. Also presented are scripts and a view for monitoring the status of locks.

See Also:

- *Oracle9i Database Reference* contains detailed descriptions of these views
- *Oracle9i Database Performance Tuning Guide and Reference* provides information for resolving performance problems and conflicts that may be revealed through the monitoring of these views

Process and Session Views

These views provide process and session specific information:

View	Description
V\$PROCESS	Contains information about the currently active processes
V\$SESSION	Lists session information for each current session
V\$SESS_IO	Contains I/O statistics for each user session
V\$SESSION_LONGOPS	This view displays the status of various operations that run for longer than 6 seconds (in absolute time). These operations currently include many backup and recovery functions, statistics gathering, and query execution. More operations are added for every Oracle release.
V\$SESSION_WAIT	Lists the resources or events for which active sessions are waiting
V\$SYSSTAT	Contains session statistics
V\$RESOURCE_LIMIT	Provides information about current and maximum global resource utilization for some system resources
V\$SQLAREA	Contains statistics about shared SQL area and contains one row for each SQL string. Also provides statistics about SQL statements that are in memory, parsed, and ready for execution
V\$LATCH	Contains statistics for non-parent latches and summary statistics for parent latches

Monitoring Locks

The `utllockt.sql` script displays, in tree-structured fashion, the sessions in the system that are waiting for locks and the locks that they are waiting for. Using an ad hoc query tool, such as SQL*Plus, the script prints the sessions in the system that are waiting for locks and the corresponding blocking locks. The location of this script file is operating system dependent; see your operating system specific Oracle documentation. A second script, `catblock.sql`, creates the lock views that `utllockt.sql` needs, so you must run it before running `utllockt.sql`.

The following view can be used for monitoring locks:

View	Description
V\$LOCK	Lists the locks currently held by the Oracle server and outstanding requests for a lock or latch

Trace Files and the Alert File

Each server and background process can write to an associated **trace file**. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, while other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

The **alert file**, or **alert log**, is a special trace file. The alert file of a database is a chronological log of messages and errors, which includes the following:

- All internal errors (ORA-600), block corruption errors (ORA-1578), and deadlock errors (ORA-60) that occur
- Administrative operations, such as CREATE, ALTER, and DROP statements and STARTUP, SHUTDOWN, and ARCHIVELOG statements
- Several messages and errors relating to the functions of shared server and dispatcher processes
- Errors occurring during the automatic refresh of a materialized view
- The values of all initialization parameters at the time the database and instance start

Oracle uses the alert file to keep a log of these special operations as an alternative to displaying such information on an operator's console (although many systems display information on the console). If an operation is successful, a "completed" message is written in the alert file, along with a timestamp.

Initialization parameters controlling the location and size of trace files are:

- BACKGROUND_DUMP_DEST
- USER_DUMP_DEST
- MAX_DUMP_FILE_SIZE

These parameters are discussed in the sections that follow.

See Also: *Oracle9i Database Reference* for information about initialization parameters that control the writing to trace files

Using the Trace Files

You should periodically check the alert file and other trace files of an instance to see if the background processes have encountered errors. For example, when the Log Writer process (LGWR) cannot write to a member of a group, an error message indicating the nature of the problem is written to the LGWR trace file and the database's alert file. If you see such error messages, a media or I/O problem has occurred, and should be corrected immediately.

Oracle also writes values of initialization parameters to the alert file, in addition to other important statistics. For example, when you shut down an instance normally or immediately, Oracle writes the highest number of sessions concurrently connected to the instance, since the instance started, to the alert file. You can use this number to see if you need to upgrade your Oracle session license.

Specifying the Location of Trace Files

All trace files for background processes and the alert file are written to the destination directory specified by the initialization parameter `BACKGROUND_DUMP_DEST`. All trace files for server processes are written to the destination directory specified by the initialization parameter `USER_DUMP_DEST`. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file (such as LGWR and RECO).

See Also: Your operating system specific Oracle documentation for information about the names of trace files

Controlling the Size of Trace Files

You can control the maximum size of all trace files (excluding the alert file) using the initialization parameter `MAX_DUMP_FILE_SIZE`. This limit is set as a number of operating system blocks. To control the size of an alert file, you must manually delete the file when you no longer need it; otherwise Oracle continues to append to

the file. You can safely delete the alert file while the instance is running, although you might want to make an archived copy of it first.

Controlling When Oracle Writes to Trace Files

Background processes always write to a trace file when appropriate. In the case of the ARC*n* background process, it is possible, through an initialization parameter, to control the amount and type of trace information that is produced. This is described in "[Controlling Trace Output Generated by the Archivelog Process](#)" on page 8-21. Other background processes do not have this flexibility.

Trace files are written on behalf of server processes whenever internal errors occur. Additionally, setting the initialization parameter `SQL_TRACE = TRUE` causes the SQL trace facility to generate performance statistics for the processing of all SQL statements for an *instance* and write them to the `USER_DUMP_DEST` directory.

Optionally, trace files can be generated for server processes at user request. Regardless of the current value of the `SQL_TRACE` initialization parameter, each session can enable or disable trace logging on behalf of the associated server process by using the SQL statement `ALTER SESSION SET SQL_TRACE`. This example enables the SQL trace facility for a specific session:

```
ALTER SESSION SET SQL_TRACE TRUE;
```

Caution: Because the SQL trace facility for server processes can cause significant system overhead resulting in severe performance impact, enable this feature only when collecting statistics.

For shared server, each session using a dispatcher is routed to a shared server process, and trace information is written to the server's trace file only if the session has enabled tracing (or if an error is encountered). Therefore, to track tracing for a specific session that connects using a dispatcher, you might have to explore several shared server's trace files.

The `DBMS_SESSION` and `DBMS_SYSTEM` packages can also be used to control SQL tracing for a session.

See Also: *Oracle9i Database Performance Tuning Guide and Reference* contains information about using the SQL trace facility and using `TKPROF` to interpret the generated trace files.

Managing Processes for Parallel Execution

This section describes how to manage parallel processing of SQL statements. In this configuration Oracle can divide the work of processing an SQL statement among multiple parallel processes.

The execution of many SQL statements can be parallelized. The **degree of parallelism** is the number of parallel execution servers that can be associated with a single operation. The degree of parallelism is determined by any of the following:

- A `PARALLEL` clause in a statement
- For objects referred to in a query, the `PARALLEL` clause that was used when the object was created or altered
- A parallel hint inserted into the statement
- A default determined by Oracle

An example of using parallel execution is contained in "[Parallelizing Table Creation](#)" on page 15-8.

The following topics are contained in this section:

- [Managing the Parallel Execution Servers](#)
- [Altering Parallel Execution for a Session](#)

Note: The parallel execution feature described in this section is available with the Oracle9i Enterprise Edition and Oracle9i Personal Edition.

See Also:

- *Oracle9i Database Concepts* and *Oracle9i Data Warehousing Guide* for additional information about parallel execution
- *Oracle9i Database Performance Tuning Guide and Reference* for information about using parallel hints

Managing the Parallel Execution Servers

With the parallel execution feature, a process known as the **parallel execution coordinator** dispatches the execution of a pool of **parallel execution servers** and coordinates the sending of results from all of these parallel execution servers back to the user. Parallel execution server processes remain associated with a statement

throughout its execution phase. When the statement is completely processed, these processes become available to process other statements.

Parallel execution can be tuned for you automatically by setting the initialization parameter `PARALLEL_AUTOMATIC_TUNING = TRUE`. With this setting, Oracle determines the default values for other initialization parameters that affect the performance of parallel execution.

Altering Parallel Execution for a Session

The `ALTER SESSION` statement can be used to control parallel execution for a session.

Disabling Parallel Execution

All subsequent DML (`INSERT`, `UPDATE`, `DELETE`), DDL (`CREATE`, `ALTER`), or query (`SELECT`) statements will not be parallelized after an `ALTER SESSION DISABLE PARALLEL DML | DDL | QUERY` statement is issued. They will be executed serially, regardless of any `PARALLEL` clause or parallel hints associated with the statement.

The following statement disables parallel DDL:

```
ALTER SESSION DISABLE PARALLEL DDL;
```

Enabling Parallel Execution

Where a `PARALLEL` clause or parallel hint is associated with a statement, those DML, DDL, or query statements will execute in parallel after an `ALTER SESSION ENABLE PARALLEL DML | DDL | QUERY` statement is issued. This is the default for DDL and query statements.

A DML statement can be parallelized *only* if you specifically issue this statement. The following statement enables parallel processing of DML statements:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Note: Parallel DML is available only if you have installed Oracle's Partitioning Option.

Forcing Parallel Execution

You can force parallel execution of all subsequent DML, DDL, or query statements for which parallelization is possible with the `ALTER SESSION FORCE PARALLEL DML | DDL | QUERY` statement. Additionally you can force a specific degree of

parallelism to be in effect, overriding any `PARALLEL` clause associated with subsequent statements. If you do not specify a degree of parallelism in this statement, the default degree of parallelism is used. However, a degree of parallelism specified in a statement through a hint will override the degree being forced.

The following statement forces parallel execution of subsequent statements and sets the overriding degree of parallelism to 5:

```
ALTER SESSION FORCE PARALLEL DDL PARALLEL 5;
```

To force the parallelization of DML, it must also be enabled as shown in "[Enabling Parallel Execution](#)".

Managing Processes for External Procedures

External procedures, are procedures that are called from another program, but are written in a different language. An example would be a PL/SQL program calling one or more C routines that are required to perform special-purpose processing.

These callable routines are stored in a dynamic link library (DLL), or `libunit` in the case of a Java class method, and are registered with the base language. Oracle provides a special-purpose interface, the call specification (call spec), that enables users to call external procedures from other languages.

Very briefly, to call an external procedure, the application must know the DLL or shared library in which the external procedure resides. It alerts a network listener process, which in turn starts an external procedure agent, which by default is named `extproc`. Using the network connection established by the listener, the application passes to the external procedure agent the name of the DLL, the name of the external procedure, and any parameters passed in by the application. Then, the external procedure agent loads the DLL and runs the external procedure and passes back to the application any values returned by the external procedure.

The agent must reside on the same computer as the database server.

To control access to DLLs, the database administrator grants `execute` privileges for the appropriate DLLs to application developers. The application developers write the external procedures and grant `execute` privilege on specific external procedures to other users.

Note: The external library (DLL file) must be statically linked. In other words, it must not reference any external symbols from other external libraries (DLL files). These symbols are not resolved and can cause your external procedure to fail.

The environment for calling external procedures, consisting of `tnsnames.ora` and `listener.ora` entries, is configured by default during the install of your database. You may need to perform additional network configuration steps for a higher level of security. These are documented in the *Oracle9i Net Services Administrator's Guide*.

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for information about external procedures

Terminating Sessions

In some situations, you might want to terminate current user sessions. For example, you might want to perform an administrative operation and need to terminate all non-administrative sessions.

This section describes the various aspects of terminating sessions, and contains the following topics:

- [Identifying Which Session to Terminate](#)
- [Terminating an Active Session](#)
- [Terminating an Inactive Session](#)

When a session is terminated, the session's transaction is rolled back and resources (such as locks and memory areas) held by the session are immediately released and available to other sessions.

Terminate a current session using the SQL statement `ALTER SYSTEM KILL SESSION`.

The following statement terminates the session whose system identifier is 7 and serial number is 15:

```
ALTER SYSTEM KILL SESSION '7,15';
```

Identifying Which Session to Terminate

To identify which session to terminate, specify the session's index number and serial number. To identify the system identifier (sid) and serial number of a session, query the `V$SESSION` dynamic performance view.

The following query identifies all sessions for the user `jward`:

```
SELECT SID, SERIAL#, STATUS
FROM V$SESSION
WHERE USERNAME = 'JWARD';
```

SID	SERIAL#	STATUS
7	15	ACTIVE
12	63	INACTIVE

A session is `ACTIVE` when it is making a SQL call to Oracle. A session is `INACTIVE` if it is not making a SQL call to Oracle.

See Also: *Oracle9i Database Reference* for a description of the status values for a session

Terminating an Active Session

If a user session is processing a transaction (`ACTIVE` status) when it is terminated, the transaction is rolled back and the user immediately receives the following message:

```
ORA-00028: your session has been killed
```

If, after receiving the `ORA-00028` message, a user submits additional statements before reconnecting to the database, Oracle returns the following message:

```
ORA-01012: not logged on
```

If an active session cannot be interrupted (it is performing network I/O or rolling back a transaction), the session cannot be terminated until the operation completes. In this case, the session holds all resources until it is terminated. Additionally, the session that issues the `ALTER SYSTEM` statement to terminate a session waits up to 60 seconds for the session to be terminated. If the operation that cannot be interrupted continues past one minute, the issuer of the `ALTER SYSTEM` statement receives a message indicating that the session has been "marked" to be terminated. A session marked to be terminated is indicated in `V$SESSION` with a status of `KILLED` and a server that is something other than `PSEUDO`.

Terminating an Inactive Session

If the session is not making a SQL call to Oracle (is `INACTIVE`) when it is terminated, the `ORA-00028` message is not returned immediately. The message is not returned until the user subsequently attempts to use the terminated session.

When an inactive session has been terminated, `STATUS` in the `V$SESSION` view is `KILLED`. The row for the terminated session is removed from `V$SESSION` after the user attempts to use the session again and receives the `ORA-00028` message.

In the following example, an inactive session is terminated. First, `V$SESSION` is queried to identify the `SID` and `SERIAL#` of the session, then the session is terminated.

```
SELECT SID,SERIAL#,STATUS,SERVER
       FROM V$SESSION
       WHERE USERNAME = 'JWARD';
```

SID	SERIAL#	STATUS	SERVER
7	15	INACTIVE	DEDICATED
12	63	INACTIVE	DEDICATED

2 rows selected.

```
ALTER SYSTEM KILL SESSION '7,15';
Statement processed.
```

```
SELECT SID, SERIAL#, STATUS, SERVER
       FROM V$SESSION
       WHERE USERNAME = 'JWARD';
```

SID	SERIAL#	STATUS	SERVER
7	15	KILLED	PSEUDO
12	63	INACTIVE	DEDICATED

2 rows selected.

Managing Control Files

This chapter explains how to create and maintain the control files for your database and contains the following topics:

- [What Is a Control File?](#)
- [Guidelines for Control Files](#)
- [Creating Control Files](#)
- [Troubleshooting After Creating Control Files](#)
- [Backing Up Control Files](#)
- [Recovering a Control File Using a Current Copy](#)
- [Dropping Control Files](#)
- [Displaying Control File Information](#)

See Also: [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating control files that are both created and managed by the Oracle database server

What Is a Control File?

Every Oracle database has a **control file**. A control file is a small binary file that records the physical structure of the database and includes:

- The database name
- Names and locations of associated datafiles and online redo log files
- The timestamp of the database creation
- The current log sequence number
- Checkpoint information

The control file must be available for writing by the Oracle database server whenever the database is open. Without the control file, the database cannot be mounted and recovery is difficult.

The control file of an Oracle database is created at the same time as the database. By default, at least one copy of the control file is created during database creation. On some operating systems the default is to create multiple copies. You should create two or more copies of the control file during database creation. You might also need to create control files later, if you lose control files or want to change particular settings in the control files.

Guidelines for Control Files

This section describes guidelines you can use to manage the control files for a database, and contains the following topics:

- [Provide Filenames for the Control Files](#)
- [Multiplex Control Files on Different Disks](#)
- [Place Control Files Appropriately](#)
- [Back Up Control Files](#)
- [Manage the Size of Control Files](#)

Provide Filenames for the Control Files

You specify control file names using the `CONTROL_FILES` initialization parameter in the database's initialization parameter file (see "[Creating Initial Control Files](#)" on page 6-4). The instance startup procedure recognizes and opens all the listed files.

The instance writes to and maintains all listed control files during database operation.

If you do not specify files for `CONTROL_FILES` before database creation, *and you are not using the Oracle Managed Files* feature, Oracle creates a control file and uses a default filename. The default name is operating system specific.

Multiplex Control Files on Different Disks

Every Oracle database should have *at least* two control files, each stored on a different disk. If a control file is damaged due to a disk failure, the associated instance must be shut down. Once the disk drive is repaired, the damaged control file can be restored using the intact copy of the control file from the other disk and the instance can be restarted. In this case, no media recovery is required.

The following describes the behavior of multiplexed control files:

- Oracle *writes* to all filenames listed for the initialization parameter `CONTROL_FILES` in the database's initialization parameter file.
- The first file listed in the `CONTROL_FILES` parameter is the only file *read* by the Oracle database server during database operation.
- If any of the control files become unavailable during database operation, the instance becomes inoperable and should be aborted.

Note: Oracle strongly recommends that your database has a minimum of two control files and that they are located on separate disks.

Place Control Files Appropriately

As already suggested, each copy of a control file should be stored on a different disk drive. One practice is to store a control file copy on every disk drive that stores members of online redo log groups, if the online redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the online redo log will be lost in a single disk failure.

Back Up Control Files

It is very important that you back up your control files. This is true initially, and at any time after you change the physical structure of your database. Such structural changes include:

- Adding, dropping, or renaming datafiles
- Adding or dropping a tablespace, or altering the read-write state of the tablespace
- Adding or dropping redo log files or groups

The methods for backing up control files are discussed in "[Backing Up Control Files](#)" on page 6-10.

Manage the Size of Control Files

The main determinants of a control file's size are the values set for the `MAXDATAFILES`, `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY`, and `MAXINSTANCES` parameters in the `CREATE DATABASE` statement that created the associated database. Increasing the values of these parameters increases the size of a control file of the associated database.

See Also:

- Your operating system specific Oracle documentation contains more information about the maximum control file size.
- *Oracle9i SQL Reference* for a description of the `CREATE DATABASE` statement

Creating Control Files

This section describes ways to create control files, and contains the following topics:

- [Creating Initial Control Files](#)
- [Creating Additional Copies, Renaming, and Relocating Control Files](#)
- [Creating New Control Files](#)

Creating Initial Control Files

The initial control files of an Oracle database are created when you issue the `CREATE DATABASE` statement. The names of the control files are specified by the `CONTROL_FILES` parameter in the initialization parameter file used during database creation. The filenames specified in `CONTROL_FILES` should be fully specified and are operating system specific. The following is an example of a `CONTROL_FILES` initialization parameter:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ct1,
```

```
/u02/oracle/prod/control02.ctl,  
/u03/oracle/prod/control03.ctl)
```

If files with the specified names currently exist at the time of database creation, you must specify the `CONTROLFILE REUSE` clause in the `CREATE DATABASE` statement, or else an error occurs. Also, if the size of the old control file differs from the `SIZE` parameter of the new one, you cannot use the `REUSE` option.

The size of the control file changes between some releases of Oracle, as well as when the number of files specified in the control file changes. Configuration parameters such as `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY`, `MAXDATAFILES`, and `MAXINSTANCES` affect control file size.

You can subsequently change the value of the `CONTROL_FILES` initialization parameter to add more control files or to change the names or locations of existing control files.

See Also: Your operating system specific Oracle documentation contains more information about specifying control files.

Creating Additional Copies, Renaming, and Relocating Control Files

You can create an additional control file copy by copying an existing control file to a new location and adding the file's name to the list of control files. Similarly, you rename an existing control file by copying the file to its new name or location, and changing the file's name in the control file list. In both cases, to guarantee that control files do not change during the procedure, shut down the instance before copying the control file.

To Multiplex or Move Additional Copies of the Current Control Files

1. Shut down the database.
2. Copy an existing control file to a different location, using operating system commands.
3. Edit the `CONTROL_FILES` parameter in the database's initialization parameter file to add the new control file's name, or to change the existing control filename.
4. Restart the database.

Creating New Control Files

This section discusses when and how to create new control files.

When to Create New Control Files

It is necessary for you to create new control files in the following situations:

- All control files for the database have been permanently damaged and you do not have a control file backup.
- You want to change one of the permanent database parameter settings originally specified in the `CREATE DATABASE` statement. These settings include the database's name and the following parameters: `MAXLOGFILES`, `MAXLOGMEMBERS`, `MAXLOGHISTORY`, `MAXDATAFILES`, and `MAXINSTANCES`.

For example, you would change a database's name if it conflicted with another database's name in a distributed environment. Or, as another example, you can change the value of `MAXLOGFILES` if the original setting is too low.

The CREATE CONTROLFILE Statement

You can create a new control file for a database using the `CREATE CONTROLFILE` statement. The following statement creates a new control file for the `prod` database (formerly a database that used a different database name):

```
CREATE CONTROLFILE
  SET DATABASE prod
  LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
                 '/u01/oracle/prod/redo01_02.log'),
  GROUP 2 ('/u01/oracle/prod/redo02_01.log',
          '/u01/oracle/prod/redo02_02.log'),
  GROUP 3 ('/u01/oracle/prod/redo03_01.log',
          '/u01/oracle/prod/redo03_02.log')
  NORESETLOGS
  DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
           '/u01/oracle/prod/rbs01.dbs' SIZE 5M,
           '/u01/oracle/prod/users01.dbs' SIZE 5M,
           '/u01/oracle/prod/temp01.dbs' SIZE 5M
  MAXLOGFILES 50
  MAXLOGMEMBERS 3
  MAXLOGHISTORY 400
  MAXDATAFILES 200
  MAXINSTANCES 6
  ARCHIVELOG;
```

Cautions:

- The `CREATE CONTROLFILE` statement can potentially damage specified datafiles and online redo log files. Omitting a filename can cause loss of the data in that file, or loss of access to the entire database. Employ caution when using this statement and be sure to follow the instructions in "[Steps for Creating New Control Files](#)".
 - If the database had forced logging enabled before creating the new control file, and you want it to continue to be enabled, then you must specify the `FORCE LOGGING` clause in the `CREATE CONTROLFILE` statement. See "[Specifying FORCE LOGGING Mode](#)" on page 2-29.
-
-

See Also: *Oracle9i SQL Reference* describes the complete syntax of the `CREATE CONTROLFILE` statement

Steps for Creating New Control Files

Complete the following steps to create a new control file.

1. Make a list of all datafiles and online redo log files of the database.

If you follow recommendations for control file backups as discussed in "[Backing Up Control Files](#)" on page 6-10, you will already have a list of datafiles and online redo log files that reflect the current structure of the database. However, if you have no such list, executing the following statements will produce one.

```
SELECT MEMBER FROM V$LOGFILE;  
SELECT NAME FROM V$DATAFILE;  
SELECT VALUE FROM V$PARAMETER WHERE NAME = 'CONTROL_FILES';
```

If you have no such lists and your control file has been damaged so that the database cannot be opened, try to locate all of the datafiles and online redo log files that constitute the database. Any files not specified in Step 5 are not recoverable once a new control file has been created. Moreover, if you omit any of the files that make up the `SYSTEM` tablespace, you might not be able to recover the database.

2. Shut down the database.

If the database is open, shut down the database normally if possible. Use the `IMMEDIATE` or `ABORT` options only as a last resort.

3. Back up all datafiles and online redo log files of the database.
4. Start up a new instance, but do not mount or open the database:

```
STARTUP NOMOUNT
```

5. Create a new control file for the database using the `CREATE CONTROLFILE` statement.

When creating a new control file, select the `RESETLOGS` option if you have lost any online redo log groups in addition to control files. In this case, you will need to recover from the loss of the redo logs (Step 8). You must also specify the `RESETLOGS` option if you have renamed the database. Otherwise, select the `NORESETLOGS` option.

6. Store a backup of the new control file on an offline storage device. See "[Backing Up Control Files](#)" on page 6-10 for instructions for creating a backup.
7. Edit the `CONTROL_FILES` initialization parameter for the database to indicate all of the control files now part of your database as created in Step 5 (not including the backup control file). If you are renaming the database, edit the `DB_NAME` parameter to specify the new name.
8. Recover the database if necessary. If you are not recovering the database, skip to Step 9.

If you are creating the control file as part of recovery, recover the database. If the new control file was created using the `NORESETLOGS` option (Step 5), you can recover the database with complete, closed database recovery.

If the new control file was created using the `RESETLOGS` option, you must specify `USING BACKUP CONTROL FILE`. If you have lost online or archived redo logs or datafiles, use the procedures for recovering those files.

9. Open the database using one of the following methods:
 - If you did not perform recovery, or you performed complete, closed database recovery in Step 8, open the database normally.

```
ALTER DATABASE OPEN;
```

- If you specified `RESETLOGS` when creating the control file, use the `ALTER DATABASE` statement, indicating `RESETLOGS`.

```
ALTER DATABASE OPEN RESETLOGS;
```

The database is now open and available for use.

See Also: *Oracle9i User-Managed Backup and Recovery Guide* contains additional information about:

- Listing database files
- Backing up all datafiles and online redo log files of the database
- Recovering online or archived redo log files
- Performing closed database recovery

Troubleshooting After Creating Control Files

After issuing the `CREATE CONTROLFILE` statement, you may encounter some common errors. This section describes the most common control file usage errors, and contains the following topics:

- [Checking for Missing or Extra Files](#)
- [Handling Errors During CREATE CONTROLFILE](#)

Checking for Missing or Extra Files

After creating a new control file and using it to open the database, check the alert file to see if Oracle has detected inconsistencies between the data dictionary and the control file, such as a datafile that the data dictionary includes but the control file does not list.

If a datafile exists in the data dictionary but not in the new control file, Oracle creates a placeholder entry in the control file under the name `MISSINGnnnn` (where `nnnn` is the file number in decimal). `MISSINGnnnn` is flagged in the control file as being offline and requiring media recovery.

The actual datafile corresponding to `MISSINGnnnn` can be made accessible by renaming `MISSINGnnnn` so that it points to the datafile only if the datafile was read-only or offline normal. If, on the other hand, `MISSINGnnnn` corresponds to a datafile that was not read-only or offline normal, then the rename operation cannot be used to make the datafile accessible, because the datafile requires media recovery that is precluded by the results of `RESETLOGS`. In this case, you must drop the tablespace containing the datafile.

In contrast, if a datafile indicated in the control file is not present in the data dictionary, Oracle removes references to it from the new control file. In both cases, Oracle includes an explanatory message in the alert.log file to let you know what was found.

Handling Errors During CREATE CONTROLFILE

If Oracle sends you an error (usually error ORA-01173, ORA-01176, ORA-01177, ORA-01215, or ORA-01216) when you attempt to mount and open the database after creating a new control file, the most likely cause is that you omitted a file from the `CREATE CONTROLFILE` statement or included one that should not have been listed. In this case, you should restore the files you backed up in Step 3 on page 6-8 and repeat the procedure from Step 4, using the correct filenames.

Backing Up Control Files

Use the `ALTER DATABASE BACKUP CONTROLFILE` statement to back up your control files. You have two options:

1. Back up the control file to a binary file (duplicate of existing control file) using the following statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/control.bkp';
```

2. Produce SQL statements that can later be used to re-create your control file:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

This command writes a SQL script to the database's trace file where it can be captured and edited to reproduce the control file.

See Also: For more information on backing up your control files, depending upon your backup strategy refer to one of the following books:

- *Oracle9i User-Managed Backup and Recovery Guide*
- *Oracle9i Recovery Manager User's Guide*

Recovering a Control File Using a Current Copy

This section presents ways that you can recover your control file from a current backup or from a multiplexed copy.

Recovering from Control File Corruption Using a Control File Copy

This procedure assumes that one of the control files specified in the `CONTROL_FILES` parameter is corrupted, the control file directory is still accessible, and you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to overwrite the bad control file with a good copy:

```
% cp /u01/oracle/prod/control03.ct1 /u01/oracle/prod/control02.ct1
```

2. Start SQL*Plus and open the database:

```
SQL> STARTUP
```

Recovering from Permanent Media Failure Using a Control File Copy

This procedure assumes that one of the control files specified in the `CONTROL_FILES` parameter is inaccessible due to a permanent media failure, and you have a multiplexed copy of the control file.

1. With the instance shut down, use an operating system command to copy the current copy of the control file to a new, accessible location:

```
% cp /u01/oracle/prod/control01.ct1 /u04/oracle/prod/control03.ct1
```

2. Edit the `CONTROL_FILES` parameter in the initialization parameter file to replace the bad location with the new location:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ct1,  
                /u02/oracle/prod/control02.ct1,  
                /u04/oracle/prod/control03.ct1)
```

3. Start SQL*Plus and open the database:

```
SQL> STARTUP
```

In any case where you have multiplexed control files, and you must get the database up in minimum time, you can do so by editing the `CONTROL_FILES` initialization parameter to remove the bad control file and restarting the database immediately. Then you can perform the reconstruction of the bad control file and at some later time shut down and restart the database after editing the `CONTROL_FILES` initialization parameter to include the recovered control file.

Dropping Control Files

You can drop control files from the database. For example, you might want to do so if the location of a control file is no longer appropriate. Remember that the database must have at least two control files at all times.

1. Shut down the database.

2. Edit the `CONTROL_FILES` parameter in the database's initialization parameter file to delete the old control file's name.
3. Restart the database.

Note: This operation does not physically delete the unwanted control file from the disk. Use operating system commands to delete the unnecessary file after you have dropped the control file from the database.

Displaying Control File Information

The following views display information about control files:

View	Description
V\$DATABASE	Displays database information from the control file
V\$CONTROLFILE	Lists the names of control files
V\$CONTROLFILE_RECORD_SECTION	Displays information about control file record sections
V\$PARAMETER	Can be used to display the names of control files as specified in the <code>CONTROL_FILES</code> initialization parameter

This example lists the names of the control files.

```
SQL> SELECT NAME FROM V$CONTROLFILE;
```

```
NAME
```

```
-----
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```

Managing the Online Redo Log

This chapter explains how to manage the online redo log and contains the following topics:

- [What Is the Online Redo Log?](#)
- [Planning the Online Redo Log](#)
- [Creating Online Redo Log Groups and Members](#)
- [Relocating and Renaming Online Redo Log Members](#)
- [Dropping Online Redo Log Groups and Members](#)
- [Forcing Log Switches](#)
- [Verifying Blocks in Redo Log Files](#)
- [Clearing an Online Redo Log File](#)
- [Viewing Online Redo Log Information](#)

See Also:

- [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating online redo log files that are both created and managed by the Oracle database server
- *Oracle9i Real Application Clusters Administration* for more information about managing the online redo logs of instances when using Oracle Real Application Clusters
- *Oracle9i Database Performance Tuning Guide and Reference* to learn how checkpoints and the redo log impact instance recovery

What Is the Online Redo Log?

The most crucial structure for recovery operations is the **online redo log**, which consists of two or more preallocated files that store all changes made to the database as they occur. Every instance of an Oracle database has an associated online redo log to protect the database in case of an instance failure.

Redo Threads

Each database instance has its own online **redo log groups**. These online redo log groups, multiplexed or not, are called an instance's **thread** of online redo. In typical configurations, only one database instance accesses an Oracle database, so only one thread is present. When running Oracle Real Application Clusters, however, two or more instances concurrently access a single database and each instance has its own thread.

This chapter describes how to configure and manage the online redo log when the Oracle9i Real Application Clusters feature is *not* used. Hence, the thread number can be assumed to be 1 in all discussions and examples of statements.

Online Redo Log Contents

Online redo log files are filled with **redo records**. A redo record, also called a **redo entry**, is made up of a group of **change vectors**, each of which is a description of a change made to a single block in the database. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the data segment block for the table, the rollback segment data block, and the transaction table of the rollback segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the rollback segments. Therefore, the online redo log also protects rollback data. When you recover the database using redo data, Oracle reads the change vectors in the redo records and applies the changes to the relevant blocks.

Redo records are buffered in a circular fashion in the redo log buffer of the SGA (see "[How Oracle Writes to the Online Redo Log](#)") and are written to one of the online redo log files by the Oracle background process Log Writer (LGWR). Whenever a transaction is committed, LGWR writes the transaction's redo records from the redo log buffer of the SGA to an online redo log file, and a **system change number (SCN)** is assigned to identify the redo records for each committed transaction. Only when all redo records associated with a given transaction are safely on disk in the online log is the user process notified that the transaction has been committed.

Redo records can also be written to an online redo log file before the corresponding transaction is committed. If the redo log buffer fills, or another transaction commits, LGWR flushes all of the redo log entries in the redo log buffer to an online redo log file, even though some redo records may not be committed. If necessary, Oracle can roll back these changes.

How Oracle Writes to the Online Redo Log

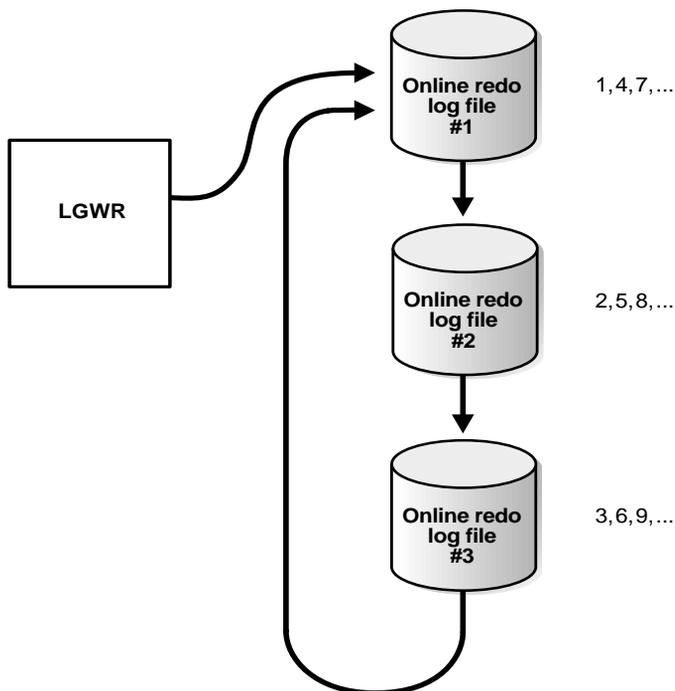
The online redo log of a database consists of two or more online redo log files. Oracle requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if in `ARCHIVELOG` mode).

LGWR writes to online redo log files in a circular fashion. When the current online redo log file fills, LGWR begins writing to the next available online redo log file. When the last available online redo log file is filled, LGWR returns to the first online redo log file and writes to it, starting the cycle again. [Figure 7-1](#) illustrates the circular writing of the online redo log file. The numbers next to each line indicate the sequence in which LGWR writes to each online redo log file.

Filled online redo log files are available to LGWR for reuse depending on whether archiving is enabled.

- If archiving is disabled (`NOARCHIVELOG` mode), a filled online redo log file is available once the changes recorded in it have been written to the datafiles.
- If archiving is enabled (`ARCHIVELOG` mode), a filled online redo log file is available to LGWR once the changes recorded in it have been written to the datafiles *and* once the file has been archived.

Figure 7-1 Circular Use of Online Redo Log Files by LGWR



Active (Current) and Inactive Online Redo Log Files

At any given time, Oracle uses only one of the online redo log files to store redo records written from the redo log buffer. The online redo log file that LGWR is actively writing to is called the **current** online redo log file.

Online redo log files that are required for instance recovery are called **active** online redo log files. Online redo log files that are not required for instance recovery are called **inactive**.

If you have enabled archiving (ARCHIVELOG mode), Oracle cannot reuse or overwrite an active online log file until ARC*n* has archived its contents. If archiving is disabled (NOARCHIVELOG mode), when the last online redo log file fills writing continues by overwriting the first available active file.

Log Switches and Log Sequence Numbers

A **log switch** is the point at which Oracle ends writing to one online redo log file and begins writing to another. Normally, a log switch occurs when the current online redo log file is completely filled and writing must continue to the next online redo log file. However, you can specify that a log switch occurs in a time-based manner, regardless of whether the current online redo log file is completely filled. You can also force log switches manually.

Oracle assigns each online redo log file a new **log sequence number** every time that a log switch occurs and LGWR begins writing to it. If Oracle archives online redo log files, the archived log retains its log sequence number. The online redo log file that is cycled back for use is given the next available log sequence number.

Each online or archived redo log file is uniquely identified by its log sequence number. During crash, instance, or media recovery, Oracle properly applies redo log files in ascending order by using the log sequence number of necessary archived and online redo log files.

Planning the Online Redo Log

This section describes guidelines you should consider when configuring a database instance's online redo log, and contains the following topics:

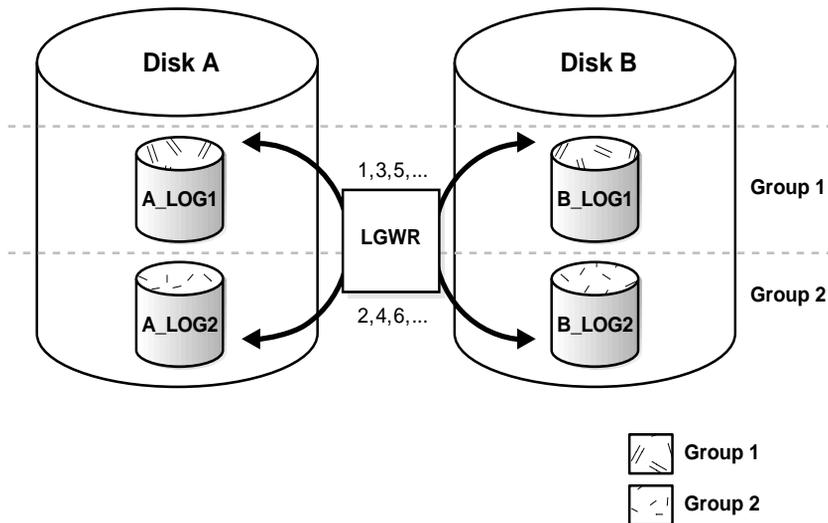
- [Multiplexing Online Redo Log Files](#)
- [Placing Online Redo Log Members on Different Disks](#)
- [Setting the Size of Online Redo Log Members](#)
- [Choosing the Number of Online Redo Log Files](#)
- [Controlling Archive Lag](#)

Multiplexing Online Redo Log Files

Oracle provides the capability to **multiplex** an instance's online redo log files to safeguard against damage to its online redo log files. When multiplexing online redo log files, LGWR concurrently writes the same redo log information to multiple identical online redo log files, thereby eliminating a single point of redo log failure.

Note: Oracle recommends that you multiplex your redo log files. The loss of the log file data can be catastrophic if recovery is required.

Figure 7-2 Multiplexed Online Redo Log Files



The corresponding online redo log files are called **groups**. Each online redo log file in a group is called a **member**. In [Figure 7-2](#), A_LOG1 and B_LOG1 are both members of Group 1, A_LOG2 and B_LOG2 are both members of Group 2, and so forth. Each member in a group must be exactly the same size.

Notice that each member of a group is concurrently active, or, concurrently written to by LGWR, as indicated by the identical log sequence numbers assigned by LGWR. In [Figure 7-2](#), first LGWR writes to A_LOG1 in conjunction with B_LOG1, then A_LOG2 in conjunction with B_LOG2, and so on. LGWR never writes concurrently to members of different groups (for example, to A_LOG1 and B_LOG2).

Responding to Online Redo Log Failure

Whenever LGWR cannot write to a member of a group, Oracle marks that member as `INVALID` and writes an error message to the LGWR trace file and to the database's alert file to indicate the problem with the inaccessible files. LGWR reacts differently when certain online redo log members are unavailable, depending on the reason for the unavailability.

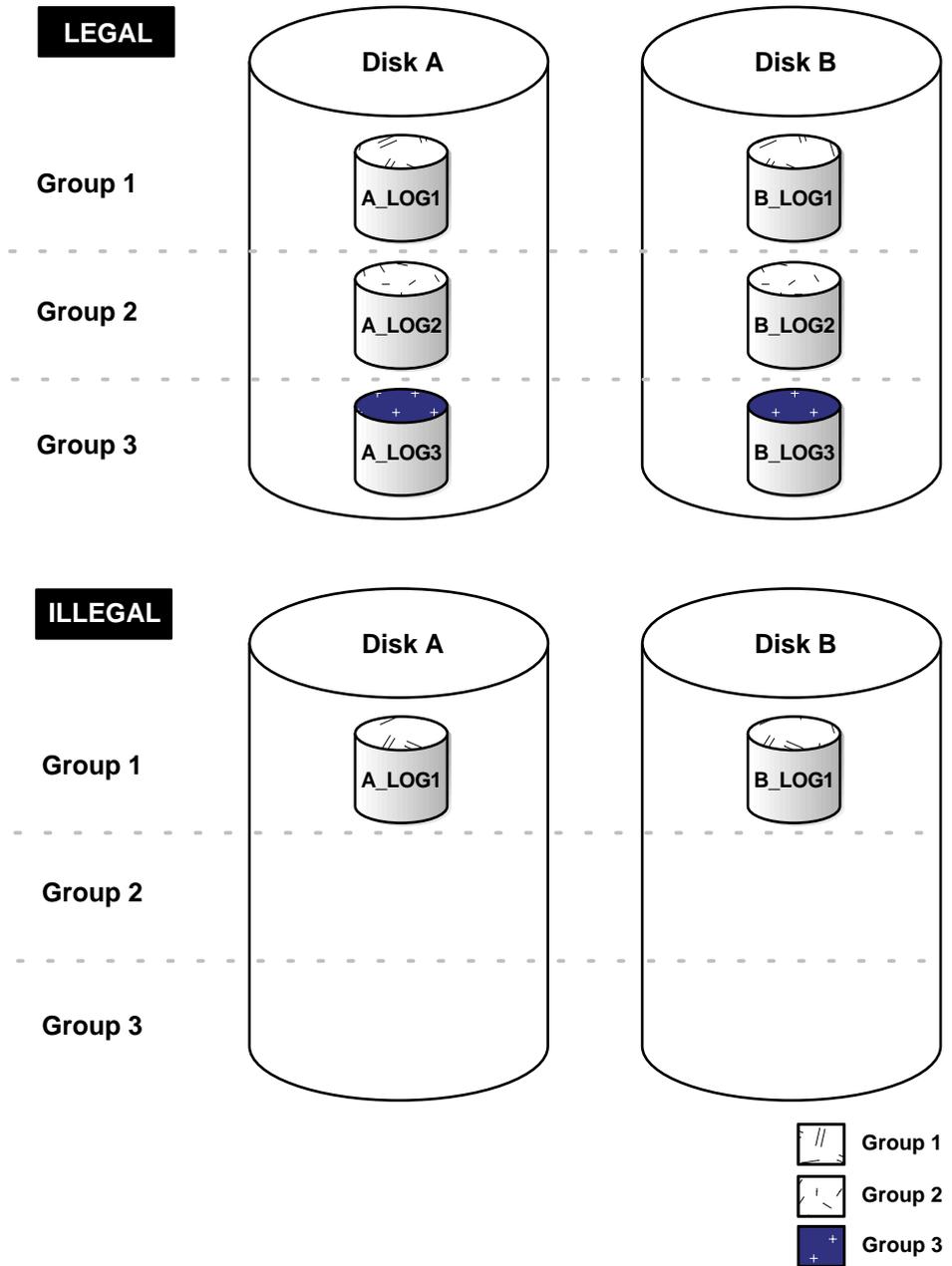
If	Then
LGWR can successfully write to at least one member in a group	Writing proceeds as normal. LGWR simply writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group needs to be archived	Database operation temporarily halts until the group becomes available, or, until the group is archived.
All members of the next group are inaccessible to LGWR at a log switch because of media failure	<p>Oracle returns an error and the database instance shuts down. In this case, you may need to perform media recovery on the database from the loss of an online redo log file.</p> <p>If the database checkpoint has moved beyond the lost redo log, media recovery is not necessary since Oracle has saved the data recorded in the redo log to the datafiles. Simply drop the inaccessible redo log group. If Oracle did not archive the bad log, use <code>ALTER DATABASE CLEAR UNARCHIVED LOG</code> to disable archiving before the log can be dropped.</p>
If all members of a group suddenly become inaccessible to LGWR while it is writing to them	Oracle returns an error and the database instance immediately shuts down. In this case, you may need to perform media recovery. If the media containing the log is not actually lost—for example, if the drive for the log was inadvertently turned off—media recovery may not be needed. In this case, you only need to turn the drive back on and let Oracle perform instance recovery.

Legal and Illegal Configurations

To safeguard against a single point of online redo log failure, a multiplexed online redo log is ideally symmetrical: all groups of the online redo log have the same number of members. Nevertheless, Oracle does not *require* that a multiplexed online redo log be symmetrical. For example, one group can have only one member, while other groups have two members. This configuration protects against disk failures that temporarily affect some online redo log members but leave others intact.

The only requirement for an instance's online redo log is that it have at least two groups. [Figure 7-3](#) shows legal and illegal multiplexed online redo log configurations. The second configuration is illegal because it has only one group.

Figure 7-3 Legal and Illegal Multiplexed Online Redo Log Configuration



Placing Online Redo Log Members on Different Disks

When setting up a multiplexed online redo log, place members of a group on different disks. If a single disk fails, then only one member of a group becomes unavailable to LGWR and other members remain accessible to LGWR, so the instance can continue to function.

If you archive the redo log, spread online redo log members across disks to eliminate contention between the LGWR and ARC*n* background processes. For example, if you have two groups of duplexed online redo log members, place each member on a different disk and set your archiving destination to a fifth disk. Consequently, there is never contention between LGWR (writing to the members) and ARC*n* (reading the members).

Datafiles and online redo log files should also be on different disks to reduce contention in writing data blocks and redo records.

See Also: *Oracle9i Backup and Recovery Concepts* for more information about how the online redo log affects backup and recovery

Setting the Size of Online Redo Log Members

When setting the size of online redo log files, consider whether you will be archiving the redo log. Online redo log files should be sized so that a filled group can be archived to a single unit of offline storage media (such as a tape or disk), with the least amount of space on the medium left unused. For example, suppose only one filled online redo log group can fit on a tape and 49% of the tape's storage capacity remains unused. In this case, it is better to decrease the size of the online redo log files slightly, so that two log groups could be archived for each tape.

With multiplexed groups of online redo logs, all members of the same group must be the same size. Members of different groups can have different sizes. However, there is no advantage in varying file size between groups. If checkpoints are not set to occur between log switches, make all groups the same size to guarantee that checkpoints occur at regular intervals.

See Also: Your operating system specific Oracle documentation. The default size of online redo log files is operating system dependent.

Choosing the Number of Online Redo Log Files

The best way to determine the appropriate number of online redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR's writing redo log information.

In some cases, a database instance may require only two groups. In other situations, a database instance may require additional groups to guarantee that a recycled group is always available to LGWR. During testing, the easiest way to determine if the current online redo log configuration is satisfactory is to examine the contents of the LGWR trace file and the database's alert log. If messages indicate that LGWR frequently has to wait for a group because a checkpoint has not completed or a group has not been archived, add groups.

Consider the parameters that can limit the number of online redo log files before setting up or altering the configuration of an instance's online redo log. The following parameters limit the number of online redo log files that you can add to a database:

- The `MAXLOGFILES` parameter used in the `CREATE DATABASE` statement determines the maximum number of groups of online redo log files for each database. Group values can range from 1 to `MAXLOGFILES`. The only way to override this upper limit is to re-create the database or its control file. Thus, it is important to consider this limit *before* creating a database. If `MAXLOGFILES` is not specified for the `CREATE DATABASE` statement, Oracle uses an operating system specific default value.
- The `MAXLOGMEMBERS` parameter used in the `CREATE DATABASE` statement determines the maximum number of members for each group. As with `MAXLOGFILES`, the only way to override this upper limit is to re-create the database or control file. Thus, it is important to consider this limit *before* creating a database. If no `MAXLOGMEMBERS` parameter is specified for the `CREATE DATABASE` statement, Oracle uses an operating system default value.

See Also: Your operating system specific Oracle documentation for the default and legal values of the `MAXLOGFILES` and `MAXLOGMEMBERS` parameters

Controlling Archive Lag

You can force all enabled online redo log threads to switch their current logs in a time-based fashion. In a primary/standby configuration, changes are made available to the standby database by archiving and shipping logs of the primary site

to the standby database. The changes that are being applied by the standby database can lag the changes that are occurring on the primary database.

This lag can happen because the standby database must wait for the changes in the primary database's online redo log to be archived (into the archived redo log) and then shipped to it. To control or limit this lag, you set the `ARCHIVE_LAG_TARGET` initialization parameter. Setting this parameter allows you to limit, measured in time, how long the lag can become.

Setting the `ARCHIVE_LAG_TARGET` Initialization Parameter

When you set the `ARCHIVE_LAG_TARGET` initialization parameter, you cause Oracle to examine an instance's current online redo log periodically. If the following conditions are met the instance will switch the log:

- The current log was created prior to n seconds ago, and the estimated archival time for the current log is m seconds (proportional to the number of redo blocks used in the current log), where $n + m$ exceeds the value of the `ARCHIVE_LAG_TARGET` initialization parameter.
- The current log contains redo records.

In an Oracle Real Application Clusters environment, the instance also nudges other threads into switching and archiving logs if they are falling behind. This can be particularly useful when one instance in the cluster is more idle than the other instances (as when you are running a 2-node primary/secondary configuration of Oracle Real Application Clusters).

Initialization parameter `ARCHIVE_LAG_TARGET` specifies the target of how many seconds of redo the standby could lose in the event of a primary shutdown or crash if the Data Guard environment is not configured in a no-data-loss mode. It also provides an upper limit of how long (in the number of seconds) the current log of the primary database can span. Because the estimated archival time is also considered, this is not the exact log switch time.

The following initialization parameter setting sets the log switch interval to 30 minutes (a typical value).

```
ARCHIVE_LAG_TARGET = 1800
```

A value of 0 disables this time-based log switching functionality. This is the default setting.

You can set the `ARCHIVE_LAG_TARGET` initialization parameter even if there is no standby database. For example, the `ARCHIVE_LAG_TARGET` parameter can be set specifically to force logs to be switched and archived.

`ARCHIVE_LAG_TARGET` is a dynamic parameter and can be set with the `ALTER SYSTEM SET` statement.

Caution: The `ARCHIVE_LAG_TARGET` parameter must be set to the same value in all instances of an Oracle Real Application Clusters environment. Failing to do so results in unspecified behavior and is strongly discouraged.

Factors Affecting the Setting of `ARCHIVE_LAG_TARGET`

Consider the following factors when determining if you want to set the `ARCHIVE_LAG_TARGET` parameter and in determining the value for this parameter.

- Overhead of switching (as well as archiving) logs
- How frequently normal log switches occur as a result of log full conditions
- How much redo loss is tolerated in the standby database

Setting `ARCHIVE_LAG_TARGET` may not be very useful if natural log switches already occur more frequently than the interval specified. However, in the case of irregularities of redo generation speed, the interval does provide an upper limit for the time range each current log covers.

If the `ARCHIVE_LAG_TARGET` initialization parameter is set to a very low value, there can be a negative impact on performance. This can force frequent log switches. Set the parameter to a reasonable value so as not to degrade the performance of the primary database.

Creating Online Redo Log Groups and Members

Plan the online redo log of a database and create all required groups and members of online redo log files during database creation. However, there are situations where you might want to create additional groups or members. For example, adding groups to an online redo log can correct redo log group availability problems.

To create new online redo log groups and members, you must have the `ALTER DATABASE` system privilege. A database can have up to `MAXLOGFILES` groups.

See Also: *Oracle9i SQL Reference* for a complete description of the `ALTER DATABASE` statement

Creating Online Redo Log Groups

To create a new group of online redo log files, use the SQL statement `ALTER DATABASE` with the `ADD LOGFILE` clause.

The following statement adds a new group of redo logs to the database:

```
ALTER DATABASE
  ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500K;
```

Note: Use fully specify filenames of new log members to indicate where the operating system file should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system.

You can also specify the number that identifies the group using the `GROUP` option:

```
ALTER DATABASE
  ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo')
  SIZE 500K;
```

Using group numbers can make administering redo log groups easier. However, the group number must be between 1 and `MAXLOGFILES`. Do not skip redo log file group numbers (that is, do not number your groups 10, 20, 30, and so on), or you will consume space in the control files of the database.

Creating Online Redo Log Members

In some cases, it might not be necessary to create a complete group of online redo log files. A group could already exist, but not be complete because one or more members of the group were dropped (for example, because of a disk failure). In this case, you can add new members to an existing group.

To create new online redo log members for an existing group, use the SQL statement `ALTER DATABASE` with the `ADD LOG MEMBER` parameter. The following statement adds a new redo log member to redo log group number 2:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.rdo' TO GROUP 2;
```

Notice that filenames must be specified, but sizes need not be. The size of the new members is determined from the size of the existing members of the group.

When using the `ALTER DATABASE` statement, you can alternatively identify the target group by specifying all of the other members of the group in the `TO` parameter, as shown in the following example:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo'  
TO ('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

Note: Fully specify the filenames of new log members to indicate where the operating system files should be created. Otherwise, the files will be created in either the default or current directory of the database server, depending upon your operating system. You may also note that the status of the new log member is shown as `INVALID`. This is normal and it will change to active (blank) when it is first used.

Relocating and Renaming Online Redo Log Members

You can use operating system commands to relocate online redo logs, then use the `ALTER DATABASE` statement to make their new names (locations) known to the database. This procedure is necessary, for example, if the disk currently used for some online redo log files is going to be removed, or if datafiles and a number of online redo log files are stored on the same disk and should be separated to reduce contention.

To rename online redo log members, you must have the `ALTER DATABASE` system privilege. Additionally, you might also need operating system privileges to copy files to the desired location and privileges to open and back up the database.

Before relocating your redo logs, or making any other structural changes to the database, completely back up the database in case you experience problems while performing the operation. As a precaution, after renaming or relocating a set of online redo log files, immediately back up the database's control file.

Use the following steps for relocating redo logs. The example used to illustrate these steps assumes:

- The log files are located on two disks: `diska` and `diskb`.
- The online redo log is duplexed: one group consists of the members `/diska/logs/log1a.rdo` and `/diskb/logs/log1b.rdo`, and the second group consists of the members `/diska/logs/log2a.rdo` and `/diskb/logs/log2b.rdo`.

- The online redo log files located on `diska` must be relocated to `diskc`. The new filenames will reflect the new location: `/diskc/logs/log1c.rdo` and `/diskc/logs/log2c.rdo`.

Steps for Renaming Online Redo Log Members

1. Shut down the database.

```
SHUTDOWN
```

2. Copy the online redo log files to the new location.

Operating system files, such as online redo log members, must be copied using the appropriate operating system commands. See your operating system specific documentation for more information about copying files.

Note: You can execute an operating system command to copy a file (or perform other operating system commands) without exiting SQL*Plus by using the `HOST` command. Some operating systems allow you to use a character in place of the word `HOST`. For example, you can use `!` in UNIX.

The following example uses operating system commands (UNIX) to move the online redo log members to a new location:

```
mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
```

3. Startup the database, mount, but do not open it.

```
CONNECT / as SYSDBA
STARTUP MOUNT
```

4. Rename the online redo log members.

Use the `ALTER DATABASE` statement with the `RENAME FILE` clause to rename the database's online redo log files.

```
ALTER DATABASE
  RENAME FILE '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
  TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
```

5. Open the database for normal operation.

The online redo log alterations take effect when the database is opened.

```
ALTER DATABASE OPEN;
```

Dropping Online Redo Log Groups and Members

In some cases, you may want to drop an entire group of online redo log members. For example, you want to reduce the number of groups in an instance's online redo log. In a different case, you may want to drop one or more specific online redo log members. For example, if a disk failure occurs, you may need to drop all the online redo log files on the failed disk so that Oracle does not try to write to the inaccessible files. In other situations, particular online redo log files become unnecessary. For example, a file might be stored in an inappropriate location.

Dropping Log Groups

To drop an online redo log group, you must have the `ALTER DATABASE` system privilege. Before dropping an online redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of online redo log files, regardless of the number of members in the groups. (A group is one or more members.)
- You can drop an online redo log group only if it is inactive. If you need to drop the current group, first force a log switch to occur.
- Make sure an online redo log group is archived (if archiving is enabled) before dropping it. To see whether this has happened, use the `V$LOG` view.

```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;
```

```

GROUP# ARC STATUS
----- --
1 YES ACTIVE
2 NO CURRENT
3 YES INACTIVE
4 YES INACTIVE

```

Drop an online redo log group with the SQL statement `ALTER DATABASE` with the `DROP LOGFILE` clause.

The following statement drops redo log group number 3:

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

When an online redo log group is dropped from the database, and you are not using the Oracle Managed Files feature, the operating system files are not deleted

from disk. Rather, the control files of the associated database are updated to drop the members of the group from the database structure. After dropping an online redo log group, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped online redo log files.

When using Oracle-managed files, the cleanup of operating systems files is done automatically for you.

Dropping Online Redo Log Members

To drop an online redo log member, you must have the `ALTER DATABASE` system privilege. Consider the following restrictions and precautions before dropping individual online redo log members:

- It is permissible to drop online redo log files so that a multiplexed online redo log becomes temporarily asymmetric. For example, if you use duplexed groups of online redo log files, you can drop one member of one group, even though all other groups have two members each. However, you should rectify this situation immediately so that all groups have at least two members, and thereby eliminate the single point of failure possible for the online redo log.
- An instance always requires at least two valid groups of online redo log files, regardless of the number of members in the groups. (A group is one or more members.) If the member you want to drop is the last valid member of the group, you cannot drop the member until the other members become valid. To see a redo log file's status, use the `V$LOGFILE` view. A redo log file becomes `INVALID` if Oracle cannot access it. It becomes `STALE` if Oracle suspects that it is not complete or correct. A stale log file becomes valid again the next time its group is made the active group.
- You can drop an online redo log member only if it is *not* part of an active or current group. If you want to drop a member of an active group, first force a log switch to occur.
- Make sure the group to which an online redo log member belongs is archived (if archiving is enabled) before dropping the member. To see whether this has happened, use the `V$LOG` view.

To drop specific inactive online redo log members, use the `ALTER DATABASE` statement with the `DROP LOGFILE MEMBER` clause.

The following statement drops the redo log `/oracle/dbs/log3c.rdo`:

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

When an online redo log member is dropped from the database, the operating system file is not deleted from disk. Rather, the control files of the associated database are updated to drop the member from the database structure. After dropping an online redo log file, make sure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped online redo log file.

To drop a member of an active group, you must first force a log switch.

Forcing Log Switches

A log switch occurs when LGWR stops writing to one online redo log group and starts writing to another. By default, a log switch occurs automatically when the current online redo log file group fills.

You can force a log switch to make the currently active group inactive and available for online redo log maintenance operations. For example, you want to drop the currently active group, but are not able to do so until the group is inactive. You may also wish to force a log switch if the currently active group needs to be archived at a specific time before the members of the group are completely filled. This option is useful in configurations with large online redo log files that take a long time to fill.

To force a log switch, you must have the `ALTER SYSTEM` privilege. Use the `ALTER SYSTEM` statement with the `SWITCH LOGFILE` clause.

The following statement forces a log switch:

```
ALTER SYSTEM SWITCH LOGFILE;
```

Verifying Blocks in Redo Log Files

You can configure Oracle to use checksums to verify blocks in the redo log files. If you set the initialization parameter `DB_BLOCK_CHECKSUM` to `TRUE`, block checking is enabled for all Oracle database blocks written to disk, including redo log blocks. The default value of `DB_BLOCK_CHECKSUM` is `FALSE`.

If you enable block checking, Oracle computes a checksum for each redo log block written to the current log. Oracle writes the checksum in the header of the block. Oracle uses the checksum to detect corruption in a redo log block. Oracle tries to verify the redo log block when it writes the block to an archive log file and when the block is read from an archived log during recovery.

If Oracle detects a corruption in a redo log block while trying to archive it, the system attempts to read the block from another member in the group. If the block is corrupted in all members the redo log group, then archiving cannot proceed.

Note: There is some overhead and decrease in database performance with `DB_BLOCK_CHECKSUM` enabled. Monitor your database performance to decide if the benefit of using data block checksums to detect corruption outweighs the performance impact.

See Also: *Oracle9i Database Reference* for a description of the `DB_BLOCK_CHECKSUM` initialization parameter

Clearing an Online Redo Log File

An online redo log file might become corrupted while the database is open, and ultimately stop database activity because archiving cannot continue. In this situation the `ALTER DATABASE CLEAR LOGFILE` statement can be used to reinitialize the file without shutting down the database.

The following statement clears the log files in redo log group number 3:

```
ALTER DATABASE CLEAR LOGFILE GROUP 3;
```

This statement overcomes two situations where dropping redo logs is not possible:

- If there are only two log groups
- The corrupt redo log file belongs to the current group

If the corrupt redo log file has not been archived, use the `UNARCHIVED` keyword in the statement.

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

This statement clears the corrupted redo logs and avoids archiving them. The cleared redo logs are available for use even though they were not archived.

If you clear a log file that is needed for recovery of a backup, then you can no longer recover from that backup. Oracle writes a message in the alert log describing the backups from which you cannot recover.

Note: If you clear an unarchived redo log file, you should make another backup of the database.

If you want to clear an unarchived redo log that is needed to bring an offline tablespace online, use the `UNRECOVERABLE DATAFILE` clause in the `ALTER DATABASE CLEAR LOGFILE` statement.

If you clear a redo log needed to bring an offline tablespace online, you will not be able to bring the tablespace online again. You will have to drop the tablespace or perform an incomplete recovery. Note that tablespaces taken offline normal do not require recovery.

Viewing Online Redo Log Information

Use the following views to display online redo log information.

View	Description
V\$LOG	Displays the redo log file information from the control file
V\$LOGFILE	Identifies redo log groups and members and member status
V\$LOG_HISTORY	Contains log history information

The following query returns the control file information about the online redo log for a database.

```
SELECT * FROM V$LOG;
```

GROUP#	THREAD#	SEQ	BYTES	MEMBERS	ARC	STATUS	FIRST_CHANGE#	FIRST_TIM
1	1	10605	1048576	1	YES	ACTIVE	11515628	16-APR-00
2	1	10606	1048576	1	NO	CURRENT	11517595	16-APR-00
3	1	10603	1048576	1	YES	INACTIVE	11511666	16-APR-00
4	1	10604	1048576	1	YES	INACTIVE	11513647	16-APR-00

To see the names of all of the member of a group, use a query similar to the following:

```
SELECT * FROM V$LOGFILE;
```

GROUP#	STATUS	MEMBER
-----	-----	-----

1	D:\ORANT\ORADATA\IDDB2\REDO04.LOG
2	D:\ORANT\ORADATA\IDDB2\REDO03.LOG
3	D:\ORANT\ORADATA\IDDB2\REDO02.LOG
4	D:\ORANT\ORADATA\IDDB2\REDO01.LOG

If STATUS is blank for a member, then the file is in use.

See Also: *Oracle9i Database Reference* for detailed information about these views

Managing Archived Redo Logs

This chapter describes how to archive redo data. It contains the following topics:

- [What Is the Archived Redo Log?](#)
- [Choosing Between NOARCHIVELOG and ARCHIVELOG Mode](#)
- [Controlling Archiving](#)
- [Specifying the Archive Destination](#)
- [Specifying the Mode of Log Transmission](#)
- [Managing Archive Destination Failure](#)
- [Tuning Archive Performance by Specifying Multiple ARCn Processes](#)
- [Controlling Trace Output Generated by the Archivelog Process](#)
- [Viewing Information About the Archived Redo Log](#)

See Also: *Oracle9i Real Application Clusters Administration* for information specific to archiving in the Oracle Real Application Clusters environment

What Is the Archived Redo Log?

Oracle enables you to save filled groups of online redo log files to one or more offline destinations, known collectively as the **archived redo log**, or more simply archive log. The process of turning online redo log files into archived redo log files is called **archiving**. This process is only possible if the database is running in **ARCHIVELOG mode**. You can choose automatic or manual archiving.

An archived redo log file is a copy of one of the identical filled members of an online redo log group. It includes the redo entries present in the identical member of a redo log group and also preserves the group's unique log sequence number. For example, if you are multiplexing your online redo log, and if Group 1 contains member files `a_log1` and `b_log1`, then the archiver process (`ARCn`) will archive one of these identical members. Should `a_log1` become corrupted, then `ARCn` can still archive the identical `b_log1`. The archived redo log contains a copy of every group created since you enabled archiving.

When running in **ARCHIVELOG mode**, the log writer process (`LGWR`) is not allowed to reuse and hence overwrite an online redo log group until it has been archived. The background process `ARCn` automates archiving operations when automatic archiving is enabled. Oracle starts multiple archiver processes as needed to ensure that the archiving of filled online redo logs does not fall behind.

You can use archived redo logs to:

- Recover a database
- Update a standby database
- Gain information about the history of a database using the LogMiner utility

Choosing Between NOARCHIVELOG and ARCHIVELOG Mode

This section describes the issues you must consider when choosing to run your database in **NOARCHIVELOG** or **ARCHIVELOG** mode, and contains these topics:

- [Running a Database in NOARCHIVELOG Mode](#)
- [Running a Database in ARCHIVELOG Mode](#)

Running a Database in NOARCHIVELOG Mode

When you run your database in **NOARCHIVELOG** mode, you disable the archiving of the online redo log. The database's control file indicates that filled groups are not

required to be archived. Therefore, when a filled group becomes inactive after a log switch, the group is available for reuse by LGWR.

The choice of whether to enable the archiving of filled groups of online redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure, use ARCHIVELOG mode. The archiving of filled online redo log files can require you to perform extra administrative operations.

NOARCHIVELOG mode protects a database only from instance failure, but not from media failure. Only the most recent changes made to the database, which are stored in the groups of the online redo log, are available for instance recovery. In other words, if a media failure occurs while in NOARCHIVELOG mode, you can only *restore* (not *recover*) the database to the point of the most recent full database backup. You cannot recover subsequent transactions.

Also, in NOARCHIVELOG mode you cannot perform online tablespace backups. Furthermore, you cannot use online tablespace backups previously taken while the database operated in ARCHIVELOG mode. You can only use whole database backups taken while the database is closed to restore a database operating in NOARCHIVELOG mode. Therefore, if you decide to operate a database in NOARCHIVELOG mode, take whole database backups at regular, frequent intervals.

Running a Database in ARCHIVELOG Mode

When you run a database in ARCHIVELOG mode, you specify the archiving of the online redo log. The database control file indicates that a group of filled online redo log files cannot be used by LGWR until the group is archived. A filled group is immediately available for archiving after a redo log switch occurs.

The archiving of filled groups has these advantages:

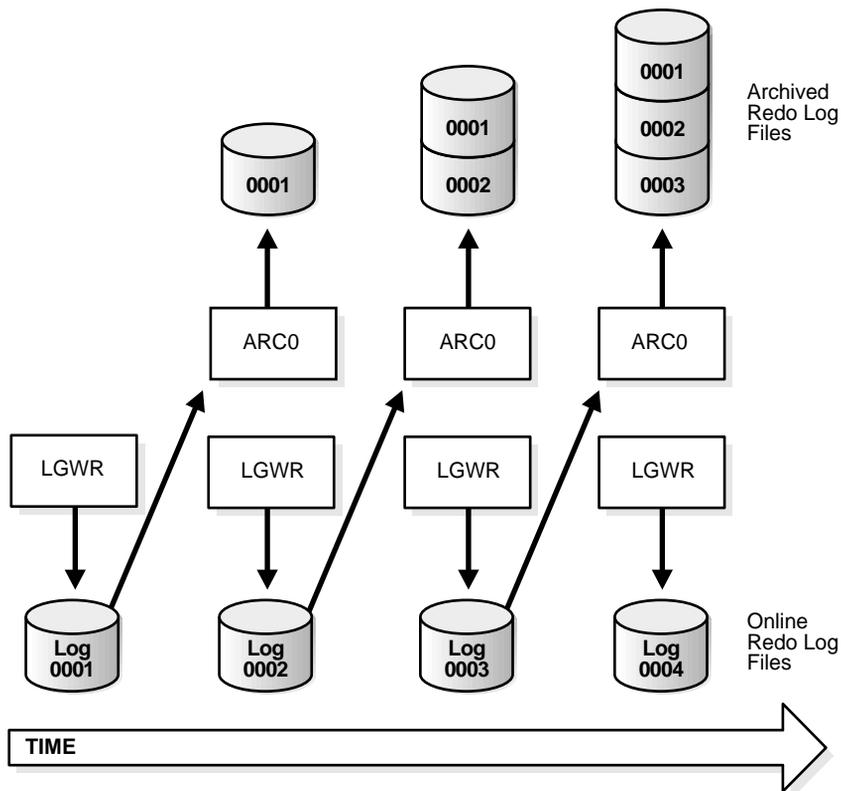
- A database backup, together with online and archived redo log files, guarantees that you can recover all committed transactions in the event of an operating system or disk failure.
- You can use a backup taken while the database is open and in normal system use if you keep an archived log.
- You can keep a standby database current with its original database by continually applying the original's archived redo logs to the standby.

Decide how you plan to archive filled groups of the online redo log. You can configure an instance to archive filled online redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually

best. [Figure 8-1](#) illustrates how the archiver process (ARC0 in this illustration) writes filled online redo log files to the database's archived redo log.

If *all* databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. If *any* database in a distributed database uses NOARCHIVELOG mode, however, recovery of a global distributed database (to make all databases consistent) is limited by the last full backup of any database operating in NOARCHIVELOG mode.

Figure 8-1 Online Redo Log File Use in ARCHIVELOG Mode



Controlling Archiving

This section describes how to control the archiving mode of the database, and how to control the archiving process. The following topics are discussed:

- [Setting the Initial Database Archiving Mode](#)
- [Changing the Database Archiving Mode](#)
- [Enabling Automatic Archiving](#)
- [Disabling Automatic Archiving](#)
- [Performing Manual Archiving](#)

See Also: Your Oracle operating system specific documentation contains additional information on controlling archiving modes.

Setting the Initial Database Archiving Mode

You set a database's initial archiving mode as part of database creation in the `CREATE DATABASE` statement. Usually, you can use the default of `NOARCHIVELOG` mode at database creation because there is no need to archive the redo information generated then. After creating the database, decide whether to change from the initial archiving mode.

Note: If a database is automatically created during Oracle installation, the initial archiving mode of the database is operating system specific.

Changing the Database Archiving Mode

To switch a database's archiving mode, use the `ALTER DATABASE` statement with the `ARCHIVELOG` or `NOARCHIVELOG` option. The following steps switch a database's archiving mode from `NOARCHIVELOG` to `ARCHIVELOG`:

1. Shut down the database instance.

```
SHUTDOWN
```

An open database must first be closed and any associated instances shut down before you can switch the database's archiving mode. You cannot disable archiving if any datafiles need media recovery.

2. Back up the database.

Before making any major change to a database, always back up the database to protect against any problems. This will be your final backup of the database in `NOARCHIVELOG` mode and can be used if something goes wrong while

trying to change to ARCHIVELOG mode. See *Oracle9i User-Managed Backup and Recovery Guide* or *Oracle9i Recovery Manager User's Guide*.

3. Edit the initialization parameter file to include initialization parameters specifying whether automatic archiving is enabled (see ["Enabling Automatic Archiving"](#) on page 8-6) and the destinations for the archive log files (see ["Specifying Archive Destinations"](#) on page 8-10).
4. Start a new instance and mount, but do not open, the database.

```
STARTUP MOUNT
```

To enable or disable archiving, the database must be mounted but not open.

5. Switch the database's archiving mode. Then open the database for normal operations.

```
ALTER DATABASE ARCHIVELOG;  
ALTER DATABASE OPEN;
```

6. Shut down the database.

```
SHUTDOWN IMMEDIATE
```

7. Back up the database.

Changing the database archiving mode updates the control file. After changing the database archiving mode, you must back up all of your database files and control file. Any previous backup is no longer usable because it was taken in NOARCHIVELOG mode.

See Also: *Oracle9i Real Application Clusters Administration* for more information about switching the archiving mode when using Oracle9i Real Application Clusters

Enabling Automatic Archiving

You can enable automatic archiving of the online redo log. When automatic archiving is enabled, no action is required to copy a group after it fills: Oracle automatically archives it. However, even when automatic archiving is enabled, you can still perform manual archiving as described in ["Performing Manual Archiving"](#) on page 8-9.

You can enable automatic archiving before or after instance startup. To enable automatic archiving after instance startup, you must be connected to Oracle with

administrator privileges (AS SYSDBA), or have the ALTER SYSTEM system privilege.

Ensure that an archived redo log destination and file name format have been specified before enabling automatic archiving. This is described in "[Specifying Archive Destinations](#)" on page 8-10.

Caution: Oracle does not automatically archive log files unless the database is also in ARCHIVELOG mode.

Enabling Automatic Archiving at Instance Startup

To enable automatic archiving of filled groups each time an instance is started, include the initialization parameter LOG_ARCHIVE_START in the database's initialization parameter file and set it to TRUE:

```
LOG_ARCHIVE_START=TRUE
```

The new value takes effect the next time you start the database.

Enabling Automatic Archiving After Instance Startup

To enable automatic archiving of filled online redo log groups without shutting down the current instance, use the ALTER SYSTEM statement specifying the ARCHIVE LOG START clause. For example:

```
ALTER SYSTEM ARCHIVE LOG START;
```

You can optionally include the archiving destination.

Note: If an instance is shut down and restarted after automatic archiving is enabled using the ALTER SYSTEM statement, the instance is reinitialized using the settings of the initialization parameter file. Those settings may or may not enable automatic archiving. If your intent is to always archive redo log files automatically, then you should include LOG_ARCHIVE_START = TRUE in your initialization parameters.

Controlling the Number of Archiver Processes

Oracle starts additional archiver processes (ARC*n*) as needed to ensure that the automatic processing of filled redo log files does not fall behind. However, to avoid

any runtime overhead of invoking additional ARC*n* processes, you can specify the number of processes to be started at instance startup using the LOG_ARCHIVE_MAX_PROCESSES initialization parameter. Up to 10 ARC*n* processes can be started.

This parameter also limits the number of ARC*n* processes that can be started for the instance. No more than the specified number of processes can ever be started.

The LOG_ARCHIVE_MAX_PROCESSES is dynamic, and can be changed using the ALTER SYSTEM statement. The following statement increases (or decreases) the number of ARC*n* processes currently running:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=3;
```

There is usually no need to change the LOG_ARCHIVE_MAX_PROCESSES initialization parameter from its default value of 2, because Oracle will adequately adjust ARC*n* processes according to system workload.

Disabling Automatic Archiving

You can disable automatic archiving of the online redo log groups at any time. After having disabled automatic archiving, you must manually archive groups of online redo log files in a timely fashion. If you run a database in ARCHIVELOG mode and disable automatic archiving, and if all groups of online redo log files are filled but not archived, then LGWR cannot reuse any inactive groups of online redo log groups. Therefore, database operation is temporarily suspended until you perform the necessary archiving.

You can disable automatic archiving at or after instance startup. To disable automatic archiving after instance startup, you must be connected with administrator privileges or have the ALTER SYSTEM privilege.

Disabling Automatic Archiving at Instance Startup

To disable the automatic archiving of filled online redo log groups at database startup, set the LOG_ARCHIVE_START initialization parameter to FALSE:

```
LOG_ARCHIVE_START=FALSE
```

Disabling Automatic Archiving after Instance Startup

To disable the automatic archiving of filled online redo log groups without shutting down the current instance, use the SQL statement ALTER SYSTEM with the ARCHIVE LOG STOP parameter. The following statement stops archiving:

```
ALTER SYSTEM ARCHIVE LOG STOP;
```

If `ARCn` is archiving a redo log group when you attempt to disable automatic archiving, `ARCn` finishes archiving the current group, but does not begin archiving the next filled online redo log group.

The instance does not have to be shut down to disable automatic archiving. If an instance is shut down and restarted after automatic archiving is disabled, however, the instance is reinitialized using the settings of the initialization parameter file, which may or may not enable automatic archiving.

Performing Manual Archiving

If you operate your database in `ARCHIVELOG` mode, but do not have automatic archiving enabled, then you must archive inactive groups of filled online redo log files or your database operation can be temporarily suspended.

You can also use manual archiving, even when automatic archiving is enabled, for such action as rearchiving an inactive group of filled online redo log members to another location. In this case, however, it is possible that the instance can reuse the redo log group before you have finished manually archiving, and thereby overwrite the files. If this happens, Oracle will write an error message to the alert file.

To archive a filled online redo log group manually, connect with administrator privileges. Use the `ALTER SYSTEM` statement with the `ARCHIVE LOG` clause to manually archive filled online redo log files. The following statement archives all unarchived log files:

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

Specifying the Archive Destination

When archiving redo logs, determine the destination to which you will archive and familiarize yourself with the various destination states. Develop a practice of using dynamic performance (V\$) views, listed in "[Viewing Information About the Archived Redo Log](#)" on page 8-23, to access archive information.

The following topics are contained in this section

- [Specifying Archive Destinations](#)
- [Understanding Archive Destination Status](#)

Specifying Archive Destinations

You must decide whether to make a **single** destination for the logs or **multiplex** them. When you multiplex them, you archive the logs to more than one location. You specify your choice by setting initialization parameters according to one of the following methods.

Method	Initialization Parameter	Host	Example
1	LOG_ARCHIVE_DEST_1 where: <i>n</i> is an integer from 1 to 10	Local or remote	LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc' LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'
2	LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST	Local only	LOG_ARCHIVE_DEST = '/disk1/arc' LOG_ARCHIVE_DUPLEX_DEST = '/disk2/arc'

See Also:

- *Oracle9i Database Reference* for additional information about the initialization parameters used to control the archiving of redo logs
- *Oracle9i Data Guard Concepts and Administration* for information about using the LOG_ARCHIVE_DEST_1 initialization parameter for specifying a standby destination. There are additional keywords that can be specified with this initialization parameter and that are not discussed in this book.

Method 1: Using the LOG_ARCHIVE_DEST_1 Parameter

The first method is to use the LOG_ARCHIVE_DEST_1 parameter (where *n* is an integer from 1 to 10) to specify from one to ten different destinations for archival. Each numerically-suffixed parameter uniquely identifies an individual destination.

You specify the location for LOG_ARCHIVE_DEST_1 using these keywords:

Keyword	Indicates	Example
LOCATION	A local file system location.	LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc'
SERVICE	Remote archival through Oracle Net service name.	LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'

If you use the `LOCATION` keyword, specify a valid path name for your operating system. If you specify `SERVICE`, Oracle translates the net service name through the `tnsnames.ora` file to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. The service name must have an associated database SID, so that Oracle correctly updates the log history of the control file for the standby database.

Perform the following steps to set the destination for archived redo logs using the `LOG_ARCHIVE_DEST_n` initialization parameter:

1. Use SQL*Plus to shut down the database.

```
SHUTDOWN
```

2. Edit the `LOG_ARCHIVE_DEST_n` parameter to specify from one to ten archiving locations. The `LOCATION` keyword specifies an operating system specific path name. For example, enter:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
LOG_ARCHIVE_DEST_3 = 'LOCATION = /disk3/archive'
```

If you are archiving to a standby database, use the `SERVICE` keyword to specify a valid net service name from the `tnsnames.ora` file. For example, enter:

```
LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
```

3. Edit the `LOG_ARCHIVE_FORMAT` initialization parameter, using `%s` to include the log sequence number as part of the file name and `%t` to include the thread number. Use capital letters (`%S` and `%T`) to pad the file name to the left with zeroes. For example, enter:

```
LOG_ARCHIVE_FORMAT = arch%s.arc
```

These settings will generate archived logs as follows for log sequence numbers 100, 101, and 102:

```
/disk1/archive/arch100.arc, /disk1/archive/arch101.arc,
/disk1/archive/arch102.arc
```

```
/disk2/archive/arch100.arc, /disk2/archive/arch101.arc,
/disk2/archive/arch102.arc
```

```
/disk3/archive/arch100.arc, /disk3/archive/arch101.arc,
/disk3/archive/arch102.arc
```

Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST

The second method, which allows you to specify a maximum of two locations, is to use the LOG_ARCHIVE_DEST parameter to specify a **primary** archive destination and the LOG_ARCHIVE_DUPLEX_DEST to specify an optional **secondary** archive destination. Whenever Oracle archives a redo log, it archives it to every destination specified by either set of parameters.

Perform the following steps to use method 2:

1. Use SQL*Plus to shut down the database.

```
SHUTDOWN
```

2. Specify destinations for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameter (you can also specify LOG_ARCHIVE_DUPLEX_DEST dynamically using the ALTER SYSTEM statement). For example, enter:

```
LOG_ARCHIVE_DEST = '/disk1/archive'  
LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
```

3. Edit the LOG_ARCHIVE_FORMAT parameter, using %s to include the log sequence number as part of the file name and %t to include the thread number. Use capital letters (%S and %T) to pad the file name to the left with zeroes. For example, enter:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc
```

For example, the above settings generates archived logs as follows for log sequence numbers 100 and 101 in thread 1:

```
/disk1/archive/arch_1_100.arc, /disk1/archive/arch_1_101.arc  
/disk2/archive/arch_1_100.arc, /disk2/archive/arch_1_101.arc
```

See Also: The following books contain more information about archiving and standby databases:

- *Oracle9i User-Managed Backup and Recovery Guide*
- *Oracle9i Recovery Manager User's Guide*
- *Oracle9i Data Guard Concepts and Administration*.

Understanding Archive Destination Status

Each archive destination has the following variable characteristics that determine its status:

- **Valid/Invalid**—indicates whether the disk location or service name information is specified and valid
- **Enabled/Disabled**—indicates the availability state of the location and whether Oracle can use the destination
- **Active/Inactive**—indicates whether there was a problem accessing the destination

Several combinations of these characteristics are possible. To obtain the current status and other information about each destination for an instance, query the `V$ARCHIVE_DEST` view.

The characteristics determining a locations status that appear in the view are shown in [Table 8-1](#). Note that for a destination to be used, its characteristics must be valid, enabled, and active.

Table 8-1 Destination Status

STATUS	Characteristics			Meaning
	Valid	Enabled	Active	
VALID	True	True	True	The user has properly initialized the destination, which is available for archiving.
INACTIVE	False	n/a	n/a	The user has not provided or has deleted the destination information.
ERROR	True	True	False	An error occurred creating or writing to the destination file; refer to error data.
FULL	True	True	False	Destination is full (no disk space).
DEFERRED	True	False	True	The user manually and temporarily disabled the destination.
DISABLED	True	False	False	The user manually and temporarily disabled the destination following an error; refer to error data.
BAD PARAM	n/a	n/a	n/a	A parameter error occurred; refer to error data. Usually this state is only seen when the <code>LOG_ARCHIVE_START</code> initialization parameter is not set.

The `LOG_ARCHIVE_DEST_STATE_n` (where *n* is an integer from 1 to 10) initialization parameter allows you to control the availability state of the specified destination (*n*). The destination state can have three values: `ENABLE`, `DEFER`, or `ALTERNATE`. The value `ENABLE` indicates that Oracle can use the destination, whereas `DEFER` indicates that the location is temporarily disabled. The third value, `ALTERNATE`, means that the destination is an alternate. Its availability state is `DEFER`, unless there is a failure of its parent destination, in which case its state becomes `ENABLE`.

Specifying the Mode of Log Transmission

There are two modes of transmitting archived logs to their destination: **normal archiving transmission** and **standby transmission** mode. Normal transmission involves transmitting files to a local disk. Standby transmission involves transmitting files through a network to either a local or remote standby database.

Normal Transmission Mode

In normal transmission mode, the archiving destination is another disk drive of the database server. In this configuration archiving does not contend with other files required by the instance and can complete more quickly. Specify the destination with either the `LOG_ARCHIVE_DEST_n` or `LOG_ARCHIVE_DEST` parameters.

Ideally, you should permanently move archived redo log files and corresponding database backups from the local disk to inexpensive offline storage media such as tape. Because a primary value of archived logs is database recovery, you want to ensure that these logs are safe should disaster strike your primary database.

Standby Transmission Mode

In standby transmission mode, the archiving destination is either a local or remote standby database.

Caution: You can maintain a standby database on a local disk, but Oracle strongly encourages you to maximize disaster protection by maintaining your standby database at a remote site.

If you are operating your standby database in **managed recovery mode**, you can keep your standby database in sync with your source database by automatically applying transmitted archive logs.

To transmit files successfully to a standby database, either ARC*n* or a server process must do the following:

- Recognize a remote location
- Transmit the archived logs in conjunction with a **remote file server (RFS)** process that resides on the remote server

Each ARC*n* process has a corresponding RFS for each standby destination. For example, if three ARC*n* processes are archiving to two standby databases, then Oracle establishes six RFS connections.

You can transmit archived logs through a network to a remote location by using Oracle Net. Indicate a remote archival by specifying a Oracle Net service name as an attribute of the destination. Oracle then translates the service name, through the `tnsnames.ora` file to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. The service name must have an associated database SID, so that Oracle correctly updates the log history of the control file for the standby database.

The RFS process, which runs on the destination node, acts as a network server to the ARC*n* client. Essentially, ARC*n* pushes information to RFS, which transmits it to the standby database.

The RFS process, which is required when archiving to a remote destination, is responsible for the following tasks:

- Consuming network I/O from the ARC*n* process
- Creating file names on the standby database by using the `STANDBY_ARCHIVE_DEST` parameter
- Populating the log files at the remote site
- Updating the standby database's control file (which Recovery Manager can then use for recovery)

Archived redo logs are integral to maintaining a standby database, which is an exact replica of a database. You can operate your database in standby archiving mode, which automatically updates a standby database with archived redo logs from the original database.

See Also:

- *Oracle9i Data Guard Concepts and Administration*
- *Oracle9i Net Services Administrator's Guide* for information about connecting to a remote database using a service name

Managing Archive Destination Failure

Sometimes archive destinations can fail, causing problems when you operate in automatic archiving mode. To minimize the problems associated with destination failure, Oracle provides you with options. Discussions of these options are contained in the following sections:

- [Specifying the Minimum Number of Successful Destinations](#)
- [Re-Archiving to a Failed Destination](#)

Specifying the Minimum Number of Successful Destinations

The optional initialization parameter `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` (where *n* is an integer from 1 to 10, or 1 to 2 if you choose to use duplexing) determines the minimum number of destinations to which Oracle must successfully archive a redo log group before it can reuse online log files. The default value is 1.

Specifying Mandatory and Optional Destinations

Using the `LOG_ARCHIVE_DEST_n` parameter, you can specify whether a destination has the attributes `OPTIONAL` (default) or `MANDATORY`. The `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` parameter uses all `MANDATORY` destinations plus some number of `OPTIONAL` non-standby destinations to determine whether LGWR can overwrite the online log.

When determining how to set your parameters, note the following:

- Not specifying `MANDATORY` for a destination is the same as specifying `OPTIONAL`.
- You must have at least one local destination, which you can declare `OPTIONAL` or `MANDATORY`.
- When using `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` at least one local destination will *operationally* be treated as `MANDATORY`, since the minimum value for `LOG_ARCHIVE_MIN_SUCCEED_DEST` is 1.
- The failure of any `MANDATORY` destination, including a `MANDATORY` standby destination, makes the `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameter irrelevant.
- The `LOG_ARCHIVE_MIN_SUCCEED_DEST` value cannot be greater than the number of destinations, nor greater than the number of `MANDATORY` destinations plus the number of `OPTIONAL` local destinations.

- If you DEFER a MANDATORY destination, and Oracle overwrites the online log without transferring the archived log to the standby site, then you must transfer the log to the standby manually.

You can also establish which destinations are mandatory or optional by using the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. Note the following rules:

- Any destination declared by LOG_ARCHIVE_DEST is mandatory.
- Any destination declared by LOG_ARCHIVE_DUPLEX_DEST is optional if LOG_ARCHIVE_MIN_SUCCEED_DEST = 1 and mandatory if LOG_ARCHIVE_MIN_SUCCEED_DEST = 2.

Sample Scenarios: Specifying the Number of Successful Destinations

You can see the relationship between the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_MIN_SUCCEED_DEST parameters most easily through sample scenarios.

Scenario 1 In this scenario, you archive to three local destinations, each of which you declare as OPTIONAL. [Table 8-2](#) illustrates the possible values for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* in this case.

Table 8-2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 1

Value	Meaning
1	Oracle can reuse log files only if at least one of the OPTIONAL destinations succeeds.
2	Oracle can reuse log files only if at least two of the OPTIONAL destinations succeed.
3	Oracle can reuse log files only if all of the OPTIONAL destinations succeed.
4 or greater	ERROR: The value is greater than the number of destinations.

This scenario shows that even though you do not explicitly set any of your destinations to MANDATORY using the LOG_ARCHIVE_DEST_*n* parameter, Oracle must successfully archive to one or more of these locations when LOG_ARCHIVE_MIN_SUCCEED_DEST is set to 1, 2, or 3.

Scenario 2 In this scenario, consider a case in which:

- You specify two MANDATORY destinations.

- You specify two `OPTIONAL` destinations.
- No destination is a standby database.

[Table 8–3](#) shows the possible values for `LOG_ARCHIVE_MIN_SUCCEED_DEST=n`.

Table 8–3 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
1	Oracle ignores the value and uses the number of <code>MANDATORY</code> destinations (in this example, 2).
2	Oracle can reuse log files even if no <code>OPTIONAL</code> destination succeeds.
3	Oracle can reuse logs only if at least one <code>OPTIONAL</code> destination succeeds.
4	Oracle can reuse logs only if both <code>OPTIONAL</code> destinations succeed.
5 or greater	ERROR: The value is greater than the number of destinations.

This case shows that Oracle must archive to the destinations you specify as `MANDATORY`, regardless of whether you set `LOG_ARCHIVE_MIN_SUCCEED_DEST` to archive to a smaller number of destinations.

Re-Archiving to a Failed Destination

Use the `REOPEN` attribute of the `LOG_ARCHIVE_DEST_n` parameter to specify whether and when `ARCn` attempts to rearchive to a failed destination following an error. `REOPEN` applies to all errors, not just `OPEN` errors.

`REOPEN=n` sets the minimum number of seconds before `ARCn` should try to reopen a failed destination. The default value for `n` is 300 seconds. A value of 0 is the same as turning off the `REOPEN` option. In other words, `ARCn` will not attempt to archive after a failure. If you do not specify the `REOPEN` keyword, `ARCn` will never reopen a destination following an error.

You cannot use `REOPEN` to specify a limit on the number of attempts to reconnect and transfer archived logs. The `REOPEN` attempt either succeeds or fails, in which case the `REOPEN` information is reset.

If you specify `REOPEN` for an `OPTIONAL` destination, Oracle can overwrite online logs if there is an error. If you specify `REOPEN` for a `MANDATORY` destination, Oracle stalls the production database when it cannot successfully archive. In this situation, consider the following options:

- Archive manually to the failed destination.

- Change the destination by deferring the destination, specifying the destination as optional, or changing the service.
- Drop the destination.

When using the `REOPEN` keyword, note the following:

- `ARCn` reopens a destination only when *starting* an archive operation from the beginning of the log file, never *during* a current operation. `ARCn` always retries the log copy from the beginning.
- If a `REOPEN` time was specified or defaulted, `ARCn` checks to see whether the time of the recorded error plus the `REOPEN` interval is less than the current time. If it is, `ARCn` retries the log copy.
- The `REOPEN` clause successfully affects the `ACTIVE=TRUE` destination state. The `VALID` and `ENABLED` states are not changed.

Tuning Archive Performance by Specifying Multiple ARCn Processes

For most databases, `ARCn` has no effect on overall system performance. On some large database sites, however, archiving can have an impact on system performance. On one hand, if `ARCn` works very quickly, overall system performance can be reduced while `ARCn` runs, since CPU cycles are being consumed in archiving. On the other hand, if `ARCn` runs extremely slowly, it has little detrimental effect on system performance, but it takes longer to archive redo log files, and can create a bottleneck if all redo log groups are unavailable because they are waiting to be archived.

You can specify up to ten `ARCn` processes for each database instance. Enable the multiple processing feature at startup or at runtime by setting the initialization parameter `LOG_ARCHIVE_MAX_PROCESSES=n` (where *n* is any integer from 1 to 10). By default, the parameter is set to 2.

Because LGWR automatically increases the number of `ARCn` processes should the current number be insufficient to handle the current workload, the parameter is intended to allow you to specify the *initial* number of `ARCn` processes or to increase or decrease the current number. Assuming the initial number of `ARCn` processes was set to 4, the following statement will decrease the number of processes to 2.

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=2;
```

When decreasing the number of `ARCn` processes, it is not determinate exactly which process will be stopped. Also, you are not allowed to alter the value of the parameter to 0, so at least one `ARCn` process is always active. Query the

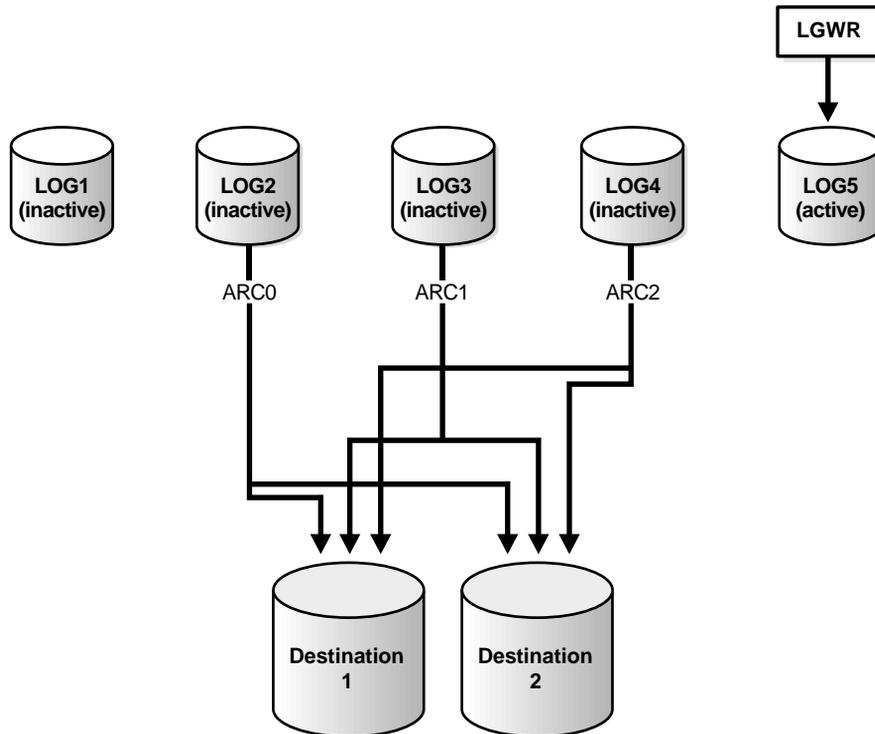
`V$ARCHIVE_PROCESSES` view to see information about the state of each archive process. Processes that have stopped show as being in the `IDLE` state.

Creating multiple processes is especially useful when you:

- Use more than two online redo logs
- Archive to more than one destination

Multiple `ARCn` processing prevents the bottleneck that occurs when `LGWR` switches through the multiple online redo logs faster than a single `ARCn` process can write inactive logs to multiple destinations. Each `ARCn` process works on only one inactive log at a time, but must archive to each specified destination.

For example, if you maintain five online redo log files, then you may decide to start the instance using three `ARCn` processes. As `LGWR` actively writes to one of the log files, the `ARCn` processes can simultaneously archive up to three of the inactive log files to various destinations. As [Figure 8-2](#) illustrates, each instance of `ARCn` assumes responsibility for a single log file and archives it to all of the defined destinations.

Figure 8–2 Using Multiple ARCn Processes

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for more information about tuning the archiving process

Controlling Trace Output Generated by the Archivelog Process

As discussed in "[Trace Files and the Alert File](#)" on page 5-15, background processes always write to a trace file when appropriate. In the case of the archivelog process, it is possible to control the output that is generated.

The `LOG_ARCHIVE_TRACE` initialization parameter can be set to specify a **trace level**. The following values can be specified:

Trace Level	Meaning
0	Disable archivelog tracing - default setting.
1	Track archival of redo log file.
2	Track archival status for each archivelog destination.
4	Track archival operational phase.
8	Track archivelog destination activity.
16	Track detailed archivelog destination activity.
32	Track archivelog destination parameter modifications.
64	Track ARC <i>n</i> process state activity
128	Track FAL (fetch archived log) server related activities
256	Supported in a future release
512	Tracks asynchronous LGWR activity
1024	RFS physical client tracking
2048	ARC <i>n</i> /RFS heartbeat tracking

You can combine tracing levels by specifying a value equal to the sum of the individual levels that you would like to trace. For example, setting `LOG_ARCHIVE_TRACE=12`, will generate trace level 8 and 4 output. You can set different values for the primary and any standby database.

The default value for the `LOG_ARCHIVE_TRACE` parameter is 0, and at this level, error conditions still generate the appropriate alert and trace entries.

You can change the value of this parameter dynamically using the `ALTER SYSTEM` statement. For example:

```
ALTER SYSTEM SET LOG_ARCHIVE_TRACE=12;
```

Changes initiated in this manner will take effect at the start of the next archiving operation.

See Also: *Oracle9i Data Guard Concepts and Administration* for information about using this parameter with a standby database

Viewing Information About the Archived Redo Log

You can display information about the archived redo logs using the following:

- [Dynamic Performance Views](#)
- [The ARCHIVE LOG LIST Command](#)

Dynamic Performance Views

There are several dynamic performance views that contain useful information about archived redo logs.

Dynamic Performance View	Description
V\$DATABASE	Identifies whether the database is in ARCHIVELOG or NOARCHIVELOG mode.
V\$ARCHIVED_LOG	Displays historical archived log information from the control file. If you use a recovery catalog, the RC_ARCHIVED_LOG view contains similar information.
V\$ARCHIVE_DEST	Describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.
V\$ARCHIVE_PROCESSES	Displays information about the state of the various archive processes for an instance.
V\$BACKUP_REDOLOG	Contains information about any backups of archived logs. If you use a recovery catalog, the RC_BACKUP_REDOLOG contains similar information.
V\$LOG	Displays all online redo log groups for the database and indicates which need to be archived.
V\$LOG_HISTORY	Contains log history information such as which logs have been archived and the SCN range for each archived log.

For example, the following query displays which online redo log group requires archiving:

```
SELECT GROUP#, ARCHIVED
```

```
FROM SYS.V$LOG;

GROUP#      ARC
-----  ---
         1  YES
         2  NO
```

To see the current archiving mode, query the V\$DATABASE view:

```
SELECT LOG_MODE FROM SYS.V$DATABASE;

LOG_MODE
-----
NOARCHIVELOG
```

See Also: *Oracle9i Database Reference* for detailed descriptions of data dictionary views

The ARCHIVE LOG LIST Command

The SQL*Plus command ARCHIVE LOG LIST can be used to show archiving information for the connected instance. For example:

```
SQL> ARCHIVE LOG LIST

Database log mode                Archive Mode
Automatic archival                Enabled
Archive destination              D:\ORANT\oradata\IDDB2\archive
Oldest online log sequence       11160
Next log sequence to archive     11163
Current log sequence             11163
```

This display tells you all the necessary information regarding the archived redo log settings for the current instance:

- The database is currently operating in ARCHIVELOG mode.
- Automatic archiving is enabled.
- The archived redo log's destination is D:\ORANT\oradata\IDDB2\archive.
- The oldest filled online redo log group has a sequence number of 11160.
- The next filled online redo log group to archive has a sequence number of 11163.
- The current online redo log file has a sequence number of 11163.

See Also: *SQL*Plus User's Guide and Reference* for more information on the `ARCHIVE LOG LIST` command

Using LogMiner to Analyze Redo Logs

The Oracle LogMiner utility enables you to query redo logs through a SQL interface. Redo logs contain information about the history of activity on a database.

This chapter contains the following sections:

- [Potential Uses for Data Stored in Redo Logs](#)
- [Accessing Information Stored in Redo Logs](#)
- [Redo Logs and Dictionary Files](#)
- [LogMiner Recommendations and Restrictions](#)
- [Filtering Data That is Returned](#)
- [Accessing LogMiner Information](#)
- [Querying VSLOGMNR_CONTENTS](#)
- [Extracting Actual Data Values from Redo Logs](#)
- [Supplemental Logging](#)
- [Steps in a Typical LogMiner Session](#)
- [Example Uses of LogMiner](#)

This chapter describes LogMiner functionality as it is used from the command line. You also have the option of accessing LogMiner functionality through the Oracle LogMiner Viewer graphical user interface (GUI). The LogMiner Viewer is a part of Oracle Enterprise Manager.

Potential Uses for Data Stored in Redo Logs

All changes made to user data or to the data dictionary are recorded in the Oracle redo logs. Therefore, redo logs contain all the necessary information to perform recovery operations. Because redo log data is often kept in archived files, the data is already available. To ensure that redo logs contain useful information, you should enable at least minimal supplemental logging.

See Also: [Supplemental Logging](#) on page 9-19

The following are some of the potential uses for data contained in redo logs:

- Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun. An example of an error made at the application level could be if a user mistakenly updated a database to give all employees 100 percent salary increases rather than 10 percent increases. It is important to know exactly when corruption began so that you know when to initiate time-based or change-based recovery. This enables you to restore the database to the state it was in just before corruption.

See Also: See [Extracting Actual Data Values from Redo Logs](#) on page 9-18 for details about how you can use LogMiner to accomplish this.

- Detecting and whenever possible, correcting user error, which is a more likely scenario than logical corruption. User errors include deleting the wrong rows because of incorrect values in a `WHERE` clause, updating rows with incorrect values, dropping the wrong index, and so forth.
- Determining what actions you would have to take to perform fine-grained recovery at the transaction level. If you fully understand and take into account existing dependencies, it may be possible to perform a table-based undo operation to roll back a set of changes. Normally you would have to restore the table to its previous state, and then apply an archived redo log to roll it forward.
- Performance tuning and capacity planning through trend analysis. You can determine which tables get the most updates and inserts. That information provides a historical perspective on disk access statistics, which can be used for tuning purposes.

- Performing post-auditing. The redo logs contain all the information necessary to track any DML and DDL statements executed on the database, the order in which they were executed, and who executed them.

Accessing Information Stored in Redo Logs

Oracle Corporation provides SQL access to the redo logs through LogMiner, which is part of the Oracle database server. LogMiner presents the information in the redo logs through the `V$LOGMNR_CONTENTS` fixed view. This view contains historical information about changes made to the database including, but not limited to, the following:

- The type of change made to the database (`INSERT`, `UPDATE`, `DELETE`, or `DDL`).
- The SCN at which a change was made (`SCN` column).
- The SCN at which a change was committed (`COMMIT_SCN` column).
- The transaction to which a change belongs (`XIDUSN`, `XIDSLT`, and `XIDSQN` columns).
- The table and schema name of the modified object (`SEG_NAME` and `SEG_OWNER` columns).
- The name of the user who issued the DDL or DML statement to make the change (`USERNAME` column).
- Reconstructed SQL statements showing SQL that is equivalent (but not necessarily identical) to the SQL used to generate the redo records (`SQL_REDO` column). If a password is part of the statement in a `SQL_REDO` column, the password is encrypted.
- Reconstructed SQL statements showing the SQL statements needed to undo the change (`SQL_UNDO` column). `SQL_UNDO` columns that correspond to DDL statements are always `NULL`. Similarly, the `SQL_UNDO` column may be `NULL` for some datatypes and for rolled back operations.

The redo logs contain internally generated numerical identifiers to identify tables and their associated columns. To reconstruct SQL statements, LogMiner needs to know how the internal identifiers map to user-defined names. This mapping information is stored in the data dictionary for the database. LogMiner provides a procedure (`DBMS_LOGMNR_D.BUILD`) that lets you extract the data dictionary.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for a complete description of the `DBMS_LOGMNR_D.BUILD` procedure

The following section describes redo logs and dictionary files in further detail.

Redo Logs and Dictionary Files

Before you begin using LogMiner, it is important to understand how LogMiner works with redo logs and dictionary files. This will help you to get accurate results and to plan the use of your system resources. The following concepts are discussed in this section:

- [Redo Logs](#)
- [Dictionary Options](#)
- [Tracking DDL Statements](#)

Redo Logs

When you run LogMiner, you specify the names of redo logs that you want to analyze. LogMiner retrieves information from those redo logs and returns it through the `V$LOGMNR_CONTENTS` view. To ensure that the redo logs contain information of value to you, you must enable at least minimal supplemental logging. See [Supplemental Logging](#) on page 9-19.

You can then use SQL to query the `V$LOGMNR_CONTENTS` view, as you would any other view. Each select operation that you perform against the `V$LOGMNR_CONTENTS` view causes the redo logs to be read sequentially.

Keep the following things in mind about redo logs:

- The redo logs must be from a release 8.0 or later Oracle database. However, several of the LogMiner features introduced as of release 9.0.1 only work with redo logs produced on an Oracle9i or later database. See [Restrictions](#) on page 9-11.
- Support for `LOB` and `LONG` datatypes is available as of release 9.2, but only for redo logs generated on a release 9.2 Oracle database.
- The redo logs must use a database character set that is compatible with the character set of the database on which LogMiner is running.
- In general, the analysis of redo logs requires a dictionary that was generated from the same database that generated the redo logs.
- If you are using a dictionary that is in flat file format or that is stored in the redo logs, then the redo logs you want to analyze can be from either the database on which LogMiner is running or from other databases.

- If you are using the online catalog as the LogMiner dictionary, you can only analyze redo logs from the database on which LogMiner is running.
- LogMiner must be running on the same hardware platform that generated the redo logs being analyzed. However, it does not have to be on the same system.
- It is important to specify the correct redo logs when running LogMiner. If you omit redo logs that contain some of the data you need, you will get inaccurate results when you query the `V$LOGMNR_CONTENTS` view.

To determine which redo logs are being analyzed in the current LogMiner session you can look at the `V$LOGMNR_LOGS` view, which contains one row for each redo log.

See Also: ["Specify Redo Logs for Analysis"](#) on page 9-24

Dictionary Options

To fully translate the contents of redo logs, LogMiner requires access to a database dictionary.

LogMiner uses the dictionary to translate internal object identifiers and datatypes to object names and external data formats. Without a dictionary, LogMiner returns internal object IDs and presents data as hex bytes.

For example, instead of the SQL statement:

```
INSERT INTO emp(name, salary) VALUES ('John Doe', 50000);
```

LogMiner will display:

```
insert into Object#2581(col#1, col#2) values (hexrow('4a6f686e20446f65'),
hexrow('c306'));"
```

A LogMiner dictionary file contains information that identifies the database it was created from and the time it was created. This information is used to validate the dictionary against the selected redo logs, automatically detecting any mismatch between LogMiner's internal dictionary and the redo logs.

The dictionary file must have the same database character set and be created from the same database as the redo logs being analyzed. However, once the dictionary is extracted, you can use it to mine the redo logs of that database in a separate database instance without being connected to the source database.

Extracting a dictionary file also prevents problems that can occur when the current data dictionary contains only the newest table definitions. For instance, if a table

you are searching for was dropped sometime in the past, the current dictionary will not contain any references to it.

LogMiner gives you three choices for your source dictionary:

- [Extracting the Dictionary to a Flat File](#)
- [Extracting a Dictionary to the Redo Logs](#)
- [Using the Online Catalog](#)

Extracting the Dictionary to a Flat File

When the dictionary is in a flat file, fewer system resources are used than when it is contained in the redo logs. It is recommended that you regularly back up the dictionary extracts to ensure correct analysis of older redo logs.

To extract database dictionary information to a flat file, use the `DBMS_LOGMNR_D.BUILD` procedure with the `STORE_IN_FLAT_FILE` option.

Be sure that no DDL operations occur while the dictionary is being built.

The following steps describe how to extract a dictionary to a flat file (including extra steps you must take if you are using Oracle8). Steps 1 through 4 are preparation steps. You only need to do them once, and then you can extract a dictionary to a flat file as many times as you wish.

1. The `DBMS_LOGMNR_D.BUILD` procedure requires access to a directory where it can place the dictionary file. Because PL/SQL procedures do not normally access user directories, you must specify a directory for use by the `DBMS_LOGMNR_D.BUILD` procedure or the procedure will fail. To specify a directory, set the initialization parameter, `UTL_FILE_DIR`, in the `init.ora` file.

See Also: *Oracle9i Database Reference* for more information about the `init.ora` file

For example, to set `UTL_FILE_DIR` to use `/oracle/database` as the directory where the dictionary file is placed, enter the following in the `init.ora` file:

```
UTL_FILE_DIR = /oracle/database
```

Remember that for the changes to the `init.ora` file to take effect, you must stop and restart the database.

2. **For Oracle8 only. Otherwise, go to the next step:** Use your operating system's copy command to copy the `dbmslmd.sql` script, which is contained in the

`$ORACLE_HOME/rdbms/admin` directory on the Oracle8i database, to the same directory in the Oracle8 database. For example, enter:

```
% cp /8.1/oracle/rdbms/admin/dbmslmd.sql /8.0/oracle/rdbms/admin/dbmslmd.sql
```

3. If the database is closed, use SQL*Plus to mount and then open the database whose redo logs you want to analyze. For example, entering the `STARTUP` command mounts and opens the database:

```
SQL> STARTUP
```

4. **For Oracle8 only. Otherwise, go to the next step:** Execute the copied `dbmslmd.sql` script on the 8.0 database to install the `DBMS_LOGMNR_D` package. For example, enter:

```
@dbmslmd.sql
```

You may need to enter the complete path to the script.

5. Execute the PL/SQL procedure `DBMS_LOGMNR_D.BUILD`. Specify a filename for the dictionary and a directory path name for the file. This procedure creates the dictionary files. For example, enter the following to create the file `dictionary.ora` in `/oracle/database`:

```
SQL> EXECUTE DBMS_LOGMNR_D.BUILD('dictionary.ora', -
  2 '/oracle/database/', -
  3 OPTIONS => DBMS_LOGMNR_D.STORE_IN_FLAT_FILE);
```

You could also specify a filename and location without specifying the `STORE_IN_FLAT_FILE` option. The result would be the same.

Extracting a Dictionary to the Redo Logs

To extract a dictionary to the redo logs, the database must be open and in `ARCHIVELOG` mode and archiving must be enabled. While the dictionary is being extracted to the redo log stream, no DDL statements can be executed. Therefore, the dictionary snapshot extracted to the redo logs is guaranteed to be consistent, whereas the dictionary extracted to a flat file is not.

To extract database dictionary information to the redo logs, use the `DBMS_LOGMNR_D.BUILD` procedure with the `STORE_IN_REDO_FILES` option. Do not specify a filename or location.

```
SQL> EXECUTE DBMS_LOGMNR_D.BUILD ( -
  2 OPTIONS=>DBMS_LOGMNR_D.STORE_IN_REDO_LOGS);
```

To ensure that the redo logs contain information of value to you, you must enable at least minimal supplemental logging. See [Supplemental Logging](#) on page 9-19.

See Also: *Oracle9i Recovery Manager User's Guide* for more information about ARCHIVELOG mode

The process of extracting the dictionary to the redo logs does consume database resources, but if you limit the extraction to off-peak hours, this should not be a problem and it is faster than extracting to a flat file. Depending on the size of the dictionary, it may be contained in multiple redo logs. Provided the relevant redo logs have been archived, you can find out which redo logs contain the start and end of an extracted dictionary. To do so, query the V\$ARCHIVED_LOG view, as follows:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_BEGIN='YES' ;
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_END='YES' ;
```

The names of the start and end redo logs, and possibly other logs in between them, are specified with the ADD_LOGFILE procedure when you are preparing to start a LogMiner session.

It is recommended that you periodically back up the redo logs so that the information is saved and available at a later date. Ideally, this will not involve any extra steps because if your database is being properly managed, there should already be a process in place for backing up and restoring archived redo logs. Again, because of the time required, it is good practice to do this during off-peak hours.

Using the Online Catalog

To direct LogMiner to use the dictionary currently in use for the database, specify the online catalog as your dictionary source when you start LogMiner, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
      2 DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG);
```

Using the online catalog means that you do not have to bother extracting a dictionary to a flat file or to the redo logs. In addition to using the online catalog to analyze online redo logs, you can use it to analyze archived redo logs provided you are on the same system that generated the archived redo logs.

The online catalog contains the latest information about the database and may be the fastest way to start your analysis. Because DDL operations that change

important tables are somewhat rare, the online catalog generally contains the information you need for your analysis.

Remember, however, that the online catalog can only reconstruct SQL statements that are executed on the latest version of a table. As soon as the table is altered, the online catalog no longer reflects the previous version of the table. This means that LogMiner will not be able to reconstruct any SQL statements that were executed on the previous version of the table. Instead, LogMiner generates nonexecutable SQL in the `SQL_REDO` column (including hex-to-raw formatting of binary values) similar to the following example:

```
insert into Object#2581(col#1, col#2) values (hextoraw('4a6f686e20446f65'),
hextoraw('c306'));"
```

The online catalog option requires that the database be open.

The online catalog option is not valid with the `DDL_DICT_TRACKING` option.

Tracking DDL Statements

LogMiner automatically builds its own internal dictionary from the source dictionary that you specify when you start LogMiner (either a flat file dictionary, a dictionary in the redo logs, or an online catalog).

If your source dictionary is a flat file dictionary or a dictionary in the redo logs, you can use the `DDL_DICT_TRACKING` option to direct LogMiner to track data definition language (DDL) statements. DDL tracking is disabled by default. To enable it, use the `OPTIONS` parameter to specify `DDL_DICT_TRACKING` when you start LogMiner. For example:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
2 DBMS_LOGMNR.DDL_DICT_TRACKING);
```

With this option set, LogMiner applies any DDL statements seen in the redo logs to its internal dictionary. For example, to see all the DDLs executed by user `SYS`, you could issue the following query:

```
SQL> SELECT USERNAME, SQL_REDO
2 FROM V$LOGMNR_CONTENTS
3 WHERE USERNAME = 'SYS' AND OPERATION = 'DDL';
```

The information returned might be similar to the following, although the actual information and how it is displayed will be different on your screen.

```
USERNAME    SQL_REDO
SYS         ALTER TABLE SCOTT.ADDRESS ADD CODE NUMBER;
```

```
SYS          CREATE USER KATHY IDENTIFIED BY VALUES 'E4C8B920449B4C32' DEFAULT
            TABLESPACE TSL;
```

Keep the following in mind when you use the `DDL_DICT_TRACKING` option:

- The `DDL_DICT_TRACKING` option is not valid with the `DICTIONARY_FROM_ONLINE_CATALOG` option.
- The `DDL_DICT_TRACKING` option requires that the database be open.

The ability to track DDL statements helps you monitor schema evolution because SQL statements used to change the logical structure of a table (because of DDL operations such as adding or dropping of columns) can be reconstructed. In addition, data manipulation language (DML) operations performed on new tables created after the dictionary was extracted can also be shown.

Note: In general, it is a good idea to keep the DDL tracking feature enabled because if it is *not* enabled and a DDL event occurs, LogMiner returns some of the redo data as hex bytes. Also, a metadata version mismatch could occur.

Because LogMiner automatically assigns versions to the database metadata, it will detect and notify you of any mismatch between its internal dictionary and the redo logs.

Note: It is important to understand that the LogMiner internal dictionary is not the same as the LogMiner dictionary contained in a flat file or in redo logs. LogMiner does update its internal dictionary, but it does *not* update the dictionary that is contained in a flat file or in redo logs.

LogMiner Recommendations and Restrictions

When you are using LogMiner, keep the recommendations and restrictions described in the following sections in mind.

Recommendations

Oracle Corporation recommends that you take the following into consideration when you are using LogMiner:

- All databases should employ an alternate tablespace for LogMiner tables. By default all LogMiner tables are created to use the `SYSTEM` tablespace. Use the `DBMS_LOGMNR_D.SET_TABLESPACE` routine to re-create all LogMiner tables in an alternate tablespace. For example, the following statement will re-create all LogMiner tables to use the `logmnrts$` tablespace:

```
SQL> EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts');
```

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for a full description of the `DBMS_LOGMNR_D.SET_TABLESPACE` routine

Restrictions

The following restrictions apply when you are using LogMiner:

- The following are not supported:
 - Simple and nested abstract datatypes (ADTs)
 - Collections (nested tables and VARRAYS)
 - Object Refs
 - Index organized tables (IOTs)
 - `CREATE TABLE AS SELECT` of a table with a clustered key
- LogMiner runs only on databases of release 8.1 or higher, but you can use it to analyze redo logs from release 8.0 databases. However, the information that LogMiner is able to retrieve from a redo log depends on the version of the log, not the version of the database in use. For example, redo logs for Oracle9i can be augmented to capture additional information when supplemental logging is enabled. This allows LogMiner functionality to be used to its fullest advantage. Redo logs created with older releases of Oracle will not have that additional data and may therefore have limitations on the operations and datatypes supported by LogMiner.

For example, the following features *require that supplemental logging be turned on*. (Note that in Oracle9i release 9.0.1, supplemental logging was always on (it was not available at all in releases prior to 9.0.1). But in release 9.2, you must specifically turn on supplemental logging; otherwise it will not be enabled.)

- Support for index clusters, chained rows, and migrated rows (for chained rows, supplemental logging is required, regardless of the compatibility level to which the database is set).

- Support for direct-path inserts (also requires that ARCHIVELOG mode be enabled).
- Extracting the data dictionary into the redo logs.
- DDL tracking.
- Generating SQL_REDO and SQL_UNDO with primary key information for updates.
- LONG and LOB datatypes are supported only if supplemental logging is enabled.

See Also: [Supplemental Logging](#) on page 9-19

Filtering Data That is Returned

LogMiner can potentially be dealing with large amounts of information. There are several methods you can use to limit the information that is returned to the V\$LOGMNR_CONTENTS view, as well as the speed at which it is returned. These options are specified when you start LogMiner.

- [Showing Only Committed Transactions](#)
- [Skipping Redo Corruptions](#)
- [Filtering Data By Time](#)
- [Filtering Data By SCN](#)

Showing Only Committed Transactions

When you use the COMMITTED_DATA_ONLY option, only rows belonging to committed transactions are shown in the V\$LOGMNR_CONTENTS view. This enables you to filter out rolled back transactions, transactions that are in progress, and internal operations.

To enable this option, you specify it when you start LogMiner, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -  
    2 DBMS_LOGMNR.COMMITTED_DATA_ONLY);
```

When you specify the COMMITTED_DATA_ONLY option, LogMiner groups together all DML operations that belong to the same transaction. Transactions are returned in the order in which they were committed.

If long-running transactions are present in the redo logs being analyzed, use of this option may cause an "Out of Memory" error.

The default is for LogMiner to show rows corresponding to all transactions and to return them in the order in which they are encountered in the redo logs.

For example, suppose you start LogMiner without specifying `COMMITTED_DATA_ONLY` and you execute the following query:

```
SQL> SELECT (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,
           2 USERNAME AS USER,
           3 SQL_REDO AS SQL_REDO
           4 FROM V$LOGMNR_CONTENTS;
```

The output would be as follows. Both committed and uncommitted transactions are returned and rows from different transactions are interwoven.

XID	USER	SQL_REDO
1.5.123	SCOTT	SET TRANSACTION READ WRITE;
1.5.123	SCOTT	INSERT INTO "SCOTT"."EMP" ("EMPNO", "ENAME") VALUES (8782, 'Frost');
1.6.124	KATHY	SET TRANSACTION READ WRITE;
1.6.124	KATHY	INSERT INTO "SCOTT"."CUSTOMER" ("ID", "NAME", "PHONE_DAY") VALUES (8839, 'Cummings', '415-321-1234');
1.6.124	KATHY	INSERT INTO "SCOTT"."CUSTOMER" ("ID", "NAME", "PHONE_DAY") VALUES (7934, 'Yeats', '033-334-1234');
1.5.123	SCOTT	INSERT INTO "SCOTT"."EMP" ("EMPNO", "ENAME") VALUES (8566, 'Browning');
1.6.124	KATHY	COMMIT;
1.7.234	GOUTAM	SET TRANSACTION READ WRITE;
1.5.123	SCOTT	COMMIT;
1.7.234	GOUTAM	INSERT INTO "SCOTT"."CUSTOMER" ("ID", "NAME", "PHONE_DAY") VALUES (8499, 'Emerson', '202-334-1234');

Now suppose you start LogMiner, but this time you specify the `COMMITTED_DATA_ONLY` option. If you executed the previous query again, the output would look as follows:

1.6.124	KATHY	SET TRANSACTION READ WRITE;
1.6.124	KATHY	INSERT INTO "SCOTT"."CUSTOMER" ("ID", "NAME", "PHONE_DAY") VALUES (8839, 'Cummings', '415-321-1234');
1.6.124	KATHY	INSERT INTO "SCOTT"."CUSTOMER" ("ID", "NAME", "PHONE_DAY") VALUES (7934, 'Yeats', '033-334-1234');
1.6.124	KATHY	COMMIT;
1.5.123	SCOTT	SET TRANSACTION READ WRITE;
1.5.123	SCOTT	INSERT INTO "SCOTT"."EMP" ("EMPNO", "ENAME")

```
VALUES (8566, 'Browning');
1.5.123 SCOTT INSERT INTO "SCOTT"."EMP" ("EMPNO", "ENAME")
VALUES (8782, 'Frost');
1.5.123 SCOTT COMMIT;
```

Because the commit for the 1.6.124 transaction happened before the commit for the 1.5.123 transaction, the entire 1.6.124 transaction is returned first. This is true even though the 1.5.123 transaction started before the 1.6.124 transaction. None of the 1.7.234 transaction is returned because a commit was never issued for it.

Skipping Redo Corruptions

When you use the `SKIP_CORRUPTION` option, any corruptions in the redo logs are skipped during select operations from the `V$LOGMNR_CONTENTS` view. Rows that are retrieved after the corruption are flagged with a "Log File Corruption Encountered" message. Additionally, for every corrupt redo record encountered, an informational row is returned that indicates how many blocks were skipped.

The default is for the select operation to terminate at the first corruption it encounters in the redo log.

To enable this option, you specify it when you start LogMiner, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
2 DBMS_LOGMNR.SKIP_CORRUPTION);
```

Filtering Data By Time

To filter data by time, set the `STARTTIME` and `ENDTIME` parameters. The procedure expects date values. Use the `TO_DATE` function to specify date and time, as in this example:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -
2 DICTFILENAME => '/oracle/dictionary.ora', -
3 STARTTIME => TO_DATE('01-Jan-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS'), -
4 ENDTIME => TO_DATE('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

If no `STARTTIME` or `ENDTIME` parameters are specified, the entire redo log is read from start to end, for each `SELECT` statement issued.

The timestamps should not be used to infer ordering of redo records. You can infer the order of redo records by using the SCN.

Filtering Data By SCN

To filter data by SCN (system change number), use the `STARTSCN` and `ENDSCN` parameters, as in this example:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -  
  2 DICTFILENAME => '/oracle/dictionary.ora', -  
  3 STARTSCN => 100, -  
  4 ENDSCN => 150);
```

The `STARTSCN` and `ENDSCN` parameters override the `STARTTIME` and `ENDTIME` parameters in situations where all are specified.

If no `STARTSCN` or `ENDSCN` parameters are specified, the entire redo log is read from start to end, for each `SELECT` statement issued.

Accessing LogMiner Information

LogMiner information is contained in the following views. You can use SQL to query them as you would any other view.

- `V$LOGMNR_CONTENTS`
Shows changes made to user and table information.
- `V$LOGMNR_DICTIONARY`
Shows information about the LogMiner dictionary file, provided the dictionary was created using the `STORE_IN_FLAT_FILE` option. The information shown includes the database name and status information.
- `V$LOGMNR_LOGS`
Shows information about specified redo logs. There is one row for each redo log.
- `V$LOGMNR_PARAMETERS`
Shows information about optional LogMiner parameters, including starting and ending system change numbers (SCNs) and starting and ending times.

See Also: *Oracle9i Database Reference* for detailed information about the contents of these views

The rest of this section discusses the following topics with regard to accessing LogMiner information:

- [Querying V\\$LOGMNR_CONTENTS](#)
- [Extracting Actual Data Values from Redo Logs](#)

Querying V\$LOGMNR_CONTENTS

LogMiner output is contained in the V\$LOGMNR_CONTENTS view. After LogMiner is started, you can issue SQL statements at the command line to query the data contained in V\$LOGMNR_CONTENTS.

When a SQL select operation is executed against the V\$LOGMNR_CONTENTS view, the redo logs are read sequentially. Translated information from the redo logs is returned as rows in the V\$LOGMNR_CONTENTS view. This continues until either the filter criteria specified at startup are met or the end of the redo log is reached.

LogMiner returns all the rows in SCN order unless you have used the COMMITTED_DATA_ONLY option to specify that only committed transactions should be retrieved. SCN order is the order normally applied in media recovery.

For example, suppose you wanted to find out about any delete operations that a user named Ron had performed on the scott.orders table. You could issue a query similar to the following:

```
SQL> SELECT OPERATION, SQL_REDO, SQL_UNDO
       2 FROM V$LOGMNR_CONTENTS
       3 WHERE SEG_OWNER = 'SCOTT' AND SEG_NAME = 'ORDERS' AND
       4 OPERATION = 'DELETE' AND USERNAME = 'RON';
```

The following output would be produced. The formatting may be different on your display than that shown here.

OPERATION	SQL_REDO	SQL_UNDO
DELETE	delete from "SCOTT"."ORDERS" where "ORDER_NO" = 2 and "QTY" = 3 and "EXPR_SHIP" = 'Y' and ROWID = 'AAABM8AABAAALm/AAA'	insert into "SCOTT"."ORDERS" ("ORDER_NO", "QTY", "EXPR_SHIP") values(2,3,'Y');
DELETE	delete from "SCOTT"."ORDERS" where "ORDER_NO" = 4 and "QTY" = 7 and "EXPR_SHIP" = 'Y' and ROWID = 'AAABM8AABAAALm/AAC';	insert into "SCOTT"."ORDERS" ("ORDER_NO", 'QTY', "EXPR_SHIP") values(4,7,'Y');

This output shows that user Ron deleted two rows from the `scott.orders` table. The reconstructed SQL statements are equivalent, but not necessarily identical, to the actual statement that Ron issued. The reason for this is that the original `WHERE` clause is not logged in the redo logs, so LogMiner can only show deleted (or updated or inserted) rows individually.

Therefore, even though a single `DELETE` statement may have been responsible for the deletion of both rows, the output in `V$LOGMNR_CONTENTS` does not reflect that. Thus, the actual `DELETE` statement may have been `DELETE FROM SCOTT.ORDERS WHERE EXPR_SHIP = 'Y'` or it might have been `DELETE FROM SCOTT.ORDERS WHERE QTY < 8`.

Executing Reconstructed SQL Statements

By default, `SQL_REDO` and `SQL_UNDO` statements are ended with a semicolon. Depending on how you plan to use the reconstructed statements, you may or may not want them to include the semicolon. To suppress the semicolon, specify the `DBMS_LOGMNR.NO_SQL_DELIMITER` option when you start LogMiner.

Note that if the `STATUS` field of `V$LOGMNR_CONTENTS` contains `dbms_logmnr.invalid_sql`, then the SQL cannot be executed.

Formatting of Returned Data

Sometimes a query can result in a large number of columns containing reconstructed SQL statements, which can be visually busy and hard to read. LogMiner provides the `DBMS_LOGMNR.PRINT_PRETTY_SQL` option to address this problem. The `PRINT_PRETTY_SQL` option formats the reconstructed SQL statements as follows, which makes them easier to read:

```
insert into "SCOTT"."EMP" values
  "EMPNO": 5505,
  "ENAME": "Parker",
  "SAL": 9000
  "DEPTNO": NULL;
update "SCOTT"."EMP"
set
  "EMPNO" = 5505 and
  "SAL" = 9000
where
  "EMPNO" = 5505 and
  "SAL" = 9000 and
  "ROWID" = AABBCXFGHA;
```

SQL statements that are reconstructed when the `PRINT_PRETTY_SQL` option is enabled are not executable because they do not use standard SQL syntax.

Extracting Actual Data Values from Redo Logs

LogMiner lets you make queries based on actual data values. For instance, you could perform a query to show all updates to `scott.emp` that increased `sal` more than a certain amount. Data such as this can be used to analyze system behavior and to perform auditing tasks.

LogMiner data extraction from redo logs is performed using two mine functions: `DBMS_LOGMNR.MINE_VALUE` and `DBMS_LOGMNR.COLUMN_PRESENT`. These functions are part of the `DBMS_LOGMNR` package. Support for these mine functions is provided by the `REDO_VALUE` and `UNDO_VALUE` columns in the `V$LOGMNR_CONTENTS` view.

The following is an example of how you could use the `MINE_VALUE` function to select all updates to `scott.emp` that increased the `sal` column to more than twice its original value:

```
SQL> SELECT SQL_REDO FROM V$LOGMNR_CONTENTS
       2 WHERE
       3 SEG_NAME = 'emp' AND
       4 SEG_OWNER = 'SCOTT' AND
       5 OPERATION = 'UPDATE' AND
       6 DBMS_LOGMNR.MINE_VALUE(REDO_VALUE, 'SCOTT.EMP.SAL') >
       7 2*DBMS_LOGMNR.MINE_VALUE(UNDO_VALUE, 'SCOTT.EMP.SAL');
```

As shown in this example, the `MINE_VALUE` function takes two arguments. The first one specifies whether to mine the redo (`REDO_VALUE`) or undo (`UNDO_VALUE`) portion of the data. The second argument is a string that specifies the fully-qualified name of the column to be mined (in this case, `SCOTT.EMP.SAL`). The `MINE_VALUE` function always returns a string that can be converted back to the original datatype.

NULL Returns From the MINE_VALUE Function

If the `MINE_VALUE` function returns a `NULL` value, it can mean either:

- The specified column is not present in the redo or undo portion of the data.
- The specified column is present and has a null value.

To distinguish between these two cases, use the `DBMS_LOGMNR.COLUMN_PRESENT` function which returns a 1 if the column is present in the redo or undo portion of the data. Otherwise, it returns a 0. For example, suppose you wanted to find out the

increment by which the values in the `sal` column were modified and the corresponding transaction identifier. You could issue the following query:

```
SQL> SELECT
  2 (XIDUSN || '.' || XIDSLT || '.' || XIDSQN) AS XID,
  3 (DBMS_LOGMNR.MINE_VALUE(REDO_VALUE, 'SCOTT.EMP.SAL') -
  4 DBMS_LOGMNR.MINE_VALUE(UNDO_VALUE, 'SCOTT.EMP.SAL')) AS INCR_SAL
  5 FROM V$LOGMNR_CONTENTS
  6 WHERE
  7 DBMS_LOGMNR.COLUMN_PRESENT(REDO_VALUE, 'SCOTT.EMP.SAL') = 1 AND
  8 DBMS_LOGMNR.COLUMN_PRESENT(UNDO_VALUE, 'SCOTT.EMP.SAL') = 1 AND
  9 OPERATION = 'UPDATE';
```

Usage Rules for the MINE_VALUE and COLUMN_PRESENT Functions

The following usage rules apply to the `MINE_VALUE` and `COLUMN_PRESENT` functions:

- They can only be used within a LogMiner session.
- They must be invoked in the context of a select operation from the `V$LOGMNR_CONTENTS` view.
- They do not support `LONG`, `LOB`, `ADT`, or `COLLECTION` datatypes.
- When the column argument is of type `DATE`, the string that is returned is formatted in canonical form (`DD-MON-YYYY HH24:MI:SS.SS`) regardless of the date format of the current session.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of the `DBMS_LOGMNR` package, which contains the `MINE_VALUE` and `COLUMN_PRESENT` functions

Supplemental Logging

Redo logs are generally used for instance recovery and media recovery. The data needed for such operations is automatically recorded in the redo logs. However, a redo-based application may require that additional information be logged in the redo logs. The following are examples of situations in which supplemental data may be needed:

- An application that wanted to apply the reconstructed SQL statements to a different database would need to identify the update statement by its primary

key, not by its ROWID which is the usual method used by LogMiner. (Primary keys are not, by default, logged in the redo logs unless the key itself is changed by the update.)

- To make tracking of row changes more efficient, an application may require that the before image of the whole row be logged, not just the modified columns.

The default behavior of the Oracle database server is to not provide any supplemental logging at all, which means that certain features will not be supported (see [Restrictions](#) on page 9-11). *If you want to make full use of LogMiner support, you must enable supplemental logging.*

The use of LogMiner with minimal supplemental logging enabled does not have any significant performance impact on the instance generating the redo logs. However, the use of LogMiner with database-wide supplemental logging enabled does impose significant overhead and effects performance.

There are two types of supplemental logging: database supplemental logging and table supplemental logging. Each of these is described in the following sections.

Database Supplemental Logging

There are two types of database supplemental logging: minimal and identification key logging.

Minimal supplemental logging logs the minimal amount of information needed for LogMiner to identify, group, and merge the REDO operations associated with DML changes. It ensures that LogMiner (and any products building on LogMiner technology) have sufficient information to support chained rows and various storage arrangements such as cluster tables. In most situations, you should at least enable minimal supplemental logging. To do so, execute the following statement:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
```

Note: In LogMiner release 9.0.1, minimal supplemental logging was the default behavior. In release 9.2, the default is *no* supplemental logging. It must be specifically enabled.

Identification key logging enables database-wide before-image logging of primary keys or unique indexes (in the absence of primary keys) for all updates. With this type of logging, an application can identify updated rows logically rather than resorting to ROWIDs.

Identification key logging is necessary when supplemental log data will be the source of change in another database, such as a logical standby.

To enable identification key logging, execute the following statement:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX)
COLUMNS;
```

This statement results in all primary key values, database-wide, being logged regardless of whether or not any of them are modified.

If a table does not have a primary key, but has one or more non-null unique key constraints, one of the constraints is chosen arbitrarily for logging as a means of identifying the row getting updated.

If the table has neither a primary key nor a unique index, then all columns except LONG and LOB are supplementally logged. Therefore, Oracle Corporation recommends that when you use supplemental logging, all or most tables be defined to have primary or unique keys.

Note: Regardless of whether or not identification key logging is enabled, the SQL statements returned by LogMiner always contain the ROWID clause. You can filter out the ROWID clause by using the RTRIM function and appropriate arguments on the reconstructed SQL statement.

To disable either minimal or identification key logging, execute the following statement.

```
SQL> ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

Usage Notes for Identification Key Logging

Keep the following in mind when you use identification key logging:

- Identification key logging is not required for delete operations because DELETE statements contain all the column values required to identify a row.
- If the database is open when you enable identification key logging, all DML cursors in the cursor cache are invalidated. This can have a performance impact until the cache is repopulated.

Table Supplemental Logging

Table supplemental logging uses log groups to log supplemental information. There are two types of log groups:

- **Unconditional log groups** - The before images of specified columns are logged any time the table is updated, regardless of whether the update affected any of the specified columns. This is sometimes referred to as an **ALWAYS** log group.
- **Conditional log groups** - The before images of all specified columns are logged only if at least one of the columns in the log group is updated.

Unconditional Log Groups

To enable supplemental logging that uses unconditional log groups, use the **ALWAYS** clause as shown in the following example:

```
SQL> ALTER TABLE scott.emp
      2 ADD SUPPLEMENTAL LOG GROUP emp_parttime (empno, ename, deptno) ALWAYS;
```

This creates a log group named `emp_parttime` on `scott.emp` that consists of the columns `empno`, `ename`, and `deptno`. These columns will be logged every time an **UPDATE** statement is executed on `scott.emp`, regardless of whether or not the update affected them. If you wanted to have the entire row image logged any time an update was made, you could create a log group that contained all the columns in the table.

Note: LOBs, LONGs, and ADTs cannot be part of a log group

Conditional Log Groups

To enable supplemental logging that uses conditional log groups, omit the **ALWAYS** clause from your **ALTER TABLE** statement, as shown in the following example:

```
SQL> ALTER TABLE scott.emp
      2 ADD SUPPLEMENTAL LOG GROUP emp_fulltime (empno, ename, deptno);
```

This creates a log group named `emp_fulltime` on `scott.emp`. Just like the previous example, it consists of the columns `empno`, `ename`, and `deptno`. But because the **ALWAYS** clause was omitted, before images of the columns will be logged only if at least one of the columns is updated.

Usage Notes for Log Groups

Keep the following in mind when you use log groups:

- A column can belong to more than one log group. However, the before image of the columns gets logged only once.
- Redo logs do not contain any information about which log group a column is part of or whether a column's before image is being logged because of log group logging or identification key logging.
- If you specify the same columns to be logged both conditionally and unconditionally, the columns are logged unconditionally.

Steps in a Typical LogMiner Session

This section describes the steps in a typical LogMiner session. Each step is described in its own subsection.

1. [Perform Initial Setup Activities](#)
2. [Extract a Dictionary](#) (unless you plan to use the online catalog)
3. [Specify Redo Logs for Analysis](#)
4. [Start a LogMiner Session](#)
5. [Query V\\$LOGMNR_CONTENTS](#)
6. [End a LogMiner Session](#)

To run LogMiner, you use the `DBMS_LOGMNR` PL/SQL package. Additionally, you might also use the `DBMS_LOGMNR_D` package if you choose to extract a dictionary rather than use the online catalog.

The `DBMS_LOGMNR` package contains the procedures used to initialize and run LogMiner, including interfaces to specify names of redo logs, filter criteria, and session characteristics. The `DBMS_LOGMNR_D` package queries the dictionary tables of the current database to create a LogMiner dictionary file.

The LogMiner packages are owned by the `SYS` schema. Therefore, if you are not connected as user `SYS`, you must include `SYS` in your call. For example:

```
EXECUTE SYS.DBMS_LOGMNR.END_LOGMNR
```

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for details about syntax and parameters for these LogMiner packages
- *Oracle9i Application Developer's Guide - Fundamentals* for information about executing PL/SQL procedures

Perform Initial Setup Activities

There are initial setup activities that you must perform before using LogMiner for the first time. You only need to perform these activities once, not every time you use LogMiner:

- Enable the type of supplemental logging you want to use. At the very least, Oracle Corporation recommends that you enable minimal supplemental logging, as follows:

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
```

See [Supplemental Logging](#) on page 9-19 for more information.

- Use the `DBMS_LOGMNR_D.SET_TABLESPACE` routine to re-create all LogMiner tables in an alternate tablespace. For example:

```
SQL> EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmrts$');
```

See [Recommendations](#) on page 9-10 for more information.

Extract a Dictionary

To use LogMiner you must supply it with a dictionary by doing one of the following:

- Extract database dictionary information to a flat file. See [Extracting the Dictionary to a Flat File](#) on page 9-6.
- Extract database dictionary information to the redo logs. See [Extracting a Dictionary to the Redo Logs](#) on page 9-7.
- Specify use of the online catalog by using the `DICT_FROM_ONLINE_CATALOG` option when you start LogMiner. See [Using the Online Catalog](#) on page 9-8.

Specify Redo Logs for Analysis

Before you can start LogMiner, you must specify the redo logs that you want to analyze. To do so, execute the `DBMS_LOGMNR.ADD_LOGFILE` procedure, as demonstrated in the following steps. You can add and remove redo logs in any order.

Note: If you will be mining in the same instance that is generating the redo logs, you only need to specify one archived redo log and the `CONTINUOUS_MINE` option when you start LogMiner. See [Continuous Mining](#) on page 9-25.

1. Use SQL*Plus to start an Oracle instance, with the database either mounted or unmounted. For example, enter:

```
SQL> STARTUP
```

2. Create a list of redo logs. Specify the `NEW` option of the `DBMS_LOGMNR.ADD_LOGFILE` procedure to signal that this is the beginning of a new list. For example, enter the following to specify `/oracle/logs/log1.f`:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME => '/oracle/logs/log1.f', -
  3 OPTIONS => DBMS_LOGMNR.NEW);
```

3. If desired, add more redo logs by specifying the `ADDFILE` option of the `DBMS_LOGMNR.ADD_LOGFILE` procedure. For example, enter the following to add `/oracle/logs/log2.f`:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME => '/oracle/logs/log2.f', -
  3 OPTIONS => DBMS_LOGMNR.ADDFILE);
```

The `OPTIONS` parameter is optional when you are adding additional redo logs. For example, you could simply enter the following:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME=>'/oracle/logs/log2.f');
```

4. If desired, remove redo logs by specifying the `REMOVEFILE` option of the `DBMS_LOGMNR.ADD_LOGFILE` procedure. For example, enter the following to remove `/oracle/logs/log2.f`:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME => '/oracle/logs/log2.f', -
  3 OPTIONS => DBMS_LOGMNR.REMOVEFILE);
```

Continuous Mining

The continuous mining option is useful if you are mining in the same instance that is generating the redo logs. When you plan to use the continuous mining option,

you only need to specify one archived redo log before starting LogMiner. Then, when you start LogMiner specify the `DBMS_LOGMNR.CONTINUOUS_MINE` option, which directs LogMiner to automatically add and mine subsequent archived redo logs and also the online catalog.

Note: Continuous mining is not available in a Real Application Clusters environment.

Start a LogMiner Session

After you have created a dictionary file and specified which redo logs to analyze, you can start a LogMiner session. Take the following steps:

1. Execute the `DBMS_LOGMNR.START_LOGMNR` procedure to start LogMiner.

It is recommended that you specify a dictionary option. If you do not, LogMiner cannot translate internal object identifiers and datatypes to object names and external data formats. Therefore, it would return internal object IDs and present data as hex bytes. Additionally, the `MINE_VALUE` and `COLUMN_PRESENT` functions cannot be used without a dictionary.

If you are specifying the name of a flat file dictionary, you must supply a fully qualified filename for the dictionary file. For example, to start LogMiner using `/oracle/database/dictionary.ora`, issue the following command:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -  
      2 DICTFILENAME =>' /oracle/database/dictionary.ora');
```

If you are not specifying a flat file dictionary name, then use the `OPTIONS` parameter to specify either the `DICTIONARY_FROM_REDO_LOGS` or `DICTIONARY_FROM_ONLINE_CATALOG` option.

If you specify `DICTIONARY_FROM_REDO_LOGS`, LogMiner expects to find a dictionary in the redo logs that you specified with the `DBMS_LOGMNR.ADD_LOGFILE` procedure. To determine which redo logs contain a dictionary, look at the `V$ARCHIVED_LOG` view. See [Extracting a Dictionary to the Redo Logs](#) on page 9-7 for an example.

Note: If you add additional redo logs after your LogMiner session has been started, you must restart LogMiner. You can specify new startup parameters if desired. Otherwise, LogMiner uses the parameters you specified for the previous session.

For more information on using the online catalog, see [Using the Online Catalog](#) on page 9-8.

2. Optionally, you can filter your query by time or by SCN. See [Filtering Data By Time](#) on page 9-14 or [Filtering Data By SCN](#) on page 9-15.
3. You can also use the `OPTIONS` parameter to specify additional characteristics of your LogMiner session. For example, you might decide to use the online catalog as your dictionary and to have only committed transactions shown in the `V$LOGMNR_CONTENTS` view, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR(OPTIONS => -
      2 DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG + -
      3 DBMS_LOGMNR.COMMITTED_DATA_ONLY);
```

The following list is a summary of LogMiner settings that you can specify with the `OPTIONS` parameter and where to find more information about them.

- `DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG` — See [Using the Online Catalog](#) on page 9-8
- `DBMS_LOGMNR.DICT_FROM_REDO_LOGS` — See step 1 in this list
- `DBMS_LOGMNR.COMMITTED_DATA_ONLY` — See [Showing Only Committed Transactions](#) on page 9-12
- `DBMS_LOGMNR.SKIP_CORRUPTION` — See [Skipping Redo Corruptions](#) on page 9-14
- `DBMS_LOGMNR.DDL_DICT_TRACKING` — See [Tracking DDL Statements](#) on page 9-9
- `DBMS_LOGMNR.NEW`, `DBMS_LOGMNR.ADDFILE`, and `DBMS_LOGMNR.REMOVEFILE` — See [Specify Redo Logs for Analysis](#) on page 9-24
- `DBMS_LOGMNR.NO_SQL_DELIMITER` — See [Formatting of Returned Data](#) on page 9-17
- `DBMS_LOGMNR.PRINT_PRETTY_SQL` — See [Formatting of Returned Data](#) on page 9-17
- `DBMS_LOGMNR.CONTINUOUS_MINE` — See [Continuous Mining](#) on page 9-25

You can execute the `DBMS_LOGMNR.START_LOGMNR` procedure multiple times, specifying different options each time. This can be useful for example, if you did not get the desired results from a query of `V$LOGMNR_CONTENTS`, and

want to restart LogMiner with different options. You do not need to re-add redo logs that were already added for a previous session.

Query V\$LOGMNR_CONTENTS

At this point, LogMiner is started and you can perform queries against the V\$LOGMNR_CONTENTS view. See [Querying V\\$LOGMNR_CONTENTS](#) on page 9-16 for examples of this.

End a LogMiner Session

To properly end a LogMiner session, use the DBMS_LOGMNR.END_LOGMNR procedure, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.END_LOGMNR;
```

This procedure closes all the redo logs and allows all the database and system resources allocated by LogMiner to be released.

If this procedure is not executed, LogMiner retains all its allocated resources until the end of the Oracle session in which it was invoked. It is particularly important to use this procedure to end LogMiner if either the DDL_DICT_TRACKING option or the DICT_FROM_REDO_LOGS option was used.

Example Uses of LogMiner

This section provides the following example uses of LogMiner.

- [Example: Using LogMiner to Track Changes Made By a Specific User](#)
- [Example: Using LogMiner to Calculate Table Access Statistics](#)

Example: Using LogMiner to Track Changes Made By a Specific User

This example shows how to see all changes made to the database in a specific time range by one of your users: joedevo. Connect to the database and then take the following steps:

- [Step 1: Creating the Dictionary File](#)
- [Step 2: Adding Redo Logs](#)
- [Step 3: Starting LogMiner and Limiting the Search Range](#)
- [Step 4: Querying V\\$LOGMNR_CONTENTS](#)

Step 1: Creating the Dictionary File To use LogMiner to analyze joedevo's data, you must either create a dictionary file before joedevo makes any changes or specify use of the online catalog at LogMiner startup. See [Extract a Dictionary](#) on page 9-24 for examples of creating dictionaries.

Step 2: Adding Redo Logs Assume that joedevo has made some changes to the database. You can now specify the names of the redo logs that you want to analyze, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME => 'log1orcl.ora', -
  3 OPTIONS => DBMS_LOGMNR.NEW);
```

If desired, add additional redo logs, as follows:

```
SQL> EXECUTE DBMS_LOGMNR.ADD_LOGFILE( -
  2 LOGFILENAME => 'log2orcl.ora', -
  3 OPTIONS => DBMS_LOGMNR.ADDFILE);
```

Step 3: Starting LogMiner and Limiting the Search Range Start LogMiner and limit the search to the specified time range:

```
SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -
  2 DICTFILENAME => 'orcdict.ora', -
  3 STARTTIME => TO_DATE('01-Jan-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS'), -
  4 ENDTIME => TO_DATE('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

Step 4: Querying V\$LOGMNR_CONTENTS At this point, the V\$LOGMNR_CONTENTS view is available for queries. You decide to find all of the changes made by user joedevo to the salary table. Execute the following SELECT statement:

```
SQL> SELECT SQL_REDO, SQL_UNDO FROM V$LOGMNR_CONTENTS
  2 WHERE USERNAME = 'joedevo' AND SEG_NAME = 'salary';
```

For both the SQL_REDO and SQL_UNDO columns, two rows are returned (the format of the data display will be different on your screen). You discover that joedevo requested two operations: he deleted his old salary and then inserted a new, higher salary. You now have the data necessary to undo this operation.

SQL_REDO	SQL_UNDO
-----	-----
delete * from SALARY	insert into SALARY(NAME, EMPNO, SAL)
where EMPNO = 12345	values ('JOEDEVO', 12345, 500)
and ROWID = 'AAABOOAABAAEPCABA';	

```

insert into SALARY(NAME, EMPNO, SAL)      delete * from SALARY
values('JOEDEVO',12345, 2500)           where EMPNO = 12345
                                         and ROWID = 'AAABOOAABAAEPCABA';

2 rows selected

```

Example: Using LogMiner to Calculate Table Access Statistics

In this example, assume you manage a direct marketing database and want to determine how productive the customer contacts have been in generating revenue for a two week period in August. Assume that you have already created the dictionary and added the redo logs you want to search (as demonstrated in the previous example). Take the following steps:

1. Start LogMiner and specify a range of times:

```

SQL> EXECUTE DBMS_LOGMNR.START_LOGMNR( -
      2 STARTTIME => TO_DATE('07-Aug-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS'), -
      3 ENDTIME => TO_DATE('21-Aug-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'), -
      4 DICTFILENAME => '/usr/local/dict.ora');

```

2. Query the V\$LOGMNR_CONTENTS view to determine which tables were modified in the time range you specified, as shown in the following example. (This query filters out system tables that traditionally have a \$ in their name.)

```

SQL> SELECT SEG_OWNER, SEG_NAME, COUNT(*) AS Hits FROM
      2 V$LOGMNR_CONTENTS WHERE SEG_NAME NOT LIKE '%$' GROUP BY
      3 SEG_OWNER, SEG_NAME;

```

3. The following data is displayed. (The format of your display may be different.)

SEG_OWNER	SEG_NAME	Hits
CUST	ACCOUNT	384
SCOTT	EMP	12
SYS	DONOR	12
UNIV	DONOR	234
UNIV	EXECDONOR	325
UNIV	MEGADONOR	32

The values in the `Hits` column show the number of times that the named table had an insert, delete, or update operation performed on it during the two week period specified in the query.

10

Managing Job Queues

This chapter describes how to use job queues to schedule the periodic execution of user jobs, and contains the following topics:

- [Enabling Processes Used for Executing Jobs](#)
- [Managing Job Queues](#)
- [Viewing Job Queue Information](#)

Enabling Processes Used for Executing Jobs

You can schedule routines (jobs) to be run periodically using the job queue. To schedule a job you submit it to the job queue, using the Oracle supplied `DBMS_JOB` package, and specify the frequency at which the job is to be run. Additional functionality enables you to alter, disable, or delete a job that you previously submitted.

Job queue (*Jnnn*) processes execute jobs in the job queue. For each instance, these job queue processes are dynamically spawned by a coordinator job queue (CJQ0) background process. The coordinator periodically selects jobs that are ready to run from the jobs shown in the `DBA_JOBS` view. It orders them by time, and then spawns *Jnnn* processes to run the selected jobs. Each *Jnnn* process executes one of the selected jobs.

The `JOB_QUEUE_PROCESSES` initialization parameter controls whether a coordinator job queue process is started by an instance. If this parameter is set to 0, no coordinator job queue process is started at database startup, and consequently no job queue jobs are executed. The `JOB_QUEUE_PROCESSES` initialization parameter also specifies the maximum number of *Jnnn* processes that can concurrently run on an instance. The maximum number of processes that can be specified is 1000.

The following initialization parameter setting causes the coordinator job queue process to start at database startup, and allows the spawning of a maximum of 60 concurrent *Jnnn* processes.

```
JOB_QUEUE_PROCESSES = 60
```

In any given period that the coordinator job queue process scans the jobs shown in the `DBA_JOBS` view, it spawns at most only the number of *Jnnn* processes required to execute the jobs it has selected. While the above example allows for 60 concurrent *Jnnn* processes, if only 20 jobs are selected for execution, then the coordinator spawns, or reuses, only the number of *Jnnn* processes necessary to execute the 20 jobs (at least, 20). Any idle existing *Jnnn* processes are considered available for reuse.

When a *Jnnn* process finishes execution of a job, it polls for another job to execute. If there are no jobs selected for execution, it enters an idle state, but wakes up periodically to poll again. If, after a predetermined number of tries, it still finds no jobs to execute, it terminates.

The `JOB_QUEUE_PROCESSES` initialization parameter is dynamic and it can be modified by an `ALTER SYSTEM` statement. For example, the following statement sets the maximum number of concurrent *Jnnn* processes allowed to 20.

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 20;
```

If the new value is lower than the previous setting and less than the number of currently executing *Jnnn* processes, the excess processes are allowed to complete before they are terminated.

Jnnn processes will not execute jobs if the instance is running in restricted mode.

See also: ["Restricting Access to an Open Database"](#) on page 4-11 for information about enabling and disabling restricted mode

Managing Job Queues

This section describes the various aspects of managing job queues and contains the following topics:

- [The DBMS_JOB Package](#)
- [Submitting a Job to the Job Queue](#)
- [How Jobs Execute](#)
- [Removing a Job from the Job Queue](#)
- [Altering a Job](#)
- [Broken Jobs](#)
- [Forcing a Job to Execute](#)
- [Terminating a Job](#)

The DBMS_JOB Package

To schedule and manage jobs in the job queue, use the procedures in the DBMS_JOB package. There are no database privileges associated with using job queues. Any user who can execute the job queue procedures can use the job queue.

The following are procedures of the DBMS_JOB package. They are described in this section as noted.

Procedure	Description
SUBMIT	Submits a job to the job queue. See "Submitting a Job to the Job Queue" on page 10-4.

Procedure	Description
REMOVE	Removes a specified job from the job queue. See "Removing a Job from the Job Queue" on page 10-10.
CHANGE	Alters a specified job that has already been submitted to the job queue. You can alter the job description, the time at which the job will be run, or the interval between executions of the job. See "Altering a Job" on page 10-11.
WHAT	Alters the job description for a specified job. See "Altering a Job" on page 10-11.
NEXT_DATE	Alters the next execution time for a specified job. See "Altering a Job" on page 10-11.
INTERVAL	Alters the interval between executions for a specified job. See "Altering a Job" on page 10-11.
BROKEN	Sets or resets the job broken flag. If a job is marked as broken, Oracle does not attempt to execute it. See "Broken Jobs" on page 10-12.
RUN	Forces a specified job to run. See "Forcing a Job to Execute" on page 10-14.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for syntax information for the DBMS_JOB package, and for information about other options available when using the DBMS_JOB package in an Oracle Real Application Clusters environment

Submitting a Job to the Job Queue

To submit a new job to the job queue, use the SUBMIT procedure in the DBMS_JOB package. You specify the following parameters with the SUBMIT procedure:

Parameter	Description
JOB	An output parameter. This is the identifier assigned to the job you are creating. You must use this job number whenever you want to alter or remove the job. See "Job Number" on page 10-7.
WHAT	This is the PL/SQL code you want to have executed. See "Job Definition" on page 10-7.

Parameter	Description
NEXT_DATE	This is the next date when the job will be run. The default value is SYSDATE.
INTERVAL	This is the date function that calculates the next time to execute the job. The default value is NULL. INTERVAL must evaluate to a future point in time or NULL. See "Job Execution Interval" on page 10-8.
NO_PARSE	This is a flag. If NO_PARSE is set to FALSE (the default), Oracle parses the procedure associated with the job. If NO_PARSE is set to TRUE, Oracle parses the procedure associated with the job the first time that the job is executed. If, for example, you want to submit a job before you have created the tables associated with the job, set NO_PARSE to TRUE.

For example, consider the following statements that submit a new job to the job queue, then prints the job number. The job calls the procedure `DBMS_DDL.ANALYZE_OBJECT` to generate statistics for the table `hr.employees`. The statistics are based on a sample of half the rows of the `employees` table. The job is run every 24 hours.

```
VARIABLE jobno NUMBER
BEGIN
  DBMS_JOB.SUBMIT(: jobno,
    'DBMS_DDL.ANALYZE_OBJECT(''TABLE'',
    ''HR'', ''EMPLOYEES'',
    ''ESTIMATE'', NULL, 50);',
    SYSDATE, 'SYSDATE + 1');
  COMMIT;
END;
/
PRINT jobno

JOBNO
-----
14144
```

Note: For the submitted job to run, you must issue a `COMMIT` statement immediately after the `DBMS_JOB.SUBMIT` statement.

Job Environment

When you submit a job to the job queue or alter a job's definition, Oracle records the following environment characteristics:

- The current user
- The user submitting or altering a job
- The current schema (may be different from current user or submitting user if `ALTER SESSION SET CURRENT_SCHEMA` statement has been issued)

Oracle also records the following NLS parameters:

- `NLS_LANGUAGE`
- `NLS_TERRITORY`
- `NLS_CURRENCY`
- `NLS_ISO_CURRENCY`
- `NLS_NUMERIC_CHARACTERS`
- `NLS_DATE_FORMAT`
- `NLS_DATE_LANGUAGE`
- `NLS_SORT`

Oracle restores all of these environment characteristics every time a job is executed. `NLS_LANGUAGE` and `NLS_TERRITORY` parameters determine the defaults for unspecified NLS parameters.

You can change a job's environment by using the `DBMS_SQL` package and the `ALTER SESSION` statement.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_SQL` package
- *Oracle9i SQL Reference* for information about use of the `ALTER SESSION` statement to alter a job's environment

Jobs and Import/Export

Jobs can be exported and imported. Thus, if you define a job in one database, you can transfer it to another database. When exporting and importing jobs, the job's number, environment, and definition remain unchanged.

Note: If the job number of a job you want to import matches the number of a job already existing in the database, you will not be allowed to import that job. Submit the job as a new job in the database.

Job Owner

When you submit a job to the job queue, Oracle identifies you as the owner of the job. Only a job's owner can alter the job, force the job to run, or remove the job from the queue.

Job Number

A queued job is identified by its job number. When you submit a job, its job number is automatically generated from the `JOBSEQ` sequence owned by user `SYS`. Once a job is assigned a job number, that number does not change. Even if the job is exported and imported, its job number remains the same.

Job Definition

The job definition is the PL/SQL code specified in the `WHAT` parameter of the `SUBMIT` procedure. Normally, the job definition is a single call to a procedure. The procedure call can have any number of parameters.

Note: In the job definition, use two single quotation marks around strings. Always include a semicolon at the end of the job definition.

The following are examples of valid job definitions:

- `'myproc(''10-JAN-99'', next_date, broken);'`
- `'scott.emppackage.give_raise(''JFEE'', 3000.00);'`
- `'dbms_job.remove(job);'`

Note: Running a job from a job is not supported. You will receive an error message if you attempt to do so. For example, the following statements produce the "ORA-32317 cannot run a job from another job" error message in the alert file:

```

DECLARE
    jobno number;
BEGIN
    DBMS_JOB.SUBMIT(jobno, 'DBMS_JOB.RUN(23587);');
    DBMS_JOB.RUN(jobno);
END;
/
    
```

Job Execution Interval

If a job should be executed periodically at a set interval, use a date expression similar to 'SYSDATE + 7' in the INTERVAL parameter. Below are shown some common date expressions used for job execution intervals.

Date Expression	Evaluation
'SYSDATE + 7'	Exactly seven days from the last execution
'SYSDATE + 1/48'	Every half hour
'NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24'	Every Monday at 3PM
'NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE), 'Q'), 3), 'THURSDAY'''	First Thursday of each quarter

Note: When specifying NEXT_DATE or INTERVAL, remember that date literals and strings must be enclosed in single quotation marks. Also, the value of INTERVAL must be enclosed in single quotation marks.

The INTERVAL date function is evaluated immediately before a job is executed. When the job completes successfully, the date calculated from INTERVAL becomes the new NEXT_DATE. For example, if you set the execution interval to 'SYSDATE + 7' on Monday, but for some reason (such as a network failure) the job is not executed until Thursday, 'SYSDATE + 7' then executes every Thursday, not Monday. If the INTERVAL date function evaluates to NULL and the job completes successfully, the job is deleted from the queue.

If you always want to automatically execute a job at a specific time, regardless of the last execution (for example, every Monday), the `INTERVAL` and `NEXT_DATE` parameters should specify a date expression similar to `'NEXT_DAY (TRUNC (SYSDATE), 'MONDAY')'`.

Database Links and Jobs

If you submit a job that uses a database link, the link must include a username and password. Anonymous database links will not succeed.

How Jobs Execute

Jnnn processes execute jobs. To execute a job, the process creates a session to run the job. When a *Jnnn* process runs a job, the job is run in the same environment in which it was submitted and with the *owner's* default privileges. The owner must be explicitly granted the necessary object privileges for all objects referenced within the job definition.

When you force a job to run using the procedure `DBMS_JOB.RUN`, the job is run by your user process and with *your* default privileges only. Privileges granted to you through roles are unavailable. You must be explicitly granted the necessary object privileges for all objects referenced within the job definition.

Job Queue Locks

Oracle uses job queue locks to ensure that a job is executed in only one session at a time. When a job is being run, its session acquires a job queue (JQ) lock for that job. You can use the locking views in the data dictionary to examine information about locks currently held by sessions.

The following query lists the session identifier, lock type, and lock identifiers for all sessions holding JQ locks:

```
SELECT SID, TYPE, ID1, ID2
   FROM V$LOCK
  WHERE TYPE = 'JQ';
```

SID	TY	ID1	ID2
12	JQ	0	14144

1 row selected.

In the query above, the identifier for the session holding the lock is 12. The `ID1` column is always 0 for JQ locks. The `ID2` column is the job number of the job the

session is running. This view can be joined with the `DBA_JOBS_RUNNING` view to obtain more information about the job.

See Also:

- ["Viewing Job Queue Information"](#) on page 10-15 for more information about views
- *Oracle9i Database Reference* for more information about the `V$LOCK` view
- *Oracle9i Database Concepts* for more information about locking

Job Execution Errors

When a job fails, information about the failure is recorded in a trace file and the alert log. Oracle writes message number `ORA-12012` and includes the job number of the failed job.

The following can prevent the successful execution of queued jobs:

- A network or instance failure
- An exception when executing the job

If a job returns an error while Oracle is attempting to execute it, Oracle tries to execute it again. The first attempt is made after one minute, the second attempt after two minutes, the third after four minutes, and so on, with the interval doubling between each attempt. If the job fails 16 times, Oracle automatically marks the job as broken and no longer tries to execute it. However, between attempts, you have the opportunity to correct the problem that is preventing the job from running. This will not disturb the retry cycle, and Oracle will eventually attempt to run the job again.

Removing a Job from the Job Queue

To remove a job from the job queue, use the `REMOVE` procedure in the `DBMS_JOB` package.

The following statements remove job number 14144 from the job queue:

```
BEGIN
DBMS_JOB.REMOVE(14144);
END;
/
```

Restrictions:

- You can remove currently executing jobs from the job queue. However, the job will not be interrupted, and the current execution will be completed.
- You can remove only jobs you own. If you try to remove a job that you do not own, you receive a message that states the job is not in the job queue.

Altering a Job

To alter a job that has been submitted to the job queue, use the procedures `CHANGE`, `WHAT`, `NEXT_DATE`, or `INTERVAL` in the `DBMS_JOB` package.

Restriction:

- You can alter only jobs that you own. If you try to alter a job that you do not own, you receive a message that states the job is not in the job queue.

CHANGE

You can alter any of the user-definable parameters associated with a job by calling the `DBMS_JOB.CHANGE` procedure.

In this example, job number 14144 is altered to execute every three days:

```
BEGIN
DBMS_JOB.CHANGE(14144, NULL, NULL, 'SYSDATE + 3');
END;
/
```

If you specify `NULL` for `WHAT`, `NEXT_DATE`, or `INTERVAL` when you call the procedure `DBMS_JOB.CHANGE`, the current value remains unchanged.

Note: When you change a job's definition using the `WHAT` parameter in the procedure `DBMS_JOB.CHANGE`, Oracle records your current environment. This becomes the new environment for the job.

WHAT

You can alter the definition of a job by calling the `DBMS_JOB.WHAT` procedure.

The following example changes the definition for job number 14144:

```
BEGIN
DBMS_JOB.WHAT(14144,
```

```
'DBMS_DDL.ANALYZE_OBJECT('TABLE',  
'HR', 'DEPARTMENTS',  
'ESTIMATE', NULL, 50));  
END;  
/
```

Note: When you execute the procedure `DBMS_JOB.WHAT`, Oracle records your current environment. This becomes the new environment for the job.

NEXT_DATE

You can alter the next execution time for a job by calling the `DBMS_JOB.NEXT_DATE` procedure, as shown in the following example:

```
BEGIN  
DBMS_JOB.NEXT_DATE(14144, SYSDATE + 4);  
END;  
/
```

INTERVAL

The following example illustrates changing the execution interval for a job by calling the `DBMS_JOB.INTERVAL` procedure:

```
BEGIN  
DBMS_JOB.INTERVAL(14144, 'NULL');  
END;  
/
```

In this case, the job will not run again after it successfully executes and it will be deleted from the job queue.

Broken Jobs

A job is labeled as either broken or not broken. Oracle does not attempt to run broken jobs. However, you can force a broken job to run by calling the procedure `DBMS_JOB.RUN`.

How a Job Becomes Broken

When you submit a job it is considered not broken.

There are two ways a job can break:

- Oracle has failed to successfully execute the job after 16 attempts.
- You have marked the job as broken, using the procedure `DBMS_JOB.BROKEN`:

```
BEGIN
DBMS_JOB.BROKEN(14144, TRUE);
END;
/
```

Once a job has been marked as broken, Oracle will not attempt to execute the job until you either mark the job as not broken, or force the job to be executed by calling the procedure `DBMS_JOB.RUN`.

The following example marks job 14144 as not broken and sets its next execution date to the following Monday:

```
BEGIN
DBMS_JOB.BROKEN(14144, FALSE, NEXT_DAY(SYSDATE, 'MONDAY'));
END;
/
```

Restriction:

- You can mark as broken only jobs that you own. If you call `DBMS_JOB.BROKEN` for a job that you do not own, you receive a message stating that the job is not in the job queue.

Running Broken Jobs

If a problem has caused a job to fail 16 times, Oracle marks the job as broken. Once you have fixed this problem, you can run the job by either:

- Forcing the job to run by calling `DBMS_JOB.RUN`
- Marking the job as not broken by calling `DBMS_JOB.BROKEN` and waiting for Oracle to execute the job

If you force the job to run by calling the procedure `DBMS_JOB.RUN`, Oracle runs the job immediately. If the job succeeds, then Oracle labels the job as not broken and resets its count of the number of failed executions for the job to zero.

Once you reset a job's broken flag (by calling either `RUN` or `BROKEN`), job execution resumes according to the scheduled execution intervals set for the job.

Forcing a Job to Execute

There may be times when you would like to manually execute a job. For example, if you have fixed a broken job, you may want to test the job immediately by forcing it to execute. To force a job to execute immediately, use the procedure `RUN` in the `DBMS_JOB` package.

When you run a job using `DBMS_JOB.RUN`, Oracle recomputes the next execution date. For example, if you create a job on a Monday with a `NEXT_DATE` value of `SYSDATE` and an `INTERVAL` value of `'SYSDATE + 7'`, the job is run every 7 days starting on Monday. However, if you execute `RUN` on Wednesday, the next execution date will be set to the next Wednesday.

The following statement runs job 14144 in your session and recomputes the next execution date:

```
BEGIN
DBMS_JOB.RUN(14144);
END;
/
```

Note: When you force a job to run, the job is executed in your current session. Running the job reinitializes your session's packages.

Restrictions:

- You can only run jobs that you own. If you try to run a job that you do not own, you receive a message that states the job is not in the job queue.
- The procedure `RUN` contains an implicit commit. Once you execute a job using `RUN`, you cannot roll back.

Terminating a Job

You can terminate a running job by marking the job as broken, identifying the session running the job, and disconnecting that session. You should mark the job as broken, so that Oracle does not attempt to run the job again.

After you have identified the session running the job (using `V$SESSION` or `V$LOCK`, as shown earlier), you can disconnect the session using the SQL statement `ALTER SYSTEM`. For examples of viewing information about jobs and sessions, see the next section, "[Viewing Job Queue Information](#)".

See Also:

- *Oracle9i Database Reference* for more information on V\$SESSION
- ["Terminating Sessions"](#) on page 10-2

Viewing Job Queue Information

You can view information about jobs in the job queue using the data dictionary views listed below:

View	Description
DBA_JOBS ALL_JOBS USER_JOBS	DBA view describes all the jobs in the database. ALL view describes all jobs that are accessible to the current user. USER view describes all jobs owned by the current user.
DBA_JOBS_RUNNING	Lists all jobs in the database that are currently running. This view can be joined with V\$LOCK to identify jobs that have locks.

Displaying Information About a Job

The following query creates a listing of the job number, next execution time, failure count, and broken status for each job you have submitted:

```
SELECT JOB, NEXT_DATE, NEXT_SEC, FAILURES, BROKEN
FROM DBA_JOBS;
```

```
JOB      NEXT_DATE  NEXT_SEC  FAILURES  B
-----  -
    9125  01-JUN-01  00:00:00         4  N
   14144  24-OCT-01  16:35:35         0  N
    9127  01-JUN-01  00:00:00        16  Y
```

3 rows selected.

Displaying Information About Running Jobs

You can also display information about only the jobs currently running. The following query lists the session identifier, job number, user who submitted the job, and the start times for all currently running jobs:

```
SELECT SID, r.JOB, LOG_USER, r.THIS_DATE, r.THIS_SEC
FROM DBA_JOBS_RUNNING r, DBA_JOBS j
WHERE r.JOB = j.JOB;
```

SID	JOB	LOG_USER	THIS_DATE	THIS_SEC
12	14144	HR	24-OCT-94	17:21:24
25	8536	QS	24-OCT-94	16:45:12

2 rows selected.

See Also: *Oracle9i Database Reference* for more information on data dictionary views

Managing Tablespaces

This chapter describes the various aspects of tablespace management, and contains the following topics:

- [Guidelines for Managing Tablespaces](#)
- [Creating Tablespaces](#)
- [Coalescing Free Space in Dictionary-Managed Tablespaces](#)
- [Specifying Nonstandard Block Sizes for Tablespaces](#)
- [Controlling the Writing of Redo Records](#)
- [Altering Tablespace Availability](#)
- [Using Read-Only Tablespaces](#)
- [Dropping Tablespaces](#)
- [Diagnosing and Repairing Locally Managed Tablespace Problems](#)
- [Migrating the SYSTEM Tablespace to a Locally Managed Tablespace](#)
- [Transporting Tablespaces Between Databases](#)
- [Viewing Tablespace Information](#)

See Also: [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating datafiles and tempfiles that are both created and managed by the Oracle database server

Guidelines for Managing Tablespaces

Before working with tablespaces of an Oracle database, familiarize yourself with the guidelines provided in the following sections:

- [Use Multiple Tablespaces](#)
- [Specify Tablespace Default Storage Parameters](#)
- [Assign Tablespace Quotas to Users](#)

See Also: *Oracle9i Database Concepts* for a complete discussion of database structure, space management, tablespaces, and datafiles

Use Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations. For example, when a database has multiple tablespaces, you can perform the following tasks:

- Separate user data from data dictionary data to reduce contention among dictionary objects and schema objects for the same datafiles.
- Separate one application's data from another's to prevent multiple applications from being affected if a tablespace must be taken offline.
- Store different tablespaces' datafiles on separate disk drives to reduce I/O contention.
- Separate rollback segment data from user data, preventing a single disk failure from causing permanent loss of data.
- Take individual tablespaces offline while others remain online, providing better overall availability.
- Reserve a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage. This enables you to optimize usage of the tablespace.
- Back up individual tablespaces.

Some operating systems set a limit on the number of files that can be simultaneously open. These limits can affect the number of tablespaces that can be simultaneously online. To avoid exceeding your operating system's limit, plan your tablespaces efficiently. Create only enough tablespaces to fill your needs, and create these tablespaces with as few files as possible. If you need to increase the size of a

tablespace, add one or two large datafiles, or create datafiles with the autoextend option set on, rather than many small datafiles.

Review your data in light of these factors and decide how many tablespaces you need for your database design.

Specify Tablespace Default Storage Parameters

When you create a new dictionary-managed tablespace, you can specify default storage parameters for objects that will be created in the tablespace. Storage parameters specified when an object is created override the default storage parameters of the tablespace containing the object. If you do not specify storage parameters when creating an object, the object's segment automatically uses the default storage parameters for the tablespace.

Set the default storage parameters for a tablespace to account for the size of a typical object that the tablespace will contain (you estimate this size). You can specify different storage parameters for an unusual or exceptional object when creating that object. You can also alter your default storage parameters at a later time.

You cannot specify default storage parameters for tablespaces that are specifically created as locally managed.

Note: If you do not specify the default storage parameters for a new dictionary-managed tablespace, Oracle chooses default storage parameters appropriate for your operating system.

Assign Tablespace Quotas to Users

Grant to users who will be creating tables, clusters, materialized views, indexes, and other objects the privilege to create the object and a **quota** (space allowance or limit) in the tablespace intended to hold the object's segment. The security administrator is responsible for granting the required privileges to create objects to database users and for assigning tablespace quotas, as necessary, to database users.

See Also: ["Assigning Tablespace Quotas"](#) on page 24-4

Creating Tablespaces

Before you can create a tablespace you must create a database to contain it. The first tablespace in any database is always the `SYSTEM` tablespace, and the first datafiles

of any database are automatically allocated in the `SYSTEM` tablespace during database creation.

The steps for creating tablespaces vary by operating system. In all cases, however, you should create through your operating system a directory structure in which your datafiles will be allocated. On most operating systems you indicate the size and fully specified filenames when creating a new tablespace or altering a tablespace by adding datafiles. In each situation Oracle automatically allocates and formats the datafiles as specified.

To create a new tablespace, use the SQL statement `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE`. You must have the `CREATE TABLESPACE` system privilege to create a tablespace. Later, you can use the `ALTER TABLESPACE` or `ALTER DATABASE` statements to alter the tablespace. You must have the `ALTER TABLESPACE` or `ALTER DATABASE` system privilege, correspondingly.

Prior to Oracle8*i*, all tablespaces were created as **dictionary-managed**. Dictionary-managed tablespaces rely on data dictionary tables to track space utilization. Beginning with Oracle8*i*, you were able to create **locally managed** tablespaces, which use bitmaps (instead of data dictionary tables) to track used and free space. These locally managed tablespaces provide better performance and greater ease of management.

Note: Beginning in Oracle9*i* the default for non-`SYSTEM` permanent tablespaces is locally managed whenever both of the following criteria are met:

- The `EXTENT MANAGEMENT` clause is not specified
 - The `COMPATIBLE` initialization parameter is set to 9.0.0 or higher
-
-

You can also create a special type of tablespace called an undo tablespace. This tablespace is specifically designed to contain undo records. These are records generated by Oracle that are used to roll back, or undo, changes to the database for recovery, read consistency, or as requested by a `ROLLBACK` statement. Creating and managing undo tablespaces is the subject of [Chapter 13, "Managing Undo Space"](#).

Permanent and temporary tablespaces are discussed in the following sections:

- [Locally Managed Tablespaces](#)
- [Dictionary-Managed Tablespaces](#)

- **Temporary Tablespaces**

See Also:

- [Chapter 2, "Creating an Oracle Database"](#) and your Oracle installation documentation for your operating system for information about tablespaces that are created at database creation
- *Oracle9i SQL Reference* for more information about the syntax and use of the `CREATE TABLESPACE`, `CREATE TEMPORARY TABLESPACE`, `ALTER TABLESPACE`, and `ALTER DATABASE` statements.
- ["Specifying Database Block Sizes"](#) on page 2-37 for information about initialization parameters necessary to create tablespaces with nonstandard block sizes

Locally Managed Tablespaces

Locally managed tablespaces track all extent information in the tablespace itself, using bitmaps, resulting in the following benefits:

- Improved concurrency and speed of space operations, because space allocations and deallocations predominantly modify locally managed resources (bitmaps stored in header files) rather than requiring centrally managed resources such as enqueues
- Improved performance, because recursive operations that are sometimes required during dictionary-managed space allocation are eliminated
- Readable standby databases are allowed, because locally managed temporary tablespaces (used, for example, for sorts) are locally managed and thus do not generate any undo or redo.
- Simplified space allocation—when the `AUTOALLOCATE` clause is specified, appropriate extent size is automatically selected
- Reduced user reliance on the data dictionary because necessary information is stored in file headers and bitmap blocks

All tablespaces, including the `SYSTEM` tablespace, can be locally managed.

Additionally, the `DBMS_SPACE_ADMIN` package provides maintenance procedures for locally managed tablespaces.

See Also:

- ["Creating a Locally Managed SYSTEM Tablespace"](#) on page 2-26
- ["Diagnosing and Repairing Locally Managed Tablespace Problems"](#) on page 11-30

Creating a Locally Managed Tablespace

To create a locally managed tablespace, specify `LOCAL` in the `EXTENT MANAGEMENT` clause of the `CREATE TABLESPACE` statement. You then have two options. You can have Oracle manage extents for you automatically with the `AUTOALLOCATE` option (the default), or you can specify that the tablespace is managed with uniform extents of a specific size (`UNIFORM SIZE`).

If the tablespace is expected to contain objects of varying sizes requiring different extent sizes and having many extents, then `AUTOALLOCATE` is the best choice. If it is not important to you to have a lot of control over space allocation and deallocation, `AUTOALLOCATE` presents a simplified way for you to manage a tablespace. Some space may be wasted but the benefit of having Oracle manage your space most likely outweighs this drawback.

On the other hand, if you want exact control over unused space, and you can predict exactly the space to be allocated for an object or objects and the number and size of extents, then `UNIFORM` is a good choice. This ensures that you will never have an unusable amount of space in your tablespace.

Note: When you do not explicitly specify the type of extent management, and the default is to create a locally managed tablespace, Oracle determines extent management as described below.

If your `CREATE TABLESPACE` statement *does not* include a `DEFAULT` storage clause, then Oracle creates a locally managed autoallocated tablespace.

If your `CREATE TABLESPACE` statement *does* include a `DEFAULT` storage clause, then Oracle considers the following:

- If you specified the `MINIMUM EXTENT` clause, Oracle evaluates whether the values of `MINIMUM EXTENT`, `INITIAL`, and `NEXT` are equal and the value of `PCTINCREASE` is 0. If so, Oracle creates a locally managed uniform tablespace with extent size = `INITIAL`. If the `MINIMUM EXTENT`, `INITIAL`, and `NEXT` parameters are not equal, or if `PCTINCREASE` is not 0, Oracle ignores any extent storage parameters you may specify and creates a locally managed, autoallocated tablespace.
- If you did not specify `MINIMUM EXTENT` clause, Oracle evaluates only whether the storage values of `INITIAL` and `NEXT` are equal and `PCTINCREASE` is 0. If so, the tablespace is locally managed and uniform. Otherwise, the tablespace is locally managed and autoallocated.

The following statement creates a locally managed tablespace named `lmtbsb` and specifies `AUTOALLOCATE`:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

`AUTOALLOCATE` causes the tablespace to be system managed with the smallest extent size being 64K. There is an increase in *initial* space allocated for objects in autoallocated tablespaces. This is because the objects have a minimum size of two blocks in dictionary-managed tablespaces, whereas in autoallocated locally managed tablespaces, the minimum object size is 64K.

Alternatively, this tablespace could be created specifying the `UNIFORM` clause. If `UNIFORM SIZE` is specified, then the tablespace is managed with uniform size extents of the specified `SIZE`. The default `SIZE` is 1M.

In the following example, a 128K extent size is specified. Each 128K extent (which, if the tablespace block size is 2K, is equivalent to 64 Oracle blocks) is represented by a bit in the extent bitmap for this file.

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

You cannot specify the `DEFAULT` storage clause, `MINIMUM EXTENT`, or `TEMPORARY` when you explicitly specify `EXTENT MANAGEMENT LOCAL`. If you want to create a temporary locally managed tablespace, use the `CREATE TEMPORARY TABLESPACE` statement.

Note: When you allocate a datafile for a locally managed tablespace, you should allow space for metadata used for space management (the extent bitmap or space header segment) which are part of user space. For example, if you do not specify the `SIZE` parameter in the extent management clause when `UNIFORM` is specified, the default extent size is 1MB. Therefore, in this case, the size specified for the datafile must be larger (at least one block plus space for the bitmap) than 1MB.

Specifying Segment Space Management in Locally Managed Tablespaces

When you create a locally managed tablespace using the `CREATE TABLESPACE` statement, the `SEGMENT SPACE MANAGEMENT` clause allows you to specify how free and used space within a segment is to be managed. Your choices are:

- `MANUAL`

Specifying `MANUAL` tells Oracle that you want to use free lists for managing free space within segments. Free lists are lists of data blocks that have space available for inserting rows. This form of managing space within segments is called manual segment-space management because of the need to specify and tune the `PCTUSED`, `FREELISTS`, and `FREELISTS GROUPS` storage parameters for schema objects created in the tablespace.

`MANUAL` is the default.

- `AUTO`

This keyword tells Oracle that you want to use bitmaps to manage the free space within segments. A bitmap, in this case, is a map that describes the status of each data block within a segment with respect to the amount of space in the block available for inserting rows. As more or less space becomes available in a

data block, its new state is reflected in the bitmap. Bitmaps allow Oracle to manage free space more automatically, and thus, this form of space management is called automatic segment-space management.

Automatic segment-space management is a simpler and more efficient way of managing space within a segment. It completely eliminates any need to specify and tune the `PCTUSED`, `FREELISTS`, and `FREELISTS GROUPS` storage parameters for schema objects created in the tablespace. If such attributes should be specified, they are ignored.

Automatic segment-space management delivers better space utilization than manual segment-space management, and it is self tuning in that it scales with increasing the number of users, as well as instances. For a Real Application Clusters environment, automatic segment-space management allows for a dynamic affinity of space to instances, thus avoiding the hard partitioning of space inherent with using free list groups.

For many standard workloads, application performance when using automatic segment space management is better than the performance of a well tuned application using manual segment-space management.

The following statement creates tablespace `lmtbsb` with automatic segment-space management:

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
    EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO;
```

Your specification at tablespace creation time of your method for managing available space in segments, applies to all segments subsequently created in the tablespace. Also, your choice of method cannot be subsequently altered. Only permanent, locally managed tablespaces can specify automatic segment-space management.

Altering a Locally Managed Tablespace

You cannot alter a locally managed tablespace to a locally managed temporary tablespace, nor can you change its method of segment space management.

Some reasons for using the `ALTER TABLESPACE` statement for locally managed tablespaces include:

- Adding a datafile. For example:

```
ALTER TABLESPACE lmtbsb
    ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- Altering a tablespace's availability (ONLINE/OFFLINE). See "[Altering Tablespace Availability](#)" on page 11-21.
- Making a tablespace read-only or read-write. See "[Using Read-Only Tablespaces](#)" on page 11-24.
- Renaming a datafile, or enabling/disabling the autoextension of the size of a datafile in the tablespace. See [Chapter 12, "Managing Datafiles"](#).

Coalescing free extents is unnecessary for locally managed tablespaces.

Dictionary-Managed Tablespaces

Starting with Oracle9i, the default for extent management when creating a tablespace is locally managed. However, you can explicitly specify that you want to create a dictionary-managed tablespace. For dictionary-managed tablespaces, Oracle updates the appropriate tables in the data dictionary whenever an extent is allocated, or freed for reuse.

Creating a Dictionary-Managed Tablespace

As an example, the following statement creates the tablespace `tbsa`, with the following characteristics:

- The data of the new tablespace is contained in a single datafile, 50M in size.
- The tablespace is explicitly created as a dictionary-managed tablespace by specifying `EXTENT MANAGEMENT DICTIONARY`.
- The default storage parameters for any segments created in this tablespace are specified.

The following statement creates the tablespace `tbsb`:

```
CREATE TABLESPACE tbsb
  DATAFILE '/u02/oracle/data/tbsa01.dbf' SIZE 50M
  EXTENT MANAGEMENT DICTIONARY
  DEFAULT STORAGE (
    INITIAL 50K
    NEXT 50K
    MINEXTENTS 2
    MAXEXTENTS 50
    PCTINCREASE 0);
```

The following parameters, included in the above example, determine segment storage allocation in the tablespace. These parameters affect both how long it takes

to access data stored in the database and how efficiently space in the database is used. They are referred to as storage parameters.

Storage Parameter	Description
INITIAL	Defines the size in bytes (K or M) of the first extent in the segment
NEXT	Defines the size of the second extent in bytes (K or M)
PCTINCREASE	Specifies the percent by which each extent, after the second (NEXT) extent, grows
MINEXTENTS	Specifies the number of extents allocated when a segment is first created in the tablespace
MAXEXTENTS	Determines the maximum number of extents that a segment can have. Can also be specified as UNLIMITED.

Another parameter on the `CREATE TABLESPACE` statement, `MIMUMUM EXTENT`, also influences segment allocation. If specified, it ensures that all free and allocated extents in the tablespace are at least as large as, and a multiple of, a specified number of bytes (K or M). This provides one means of controlling free space fragmentation in the tablespace.

See Also:

- ["Setting Storage Parameters"](#) on page 14-8
- *Oracle9i Database Performance Tuning Guide and Reference* for more discussion of the effects of these parameters
- *Oracle9i SQL Reference* for a complete description of storage parameters

Altering a Dictionary-Managed Tablespace

One reason for using an `ALTER TABLESPACE` statement is to add a datafile. The following statement creates a new datafile for the `tbsa` tablespace:

```
ALTER TABLESPACE tbsa
  ADD DATAFILE '/u02/oracle/data/tbsa02.dbf' SIZE 1M;
```

You might also want to change the default storage parameters.

You can change the default storage parameters of a tablespace using the `ALTER TABLESPACE` statement, as illustrated in the following example:

```
ALTER TABLESPACE users
  DEFAULT STORAGE (
    NEXT 100K
    MAXEXTENTS 20
    PCTINCREASE 0);
```

New values for the default storage parameters of a tablespace affect only future objects that are created, or extents allocated for existing segments within the tablespace.

Other reasons for issuing an `ALTER TABLESPACE` statement include, but are not limited to:

- Coalescing free space in a tablespace. See ["Coalescing Free Space in Dictionary-Managed Tablespaces"](#) on page 11-16.
- Altering a tablespace's availability (`ONLINE/OFFLINE`). See ["Altering Tablespace Availability"](#) on page 11-21.
- Making a tablespace read-only or read-write. See ["Using Read-Only Tablespaces"](#) on page 11-24.
- Adding or renaming a datafile, or enabling/disabling the autoextension of the size of a datafile in the tablespace. See [Chapter 12, "Managing Datafiles"](#).

Temporary Tablespaces

To improve the concurrence of multiple sort operations, reduce their overhead, or avoid Oracle space management operations altogether, create **temporary tablespaces**. A temporary tablespace can be shared by multiple users and can be assigned to users with the `CREATE USER` statement when you create users in the database.

Within a temporary tablespace, all sort operations for a given instance and tablespace share a single **sort segment**. Sort segments exist for every instance that performs sort operations within a given tablespace. The sort segment is created by the first statement that uses a temporary tablespace for sorting, after startup, and is released only at shutdown. An extent cannot be shared by multiple transactions.

You can view the allocation and deallocation of space in a temporary tablespace sort segment using the `V$SORT_SEGMENT` view. The `V$TEMPSEG_USAGE` view identifies the current sort users in those segments.

You cannot explicitly create objects in a temporary tablespace.

See Also:

- [Chapter 24, "Managing Users and Resources"](#) for information about assigning temporary tablespaces to users
- *Oracle9i Database Reference* for more information about the V\$SORT_SEGMENT and V\$TEMPSEG_USAGE views
- *Oracle9i Database Performance Tuning Guide and Reference* for a discussion on tuning sorts

Creating a Locally Managed Temporary Tablespace

Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces. Locally managed temporary tablespaces use **tempfiles**, which do not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data. Therefore, they can be used in standby or read-only databases.

You also use different views for viewing information about tempfiles than you would for datafiles. The V\$TEMPFILE and DBA_TEMP_FILES views are analogous to the V\$DATAFILE and DBA_DATA_FILES views.

To create a locally managed temporary tablespace, you use the CREATE TEMPORARY TABLESPACE statement, which requires that you have the CREATE TABLESPACE system privilege.

The following statement creates a temporary tablespace in which each extent is 16M. Each 16M extent (which is the equivalent of 8000 blocks when the standard block size is 2K) is represented by a bit in the bitmap for the file.

```
CREATE TEMPORARY TABLESPACE lmtemp TEMPFILE '/u02/oracle/data/lmtemp01.dbf'
    SIZE 20M REUSE
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

The extent management clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The Oracle default for SIZE is 1M. But if you want to specify another value for SIZE, you can do so as shown in the above statement.

The AUTOALLOCATE clause is not allowed for temporary tablespaces.

Note: On some operating systems, Oracle does not allocate space for the tempfile until the tempfile blocks are actually accessed. This delay in space allocation results in faster creation and resizing of tempfiles, but it requires that sufficient disk space is available when the tempfiles are later used. Please refer to your operating system documentation to determine whether Oracle allocates tempfile space in this way on your system.

Altering a Locally Managed Temporary Tablespace

Except for adding a tempfile, as illustrated in the following example, you cannot use the `ALTER TABLESPACE` statement for a locally managed temporary tablespace.

```
ALTER TABLESPACE ltemp
  ADD TEMPFILE '/u02/oracle/data/ltemp02.dbf' SIZE 2M REUSE;
```

Note: You cannot use the `ALTER TABLESPACE` statement, with the `TEMPORARY` keyword, to change a locally managed permanent tablespace into a locally managed temporary tablespace. You must use the `CREATE TEMPORARY TABLESPACE` statement to create a locally managed temporary tablespace.

However, the `ALTER DATABASE` statement can be used to alter tempfiles.

The following statements take offline and bring online temporary files:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/ltemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/ltemp02.dbf' ONLINE;
```

The following statement resizes a temporary file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/ltemp02.dbf' RESIZE 4M;
```

The following statement drops a temporary file and deletes the operating system file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/ltemp02.dbf' DROP
  INCLUDING DATAFILES;
```

The tablespace to which this tempfile belonged remains. A message is written to the alert file for the datafile that was deleted. If an operating system error prevents the deletion of the file, the statement still succeeds, but a message describing the error is written to the alert file.

It is also possible, but not shown, to use the `ALTER DATABASE` statement to enable or disable the automatic extension of an existing tempfile, and to rename (`RENAME FILE`) a tempfile.

Creating a Dictionary-Managed Temporary Tablespace

To identify a tablespace as temporary during tablespace creation, specify the `TEMPORARY` keyword on the `CREATE TABLESPACE` statement. You cannot specify `EXTENT MANAGEMENT LOCAL` for a temporary tablespace created in this fashion. To create a locally managed temporary tablespace, use the `CREATE TEMPORARY TABLESPACE` statement, which is the preferred method of creating a temporary tablespace.

The following statement creates a temporary dictionary-managed tablespace:

```
CREATE TABLESPACE sort
  DATAFILE '/u02/oracle/data/sort01.dbf' SIZE 50M
  DEFAULT STORAGE (
    INITIAL 2M
    NEXT 2M
    MINEXTENTS 1
    PCTINCREASE 0)
  EXTENT MANAGEMENT DICTIONARY
  TEMPORARY;
```

Altering a Dictionary-Managed Temporary Tablespace

You can issue the `ALTER TABLESPACE` statement against a dictionary-managed temporary tablespace using many of the same keywords and clauses as for a permanent dictionary-managed tablespace. Any restrictions are noted in the *Oracle9i SQL Reference*.

Note: When you take dictionary-managed temporary tablespaces offline with the `ALTER TABLESPACE . . . OFFLINE` statement, returning them online does not affect their temporary status.

You can change an existing permanent dictionary-managed tablespace to a temporary tablespace, using the `ALTER TABLESPACE` statement. For example:

```
ALTER TABLESPACE tbsa TEMPORARY;
```

Coalescing Free Space in Dictionary-Managed Tablespaces

Over time, the free space in a dictionary-managed tablespace can become fragmented, making it difficult to allocate new extents. Ways of defragmenting this free space are discussed in this section.

The following topics are contained in this section:

- [How Oracle Coalesces Free Space](#)
- [Manually Coalescing Free Space](#)
- [Monitoring Free Space](#)

How Oracle Coalesces Free Space

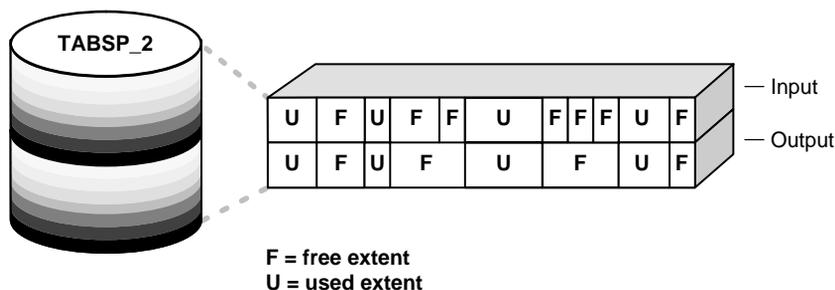
A free extent in a dictionary-managed tablespace is comprised of a collection of contiguous free blocks. When allocating new extents to a tablespace segment, the free extent closest in size to the required extent is used. In some cases, when segments are dropped, their extents are deallocated and marked as free, but any adjacent free extents are not immediately recombined into larger free extents. The result is fragmentation that makes allocation of larger extents more difficult.

This fragmentation is addressed in several ways:

- When attempting to allocate a new extent for a segment, Oracle first tries to find a free extent large enough for the new extent. If no free extent that is large enough is found, Oracle then coalesces adjacent free extents in the tablespace and looks again. This coalescing is *always* performed by Oracle whenever it cannot find a free extent into which the new extent will fit.
- The SMON background process periodically coalesces neighboring free extents when the `PCTINCREASE` value for a tablespace is nonzero. If you set `PCTINCREASE=0`, no coalescing of free extents will occur. If you are concerned about the overhead of SMON's ongoing coalescing, an alternative is to set `PCTINCREASE=0`, and periodically coalesce free space manually.
- When a segment is dropped or truncated, a limited form of coalescing is performed if the `PCTINCREASE` value for the segment is not zero. This is done even if `PCTINCREASE=0` for the tablespace containing the segment.
- You can use the `ALTER TABLESPACE ... COALESCE` statement to manually coalesce any adjacent free extents.

The process of coalescing free space is illustrated in the following figure.

Figure 11–1 Coalescing Free Space



Note: Coalescing free space is not necessary for locally managed tablespaces because bitmaps automatically track adjacent free space.

See Also: *Oracle9i Database Concepts* for detailed information on allocating extents and coalescing free space

Manually Coalescing Free Space

If you find that fragmentation of space in a tablespace is high (contiguous space on your disk appears as noncontiguous), you can coalesce any free space using the `ALTER TABLESPACE . . . COALESCE` statement. You must have the `ALTER TABLESPACE` system privilege to coalesce tablespaces.

You might want to use this statement if `PCTINCREASE=0`, or you can use it to supplement SMON and extent allocation coalescing. If all extents within the tablespace are of the same size, coalescing is not necessary. This would be the case if the default `PCTINCREASE` value for the tablespace were set to zero, all segments used the default storage parameters of the tablespace, and `INITIAL=NEXT=MINIMUM EXTENT`.

The following statement coalesces free space in the tablespace `tabsp_4`:

```
ALTER TABLESPACE tabsp_4 COALESCE;
```

Like other options of the `ALTER TABLESPACE` statement, the `COALESCE` option is exclusive: when specified, it must be the only option.

This statement does not coalesce free extents that are separated by data extents. If you observe that there are many free extents located between data extents, you must reorganize the tablespace (for example, by exporting and importing its data) to create useful free space extents.

Monitoring Free Space

You can use the following views for monitoring free space in a tablespace:

- `DBA_FREE_SPACE`
- `DBA_FREE_SPACE_COALESCED`

The following statement displays the free space in tablespace `tabsp_4`:

```
SELECT BLOCK_ID, BYTES, BLOCKS
       FROM DBA_FREE_SPACE
       WHERE TABLESPACE_NAME = 'TABSP_4'
       ORDER BY BLOCK_ID;
```

BLOCK_ID	BYTES	BLOCKS
2	16384	2
4	16384	2
6	81920	10
16	16384	2
27	16384	2
29	16384	2
31	16384	2
33	16384	2
35	16384	2
37	16384	2
39	8192	1
40	8192	1
41	196608	24

13 rows selected.

This view shows that there is adjacent free space in `tabsp_4` (for example, blocks starting with `BLOCK_ID`s 2, 4, 6, 16) that has not been coalesced. After coalescing the tablespace using the `ALTER TABLESPACE` statement shown previously, the results of this query would read:

```
BLOCK_ID  BYTES      BLOCKS
```

```

-----
          2      131072      16
          27      311296      38
2 rows selected.

```

The `DBA_FREE_SPACE_COALESCED` view displays statistics for coalescing activity. It is also useful in determining if you need to coalesce space.

See Also: *Oracle9i Database Reference* for more information about these views

Specifying Nonstandard Block Sizes for Tablespaces

You can create tablespaces of different block sizes than the standard database block size specified by the `DB_BLOCK_SIZE` initialization parameter. This feature enables the transporting of tablespaces with unlike block sizes between databases.

The `BLOCKSIZE` clause of the `CREATE TABLESPACE` statement enables you to create a tablespace with a block size other than the database's standard block size. However, your buffer cache in SGA memory must be configured for the nonstandard block sizes.

The following statement creates tablespace `lmtbsb`, but specifies a block size that differs from the standard database block size (as specified by the `DB_BLOCK_SIZE` initialization parameter):

```

CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
BLOCKSIZE 8K;

```

Note: In order for the `BLOCKSIZE` clause to succeed, you must have the `DB_CACHE_SIZE` and at least one `DB_nK_CACHE_SIZE` initialization parameter set, and the integer you specify in this clause must correspond with the setting of one `DB_nK_CACHE_SIZE` parameter setting. Although redundant, specifying a `BLOCKSIZE` equal to the standard block size, as specified by the `DB_BLOCK_SIZE` initialization parameter, is allowed.

For information about these parameters, see "[Setting the Buffer Cache Initialization Parameters](#)" on page 2-40.

See Also:

- ["Specifying Database Block Sizes"](#) on page 2-37
- ["Transporting Tablespaces Between Databases"](#) on page 11-34

Controlling the Writing of Redo Records

For some database operations, it is possible to control whether redo records are generated. Suppressing redo generation can improve performance, and may be appropriate for easily recoverable operations. This might include a `CREATE TABLE . . . AS SELECT` statement, where the operation can be repeated if there is a database or instance failure. Without redo, no media recovery is possible.

Specify the `NOLOGGING` clause in the `CREATE TABLESPACE` statement if you wish to suppress redo when these operations are performed for objects within the tablespace. If you do not include this clause, or specify `LOGGING` instead, then redo is generated when changes are made to objects in the tablespace. Redo is never generated for temporary segments or in temporary tablespaces, regardless of the logging attribute.

The logging attribute specified at the tablespace level is the default attribute for objects created within the tablespace. You can override this default logging attribute by specifying `LOGGING` or `NOLOGGING` at the schema object level; for example, in a `CREATE TABLE` statement.

In the case where you have a standby database, specifying `NOLOGGING` causes problems with the availability and accuracy of the standby database. To overcome this problem, you can specify `FORCE LOGGING` mode. When you include the `FORCE LOGGING` clause in the `CREATE TABLESPACE` statement, you force the generation of redo records for all operations that make changes to objects in a tablespace. This overrides any specification made at the object level.

If you transport a tablespace that is in `FORCE LOGGING` mode to another database, the new tablespace will not maintain the `FORCE LOGGING` mode.

See Also:

- *Oracle9i Database Concepts* for additional information about NOLOGGING mode
- *Oracle9i SQL Reference* for information about operations that can be done in NOLOGGING mode
- ["Specifying FORCE LOGGING Mode"](#) on page 2-29 for more information about FORCE LOGGING mode and for information about the effects of the FORCE LOGGING clause used with the CREATE DATABASE statement

Altering Tablespace Availability

You can take an online tablespace offline so that this portion of the database is temporarily unavailable for general use. The rest of the database is open and available for users to access data. Conversely, you can bring an offline tablespace online to make the schema objects within the tablespace available to database users. The database must be open.

To alter the availability of a tablespace, use the SQL statement ALTER TABLESPACE. You must have the ALTER TABLESPACE or MANAGE TABLESPACE system privilege to perform this action.

You can also take all of the datafiles or tempfiles in a tablespace offline, and bring them back online, without affecting the OFFLINE or ONLINE status of the tablespace itself.

Taking Tablespaces Offline

You may want to take a tablespace offline for any of the following reasons:

- To make a portion of the database unavailable while allowing normal access to the remainder of the database
- To perform an offline tablespace backup (even though a tablespace can be backed up while online and in use)
- To make an application and its group of tables temporarily unavailable while updating or maintaining the application

When a tablespace is taken offline, Oracle takes all the associated files offline. The SYSTEM tablespace can never be taken offline.

You can specify any of the following options when taking a tablespace offline:

Option	Description
NORMAL	A tablespace can be taken offline normally if no error conditions exist for any of the datafiles of the tablespace. No datafile in the tablespace can be currently offline as the result of a write error. When <code>OFFLINE NORMAL</code> is specified, Oracle takes a checkpoint for all datafiles of the tablespace as it takes them offline. <code>NORMAL</code> is the default.
TEMPORARY	A tablespace can be taken offline temporarily, even if there are error conditions for one or more files of the tablespace. When <code>OFFLINE TEMPORARY</code> is specified, Oracle takes offline the datafiles that are not already offline, checkpointing them as it does so. If no files are offline, but you use the temporary option, media recovery is not required to bring the tablespace back online. However, if one or more files of the tablespace are offline because of write errors, and you take the tablespace offline temporarily, the tablespace requires recovery before you can bring it back online.
IMMEDIATE	A tablespace can be taken offline immediately, without Oracle taking a checkpoint on any of the datafiles. When you specify <code>OFFLINE IMMEDIATE</code> , media recovery for the tablespace is required before the tablespace can be brought online. You cannot take a tablespace offline immediately if the database is running in <code>NOARCHIVELOG</code> mode.
FOR RECOVER	Takes the database tablespaces in the recovery set offline for tablespace point-in-time recovery. For additional information, see <i>Oracle9i User-Managed Backup and Recovery Guide</i> .

Caution: If you must take a tablespace offline, use the `NORMAL` option (the default) if possible. This guarantees that the tablespace will not require recovery to come back online. It will not require recovery, even if after incomplete recovery you reset the redo log sequence using an `ALTER DATABASE OPEN RESETLOGS` statement.

Specify `TEMPORARY` only when you cannot take the tablespace offline normally. In this case, only the files taken offline because of errors need to be recovered before the tablespace can be brought online. Specify `IMMEDIATE` only after trying both the normal and temporary options.

The following example takes the `users` tablespace offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

Before taking an online tablespace offline, consider taking the following actions:

- Verify that the tablespace contains no active rollback segments. Such a tablespace cannot be taken offline.
- You may want to alter the tablespace allocation of any users who have been assigned the tablespace as either a default or temporary tablespace. This is advisable because they will not be able to access objects or sort areas in the tablespace while it is offline.

See Also: ["Taking Rollback Segments Offline"](#) on page 13-23

Bringing Tablespaces Online

You can bring any tablespace in an Oracle database online whenever the database is open. A tablespace is normally online so that the data contained within it is available to database users.

Note: If a tablespace to be brought online was not taken offline "cleanly" (that is, using the `NORMAL` option of the `ALTER TABLESPACE OFFLINE` statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, Oracle returns an error and the tablespace remains offline.

Depending upon your archiving strategy, refer to one of the following books for information about performing media recovery:

- *Oracle9i User-Managed Backup and Recovery Guide*
 - *Oracle9i Recovery Manager User's Guide*
-
-

The following statement brings the `users` tablespace online:

```
ALTER TABLESPACE users ONLINE;
```

Altering the Availability of Datafiles or Tempfiles

Clauses of the `ALTER TABLESPACE` statement enable you to change the online or offline status of all of the datafiles or tempfiles within a tablespace. Specifically, the statements that affect online/offline status are:

- `ALTER TABLESPACE ... DATAFILE {ONLINE|OFFLINE}`

- `ALTER TABLESPACE ... TEMPFILE {ONLINE|OFFLINE}`

You are required only to enter the tablespace name, not the individual datafiles or tempfiles. All of the datafiles or tempfiles are affected, but the online/offline status of the tablespace itself is not changed.

In most cases the above `ALTER TABLESPACE` statements can be issued whenever the database is mounted, even if it is not open. The database *must not* be open if the tablespace is the `SYSTEM` tablespace, an undo tablespace, or the default temporary tablespace. The `ALTER DATABASE DATAFILE` and `ALTER DATABASE TEMPFILE` statements also have `ONLINE/OFFLINE` clauses, however in those statements require that you enter all of the filenames for the tablespace.

The syntax is different from the `ALTER TABLESPACE ... ONLINE|OFFLINE` statement that alters a tablespace's availability, because that is a different operation. The `ALTER TABLESPACE` statement takes datafiles offline as well as the tablespace, but it cannot be used to alter the status of a temporary tablespace or its tempfile(s).

Using Read-Only Tablespaces

Making a tablespace read-only prevents write operations on the datafiles in the tablespace. The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database, but they also provide a means of completely protecting historical data so that no one can modify the data after the fact. Making a tablespace read-only prevents updates on all tables in the tablespace, regardless of a user's update privilege level.

Note: Making a tablespace read-only cannot in itself be used to satisfy archiving or data publishing requirements, because the tablespace can only be brought online in the database in which it was created. However, you can meet such requirements by using the transportable tablespace feature.

You can drop items, such as tables or indexes, from a read-only tablespace, but you cannot create or alter objects in the tablespace. You can execute statements that update the file description in the data dictionary, such as `ALTER TABLE ... ADD` or `ALTER TABLE ... MODIFY`, but you will not be able to utilize the new description until the tablespace is made read-write.

Read-only tablespaces can be transported to other databases. And, since read-only tablespaces can never be updated, they can reside on CD-ROM or WORM (Write Once-Read Many) devices.

The following topics are discussed in this section:

- [Making a Tablespace Read-Only](#)
- [Making a Read-Only Tablespace Writable](#)
- [Creating a Read-Only Tablespace on a WORM Device](#)
- [Delaying the Opening of Datafiles in Read Only Tablespaces](#)

See Also:

- *Oracle9i Database Concepts* for more information about read-only tablespaces
- ["Transporting Tablespaces Between Databases"](#) on page 11-34

Making a Tablespace Read-Only

All tablespaces are initially created as read-write. Use the `READ ONLY` clause in the `ALTER TABLESPACE` statement to change a tablespace to read-only. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

Before you can make a tablespace read-only, the following conditions must be met.

- The tablespace must be online.
This is necessary to ensure that there is no undo information that needs to be applied to the tablespace.
- The tablespace must not contain any active rollback segments (this would be the normal situation, as a data tablespace should not contain rollback segments).
For this reason, the `SYSTEM` tablespace can never be made read-only, since it contains the `SYSTEM` rollback segment. Additionally, because any rollback segments of a read-only tablespace would not be accessible, you would have to drop the rollback segments before you made a tablespace read-only.
- The tablespace must not currently be involved in an online backup, since the end of a backup updates the header file of all datafiles in the tablespace.

For better performance while accessing data in a read-only tablespace, you can issue a query that accesses all of the blocks of the tables in the tablespace just before making it read-only. A simple query, such as `SELECT COUNT (*)`, executed against each table ensures that the data blocks in the tablespace can be subsequently accessed most efficiently. This eliminates the need for Oracle to check the status of the transactions that most recently modified the blocks.

The following statement makes the `flights` tablespace read-only:

```
ALTER TABLESPACE flights READ ONLY;
```

You do not have to wait for transactions to complete before issuing the `ALTER TABLESPACE ... READ ONLY` statement. When the statement is issued, the target tablespace goes into a transitional read-only mode in which no further write operations (DML statements) are allowed against the tablespace. Existing transactions that modified the tablespace are allowed to commit or rollback. Once all transactions (in the database) have completed, the tablespace becomes read-only.

Note: This transitional read-only state only occurs if the value of the initialization parameter `COMPATIBLE` is 8.1.0 or greater. If this parameter is set to a value less than 8.1.0, the `ALTER TABLESPACE ... READ ONLY` statement fails if any active transactions exist.

If you find it is taking a long time for the tablespace to quiesce, it is possible to identify the transactions which are preventing the read-only state from taking effect. The owners of these transactions can be notified and a decision can be made to terminate the transactions, if necessary. The following example illustrates how you might identify the blocking transactions:

- Identify the transaction entry for the `ALTER TABLESPACE ... READ ONLY` statement and note its session address (`saddr`).

```
SELECT SQL_TEXT, SADDR
       FROM V$SQLAREA, V$SESSION
       WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS
             AND SQL_TEXT LIKE 'alter tablespace%';
```

SQL_TEXT	SADDR
alter tablespace tbs1 read only	80034AF0

- The start SCN of each active transaction is stored in the `V$TRANSACTION` view. Displaying this view sorted by ascending start SCN lists the transactions in execution order. Since you know the session address of the transaction entry for the read-only statement, it can be located in the `V$TRANSACTION` view. All transactions with lesser start SCN can potentially hold up the quiesce and subsequent read-only state of the tablespace.

```
SELECT SES_ADDR, START_SCNB
       FROM V$TRANSACTION
```

```
ORDER BY START_SCNB;
```

```
SES_ADDR START_SCNB
-----
800352A0      3621 --> waiting on this txn
80035A50      3623 --> waiting on this txn
80034AF0      3628 --> this is the ALTER TABLESPACE statement
80037910      3629 --> don't care about this txn
```

After making the tablespace read-only, it is advisable to back it up immediately. As long as the tablespace remains read-only, no further backups of the tablespace are necessary since no changes can be made to it.

See Also: Depending upon your backup and recovery strategy, refer to one of the following books for information about recovering a database with read-only datafiles:

- *Oracle9i User-Managed Backup and Recovery Guide*
- *Oracle9i Recovery Manager User's Guide*

Making a Read-Only Tablespace Writable

Use the `READ WRITE` keywords in the `ALTER TABLESPACE` statement to change a tablespace to allow write operations. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

A prerequisite to making the tablespace read-write is that all of the datafiles in the tablespace, as well as the tablespace itself, must be online. Use the `DATAFILE . . . ONLINE` clause of the `ALTER DATABASE` statement to bring a datafile online. The `V$DATAFILE` view lists the current status of datafiles.

The following statement makes the `flights` tablespace writable:

```
ALTER TABLESPACE flights READ WRITE;
```

Making a read-only tablespace writable updates the control file entry for the datafiles, so that you can use the read-only version of the datafiles as a starting point for recovery.

Creating a Read-Only Tablespace on a WORM Device

Follow these steps to create a read-only tablespace on a CD-ROM or WORM (Write Once-Read Many) device.

1. Create a writable tablespace on another device. Create the objects that belong in the tablespace and insert your data.
2. Alter the tablespace to make it read-only.
3. Copy the datafiles of the tablespace onto the WORM device. Use operating system commands to copy the files.
4. Take the tablespace offline.
5. Rename the datafiles to coincide with the names of the datafiles you copied onto your WORM device. Use `ALTER TABLESPACE` with the `RENAME DATAFILE` clause. Renaming the datafiles changes their names in the control file.
6. Bring the tablespace back online.

Delaying the Opening of Datafiles in Read Only Tablespaces

When substantial portions of a very large database are stored in read-only tablespaces that are located on slow-access devices or hierarchical storage, you should consider setting the `READ_ONLY_OPEN_DELAYED` initialization parameter to `TRUE`. This speeds certain operations, primarily opening the database, by causing datafiles in read-only tablespaces to be accessed for the first time only when an attempt is made to read data stored within them.

Setting `READ_ONLY_OPEN_DELAYED=TRUE` has the following side-effects:

- A missing or bad read-only file is not detected at open time. It is only discovered when there is an attempt to access it.
- `ALTER SYSTEM CHECK DATAFILES` does not check read-only files.
- `ALTER TABLESPACE ... ONLINE` and `ALTER DATABASE DATAFILE ... ONLINE` does not check read-only files. They are checked only upon the first access.
- `V$RECOVER_FILE`, `V$BACKUP`, and `V$DATAFILE_HEADER` do not access read-only files. Read-only files are indicated in the results list with the error "DELAYED OPEN", with zeroes for the values of other columns.
- `V$DATAFILE` does not access read-only files. Read-only files have a size of "0" listed.
- `V$RECOVER_LOG` does not access read-only files. Logs they could need for recovery are not added to the list.

- `ALTER DATABASE NOARCHIVELOG` does not access read-only files. It proceeds even if there is a read-only file that requires recovery.

Notes:

- `RECOVER DATABASE` and `ALTER DATABASE OPEN RESETLOGS` continue to access all read-only datafiles regardless of the parameter value. If you want to avoid accessing read-only files for these operations, those files should be taken offline.
 - If a backup control file is used, the read-only status of some files may be inaccurate. This can cause some of these operations to return unexpected results. Care should be taken in this situation.
-
-

Dropping Tablespaces

You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required. Any tablespace in an Oracle database, except the `SYSTEM` tablespace, can be dropped. You must have the `DROP TABLESPACE` system privilege to drop a tablespace.

Caution: Once a tablespace has been dropped, the tablespace's data is not recoverable. Therefore, make sure that all data contained in a tablespace to be dropped will not be required in the future. Also, immediately before and after dropping a tablespace from a database, back up the database completely. This is *strongly recommended* so that you can recover the database if you mistakenly drop a tablespace, or if the database experiences a problem in the future after the tablespace has been dropped.

When you drop a tablespace, the file pointers in the control file of the associated database are removed. You can optionally direct Oracle to delete the operating system files (datafiles) that constituted the dropped tablespace. If you do not direct Oracle to delete the datafiles at the same time that it deletes the tablespace, you must later use the appropriate commands of your operating system to delete them.

You cannot drop a tablespace that contains any active segments. For example, if a table in the tablespace is currently being used or the tablespace contains an active rollback segment, you cannot drop the tablespace. The tablespace can be online or offline, but it is best to take the tablespace offline before dropping it.

To drop a tablespace, use the `DROP TABLESPACE` statement. The following statement drops the `users` tablespace, including the segments in the tablespace:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

If the tablespace is empty (does not contain any tables, views, or other structures), you do not need to specify the `INCLUDING CONTENTS` option. Use the `CASCADE CONSTRAINTS` option to drop all referential integrity constraints from tables outside the tablespace that refer to primary and unique keys of tables inside the tablespace.

To delete the datafiles associated with a tablespace at the same time that the tablespace is dropped, use the `INCLUDING CONTENTS AND DATAFILES` clause. The following statement drops the `USER` tablespace and its associated datafiles:

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

A message is written to the alert file for each datafile that is deleted. If an operating system error prevents the deletion of a file, the `DROP TABLESPACE` statement still succeeds, but a message describing the error is written to the alert file.

Diagnosing and Repairing Locally Managed Tablespace Problems

Note: The `DBMS_SPACE_ADMIN` package provides administrators with defect diagnosis and repair functionality for locally managed tablespaces. It cannot be used in this capacity for dictionary-managed tablespaces.

It also provides procedures for migrating from dictionary-managed tablespaces to locally managed tablespaces, and the reverse.

The `DBMS_SPACE_ADMIN` package contains the following procedures:

Procedure	Description
<code>SEGMENT_VERIFY</code>	Verifies the consistency of the extent map of the segment.

Procedure	Description
SEGMENT_CORRUPT	Marks the segment corrupt or valid so that appropriate error recovery can be done. Cannot be used for a locally managed SYSTEM tablespace.
SEGMENT_DROP_CORRUPT	Drops a segment currently marked corrupt (without reclaiming space). Cannot be used for a locally managed SYSTEM tablespace.
SEGMENT_DUMP	Dumps the segment header and extent map of a given segment.
TABLESPACE_VERIFY	Verifies that the bitmaps and extent maps for the segments in the tablespace are in sync.
TABLESPACE_REBUILD_BITMAPS	Rebuilds the appropriate bitmap. Cannot be used for a locally managed SYSTEM tablespace.
TABLESPACE_FIX_BITMAPS	Marks the appropriate data block address range (extent) as free or used in bitmap. Cannot be used for a locally managed SYSTEM tablespace.
TABLESPACE_REBUILD_QUOTAS	Rebuilds quotas for given tablespace.
TABLESPACE_MIGRATE_FROM_LOCAL	Migrates a locally managed tablespace to dictionary-managed tablespace. Cannot be used to migrate a locally managed SYSTEM tablespace to a dictionary-managed SYSTEM tablespace.
TABLESPACE_MIGRATE_TO_LOCAL	Migrates a tablespace from dictionary-managed format to locally managed format.
TABLESPACE_RELOCATE_BITMAPS	Relocates the bitmaps to the destination specified. Cannot be used for a locally managed system tablespace.
TABLESPACE_FIX_SEGMENT_STATES	Fixes the state of the segments in a tablespace in which migration was aborted.

The following scenarios describe typical situations in which you can use the DBMS_SPACE_ADMIN package to diagnose and resolve problems.

Note: Some of these procedures can result in lost and unrecoverable data if not used properly. You should work with Oracle Support Services if you have doubts about these procedures.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for details about the `DBMS_SPACE_ADMIN` package

Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap)

The `TABLESPACE_VERIFY` procedure discovers that a segment has allocated blocks that are marked free in the bitmap, but no overlap between segments is reported.

In this scenario, perform the following tasks:

1. Call the `SEGMENT_DUMP` procedure to dump the ranges that the administrator allocated to the segment.
2. For each range, call the `TABLESPACE_FIX_BITMAPS` procedure with the `TABLESPACE_EXTENT_MAKE_USED` option to mark the space as used.
3. Call `TABLESPACE_REBUILD_QUOTAS` to fix up quotas.

Scenario 2: Dropping a Corrupted Segment

You cannot drop a segment because the bitmap has segment blocks marked "free". The system has automatically marked the segment corrupted.

In this scenario, perform the following tasks:

1. Call the `SEGMENT_VERIFY` procedure with the `SEGMENT_VERIFY_EXTENTS_GLOBAL` option. If no overlaps are reported, then proceed with steps 2 through 5.
2. Call the `SEGMENT_DUMP` procedure to dump the DBA ranges allocated to the segment.
3. For each range, call `TABLESPACE_FIX_BITMAPS` with the `TABLESPACE_EXTENT_MAKE_FREE` option to mark the space as free.
4. Call `SEGMENT_DROP_CORRUPT` to drop the `SEG$` entry.
5. Call `TABLESPACE_REBUILD_QUOTAS` to fix up quotas.

Scenario 3: Fixing Bitmap Where Overlap is Reported

The `TABLESPACE_VERIFY` procedure reports some overlapping. Some of the real data must be sacrificed based on previous internal errors.

After choosing the object to be sacrificed, in this case say, table `t1`, perform the following tasks:

1. Make a list of all objects that `t1` overlaps.
2. Drop table `t1`. If necessary, follow up by calling the `SEGMENT_DROP_CORRUPT` procedure.
3. Call the `SEGMENT_VERIFY` procedure on all objects that `t1` overlapped. If necessary, call the `TABLESPACE_FIX_BITMAPS` procedure to mark appropriate bitmap blocks as used.
4. Rerun the `TABLESPACE_VERIFY` procedure to verify the problem is resolved.

Scenario 4: Correcting Media Corruption of Bitmap Blocks

A set of bitmap blocks has media corruption.

In this scenario, perform the following tasks:

1. Call the `TABLESPACE_REBUILD_BITMAPS` procedure, either on all bitmap blocks, or on a single block if only one is corrupt.
2. Call the `TABLESPACE_REBUILD_QUOTAS` procedure to rebuild quotas.
3. Call the `TABLESPACE_VERIFY` procedure to verify that the bitmaps are consistent.

Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace

You migrate a dictionary-managed tablespace to a locally managed tablespace. You use the `TABLESPACE_MIGRATE_TO_LOCAL` procedure.

Let us assume that the database block size is 2K, and the existing extent sizes in tablespace `tbs_1` are 10, 50, and 10,000 blocks (used, used, and free). The `MINIMUM_EXTENT` value is 20K (10 blocks). In this scenario, you allow the bitmap allocation unit to be chosen by the system. The value of 10 blocks is chosen, because it is the highest common denominator and does not exceed `MINIMUM_EXTENT`.

The statement to convert `tbs_1` to a locally managed tablespace is as follows:

```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

If you choose to specify a allocation unit size, it must be a factor of the unit size calculated by the system, otherwise an error message is issued.

Migrating the SYSTEM Tablespace to a Locally Managed Tablespace

Use the `DBMS_SPACE_ADMIN` package to migrate the `SYSTEM` tablespace from dictionary-managed to locally managed. The following statement performs the migration:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL('SYSTEM');
```

Before performing the migration the following conditions must be met:

- The database has a default temporary tablespace that is not `SYSTEM`.
- There are no rollback segments in dictionary-managed tablespaces.
- There is at least one online rollback segment in a locally managed tablespace, or if using automatic undo management, an undo tablespace is online.
- All tablespaces other than the tablespace containing the undo space (that is, the tablespace containing the rollback segment or the undo tablespace) are in read-only mode.
- There is a cold backup of the database.
- The system is in restricted mode.

All of these conditions, except for the cold backup, are enforced by the `TABLESPACE_MIGRATE_TO_LOCAL` procedure.

Note: After the `SYSTEM` tablespace is migrated to locally managed, any dictionary-managed tablespaces in the database cannot be made `READ WRITE`. If you want to be able to use the dictionary-managed tablespaces in `READ-WRITE` mode, Oracle recommends that you first migrate these tablespaces to locally managed before migrating the `SYSTEM` tablespace.

Transporting Tablespaces Between Databases

This section describes how to transport tablespaces between databases, and contains the following topics:

- [Introduction to Transportable Tablespaces](#)
- [Limitations](#)
- [Compatibility Considerations for Transportable Tablespaces](#)
- [Transporting Tablespaces Between Databases: A Procedure](#)

- [Object Behaviors](#)
- [Using Transportable Tablespaces](#)

Introduction to Transportable Tablespaces

Note: You must be using the Enterprise Edition of Oracle8i (or higher) to generate a transportable tablespace set. However, you can use any edition of Oracle8i (or higher) to plug a transportable tablespace set into an Oracle database.

See "[Compatibility Considerations for Transportable Tablespaces](#)" on page 11-36 for a discussion of database compatibility for transporting tablespaces across release levels.

You can use the **transportable tablespaces** feature to move a subset of an Oracle database and "plug" it in to another Oracle database, essentially moving tablespaces between the databases. The tablespaces being transported can be either dictionary managed or locally managed. Starting with Oracle9i, the transported tablespaces are not required to be of the same block size as the target database's standard block size. Transporting tablespaces is particularly useful for:

- Moving data from OLTP systems to data warehouse staging systems
- Updating data warehouses and data marts from staging systems
- Loading data marts from central data warehouses
- Archiving OLTP and data warehouse systems efficiently
- Data publishing to internal and external customers
- Performing Tablespace Point-in-Time Recovery (TSPITR)

Moving data using transportable tablespaces can be much faster than performing either an export/import or unload/load of the same data, because transporting a tablespace only requires the copying of datafiles and integrating the tablespace structural information. You can also use transportable tablespaces to move index data, thereby avoiding the index rebuilds you would have to perform when importing or loading table data.

See Also:

- *Oracle9i Database Concepts* for more details about transportable tablespaces and their use in data marts and data warehousing
- *Oracle9i Database Migration* for information about transportable tablespace compatibility issues between different Oracle releases
- *Oracle9i Recovery Manager User's Guide* for information about using RMAN to transport an RMAN tablespace backup into another database. Using this method, you are not required to make the tablespace in the original database read-only.

Limitations

Be aware of the following limitations as you plan for transportable tablespace use:

- The source and target database must be on the same hardware platform. For example, you can transport tablespaces between Sun Solaris Oracle databases, or you can transport tablespaces between Windows NT Oracle databases. However, you cannot transport a tablespace from a Sun Solaris Oracle database to an Windows NT Oracle database.
- The source and target database must use the same character set and national character set.
- You cannot transport a tablespace to a target database in which a tablespace with the same name already exists.
- Transportable tablespaces do not support:
 - Materialized views/replication
 - Function-based indexes
 - Scoped `REFS`
 - 8.0-compatible advanced queues with multiple recipients

Compatibility Considerations for Transportable Tablespaces

To use the transportable tablespaces feature, the `COMPATIBLE` initialization parameter for both the source and target databases must be set to 8.1 or higher. If the block size of any tablespace being transported is different from the standard block size for the target database, the `COMPATIBLE` initialization parameter must be set to 9.0 or higher for the target database. You are not required to be running the

same release of Oracle for both the source and target database. Oracle guarantees that the transportable tablespace set is compatible with the target database. If not, an error is signaled at the beginning of the plug-in operation.

It is always possible to transport a tablespace from a database running an *older* release of Oracle (starting with Oracle8i) to a database running a *newer* release of Oracle (for example, Oracle9i).

When creating a transportable tablespace set, Oracle computes the lowest compatibility level at which the target database must run. This is referred to as the compatibility level of the transportable set. When plugging the transportable set into a target database, Oracle signals an error if the compatibility level of the transportable set is greater than the compatibility level of the target database.

Transporting Tablespaces Between Databases: A Procedure

To move or copy a set of tablespaces, perform the following steps. These steps are illustrated more fully in succeeding sections that detail transporting tablespaces `sales_1` and `sales_2` between databases.

1. Pick a self-contained set of tablespaces.
2. Generate a transportable tablespace set.

A **transportable tablespace set** consists of datafiles for the set of tablespaces being transported and a file containing structural information for the set of tablespaces.

3. Transport the tablespace set.

Copy the datafiles and the export file to the target database. You can do this using any facility for copying flat files (for example, an operating system copy utility, ftp, or publishing on CDs).

4. Plug in the tablespace.

Invoke the Import utility to plug the set of tablespaces into the target database.

Step 1: Pick a Self-Contained Set of Tablespaces

There may be logical or physical dependencies between objects in the transportable set and those outside of the set. You can only transport a set of tablespaces that is self-contained. In this context "self-contained" means that there are no references from inside the set of tablespaces pointing outside of the tablespaces. Some examples of self contained tablespace violations are:

- An index inside the set of tablespaces is for a table outside of the set of tablespaces.

Note: It is not a violation if a corresponding index for a table is outside of the set of tablespaces.

- A partitioned table is partially contained in the set of tablespaces.

The tablespace set you want to copy must contain either all partitions of a partitioned table, or none of the partitions of a partitioned table. If you want to transport a subset of a partition table, you must exchange the partitions into tables.

- A referential integrity constraint points to a table across a set boundary.

When transporting a set of tablespaces, you can choose to include referential integrity constraints. However, doing so can affect whether or not a set of tablespaces is self-contained. If you decide not to transport constraints, then the constraints are not considered as pointers.

- A table inside the set of tablespaces contains a LOB column that points to LOBs outside the set of tablespaces.

To determine whether a set of tablespaces is self-contained, you can invoke the `TRANSPORT_SET_CHECK` procedure in the Oracle supplied package `DBMS_TTS`. You must have been granted the `EXECUTE_CATALOG_ROLE` role (initially signed to `SYS`) to execute this procedure.

When you invoke the `DBMS_TTS` package, you specify the list of tablespaces in the transportable set to be checked for self containment. You can optionally specify if constraints must be included. For **strict or full containment**, you must additionally set the `TTS_FULL_CHECK` parameter to `TRUE`.

The strict or full containment check is for cases that require capturing not only references going outside the transportable set, but also those coming into the set. Tablespace Point-in-Time Recovery (TSPITR) is one such case where dependent objects must be fully contained or fully outside the transportable set.

For example, it is a violation to perform TSPITR on a tablespace containing a table `t` but not its index `i` because the index and data will be inconsistent after the transport. A full containment check ensures that there are no dependencies going outside or coming into the transportable set. See the example for TSPITR in the *Oracle9i User-Managed Backup and Recovery Guide*.

Note: The default for transportable tablespaces is to check for self containment rather than full containment.

Here we determine whether tablespaces `sales_1` and `sales_2` are self-contained, with referential integrity constraints taken into consideration (indicated by `TRUE`).

```
EXECUTE dbms_tts.transport_set_check('sales_1,sales_2', TRUE);
```

After invoking this PL/SQL package, you can see all violations by selecting from the `TRANSPORT_SET_VIOLATIONS` view. If the set of tablespaces is self-contained, this view is empty. The following query shows a case where there are two violations: a foreign key constraint, `dept_fk`, across the tablespace set boundary, and a partitioned table, `jim.sales`, that is partially contained in the tablespace set.

```
SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

```
VIOLATIONS
```

```
-----
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table
JIM.DEPT in tablespace OTHER
Partitioned table JIM.SALES is partially contained in the transportable set
```

These violations must be resolved before `sales_1` and `sales_2` are transportable. As noted in the next step, one choice for bypassing the integrity constrain violation is to not export the integrity constraints.

Object references (such as `REFs`) across the tablespace set are not considered violations. `REFs` are not checked by the `TRANSPORT_SET_CHECK` routine. When a tablespace containing dangling `REFs` is plugged into a database, queries following that dangling `REF` indicate user error.

See Also:

- *Oracle9i Application Developer's Guide - Fundamentals* for more information about `REFs`
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_TTS` package
- *Oracle9i User-Managed Backup and Recovery Guide* for information specific to using the `DBMS_TTS` package for `TSPITR`

Step 2: Generate a Transportable Tablespace Set

After ensuring you have a self-contained set of tablespaces that you want to transport, generate a transportable tablespace set by performing the following tasks:

1. Make all tablespaces in the set you are copying read-only.

```
ALTER TABLESPACE sales_1 READ ONLY;  
ALTER TABLESPACE sales_2 READ ONLY;
```

2. Invoke the Export utility and specify which tablespaces are in the transportable set, as follows:

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=(sales_1,sales_2)  
TRIGGERS=y CONSTRAINTS=n GRANTS=n FILE=expdat.dmp
```

Note: Although the Export utility is used, only data dictionary structural information (metadata) for the tablespaces is exported. Hence, this operation goes quickly even for a large tablespace.

When prompted, connect as SYS (or other administrative user) with the SYSDBA system privilege :

```
CONNECT SYS/password AS SYSDBA
```

You must always specify TABLESPACES. In this example, we also specify that:

- Triggers are to be exported.
If you set TRIGGERS=y, triggers are exported without a validity check. Invalid triggers cause compilation errors during the subsequent import. If you set TRIGGERS=n, triggers are not exported.
- Referential integrity constraints are not to be exported
- Grants are not to be exported.
- The name of the structural information export file to be created is expdat.dmp.

If you are performing TSPITR or transport with a strict containment check, use:

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=(sales_1,sales_2)  
TTS_FULL_CHECK=Y FILE=expdat.dmp
```

If the tablespace sets being transported are not self-contained, export fails and indicate that the transportable set is not self-contained. You must then return to Step 1 to resolve all violations.

See Also: *Oracle9i Database Utilities* for information about using the Export utility

Step 3: Transport the Tablespace Set

Transport *both the datafiles and the export file* of the tablespaces to a place accessible to the target database. You can use any facility for copying flat files (for example, an operating system copy utility, ftp, or publishing on CDs).

Step 4: Plug In the Tablespace Set

Note: If you are transporting a tablespace of a different block size than the standard block size of the database receiving the tablespace set, then you must first have a `DB_nK_CACHE_SIZE` initialization parameter entry in the receiving database's parameter file.

For example, if you are transporting a tablespace with an 8K block size into a database with a 4K standard block size, then you must include a `DB_8K_CACHE_SIZE` initialization parameter entry in the parameter file. If it is not already included in the parameter file, this parameter can be set using the `ALTER SYSTEM SET` statement.

See *Oracle9i SQL Reference* for information about specifying values for the `DB_nK_CACHE_SIZE` initialization parameter.

To plug in a tablespace set, perform the following tasks:

1. Plug in the tablespaces and integrate the structural information using the Import utility.

```
IMP TRANSPORT_TABLESPACE=y FILE=expdat.dmp
  DATAFILES=(' /db/sales_jan', '/db/sales_feb', ...)
  TABLESPACES=(sales_1,sales_2) TTS_OWNERS=(dcranney,jfee)
  FROMUSER=(dcranney,jfee) TOUSER=(smith,williams)
```

When prompted, connect as `SYS` (or other administrative user) with the `SYSDBA` system privilege :

```
CONNECT SYS/password AS SYSDBA
```

In this example we specify the following:

- `TRANSPORT_TABLESPACE=y` tells the Export utility that we are transporting a tablespace.
- The exported file containing the metadata for the tablespaces is `expdat.dmp`.
- `DATAFILES` specifies the datafiles of the transported tablespaces and must be specified.
- The tablespace names are `sales_1` and `sales_2`.

When you specify `TABLESPACES`, the supplied tablespace names are compared to those in the export file. Import returns an error if there is any mismatch. Otherwise, tablespace names are extracted from the export file.

- `TTS_OWNERS` lists all users who own data in the tablespace set.

When you specify `TTS_OWNERS`, the user names are compared to those in the export file. Import returns an error if there is any mismatch. Otherwise, owner names are extracted from the export file.

- `FROMUSER` and `TOUSER` are specified to change the ownership of database objects.

If you do not specify `FROMUSER` and `TOUSER`, all database objects (such as tables and indexes) are created under the same user as in the source database. Those users must already exist in the target database. If not, import returns an error indicating that some required users do not exist in the target database.

You can use `FROMUSER` and `TOUSER` to change the owners of objects. In this example we specify `FROMUSER=(dcranney, jfee)` and `TOUSER=(smith, williams)`. Objects in the tablespace set owned by `dcranney` in the source database will be owned by `smith` in the target database after the tablespace set is plugged in. Similarly, objects owned by `jfee` in the source database will be owned by `williams` in the target database. In this case, the target database is not required to have users `dcranney` and `jfee`, but must have users `smith` and `williams`.

After this statement successfully executes, all tablespaces in the set being copied remain in read-only mode. Check the import logs to ensure no error has occurred.

When dealing with a large number of datafiles, specifying the list of datafile names in the statement line can be a laborious process. It can even exceed the statement line limit. In this situation, you can use an import parameter file. For example, you can invoke the Import utility as follows:

```
IMP PARFILE='par.f'
```

The file `par.f` file contains the following:

```
TRANSPORT_TABLESPACE=y
FILE=expdat.dmp
DATAFILES=(' /db/sales_jan', '/db/sales_feb', ...)
TABLESPACES=(sales_1,sales_2)
TTS_OWNERS=(dcranney,jfee)
FROMUSER=(dcranney,jfee)
TOUSER=(smith,williams)
```

2. If necessary, put the tablespaces in the copied space back into read-write mode as follows:

```
ALTER TABLESPACE sales_1 READ WRITE
ALTER TABLESPACE sales_1 READ WRITE
```

See Also: *Oracle9i Database Utilities* for information about using the Import utility

Object Behaviors

Most objects, whether data in a tablespace or structural information associated with the tablespace, behave normally after being transported to a different database. However, the following objects are exceptions:

- [ROWIDs](#)
- [REFs](#)
- [Privileges](#)
- [Partitioned Tables](#)
- [Objects](#)
- [Advanced Queues](#)
- [Indexes](#)
- [Triggers](#)

- [Materialized Views/Replication](#)

ROWIDs

When a database contains tablespaces that have been plugged in (from other databases), the ROWIDs in that database are no longer unique. A ROWID is guaranteed unique only within a table.

REFs

REFs are not checked when Oracle determines if a set of tablespaces is self-contained. As a result, a plugged-in tablespace may contain dangling REFs. Any query following dangling REFs returns a user error.

Privileges

Privileges are transported if you specify `GRANTS=Y` during export. During import, some grants may fail. For example, the user being granted a certain right may not exist, or a role being granted a particular right may not exist.

Partitioned Tables

You cannot move a partitioned table using transportable tablespaces when only a subset of the partitioned table is contained in the set of tablespaces. You must ensure that all partitions in a table are in the tablespace set, or exchange the partitions into tables before copying the tablespace set. However, you should note that exchanging partitions with tables invalidates the global index of the partitioned table.

At the target database, you can exchange the tables back into partitions if there is already a partitioned table that exactly matches the column in the target database. If all partitions of that table come from the same foreign database, the exchange operation is guaranteed to succeed. If they do not, in rare cases, the exchange operation may return an error indicating that there is a data object number conflict.

If you receive a data object number conflict error when exchanging tables back into partitions, you can move the offending partition using the `ALTER TABLE MOVE PARTITION` statement. After doing so, retry the exchange operation.

If you specify the `WITHOUT VALIDATION` option of the exchange statement, the statement returns immediately because it only manipulates structural information. Moving partitions, however, may be slow because the data in the partition can be copied.

See Also: ["Transporting and Attaching Partitions for Data Warehousing"](#) on page 11-46 for an example of transporting a partitioned table

Objects

A transportable tablespace set can contain:

- Tables
- Indexes
- Domain indexes
- Bitmap indexes
- Index-organized tables
- LOBs
- Nested tables
- Varrays
- Tables with user-defined type columns

If the tablespace set contains a pointer to a `BFILE`, you must move the `BFILE` and set the directory correctly in the target database.

Advanced Queues

You can use transportable tablespaces to move or copy Oracle advanced queues, as long as these queues are not 8.0 compatible queues with multiple recipients. After a queue is transported to a target database, the queue is initially disabled. After making the transported tablespaces read-write in the target database, you can enable the queue by starting it up using the built-in PL/SQL routine `DBMS_AQADM.START_QUEUE`.

Indexes

You can transport regular indexes, domain indexes, and bitmap indexes. When the transportable set fully contains a partitioned table, you can also transport the global index of the partitioned table.

Function-based indexes are not supported. If they exist in a tablespace, you must drop them before you can transport the tablespace.

Triggers

Triggers are exported without a validity check. In other words, Oracle does not verify that the trigger refers only to objects within the transportable set. Invalid triggers cause a compilation error during the subsequent import.

Materialized Views/Replication

Transporting materialized views or replication structural information is not supported. When transporting a tablespace, the materialized view or replication metadata associated with the tables in the tablespace is not exported and, thus, is not be available to the target database.

Using Transportable Tablespaces

The following are some possible applications for transportable tablespaces.

Transporting and Attaching Partitions for Data Warehousing

Typical enterprise data warehouses contain one or more large fact tables. These fact tables can be partitioned by date, making the enterprise data warehouse a historical database. You can build indexes to speed up star queries. In fact, Oracle recommends that you build local indexes for such historically partitioned tables to avoid rebuilding global indexes every time you drop the oldest partition from the historical database.

Suppose every month you would like to load one month's worth of data into the data warehouse. There is a large fact table in the data warehouse called `sales`, which has the following columns:

```
CREATE TABLE sales (invoice_no NUMBER,
  sale_year INT NOT NULL,
  sale_month INT NOT NULL,
  sale_day INT NOT NULL)
PARTITION BY RANGE (sale_year, sale_month, sale_day)
(partition jan98 VALUES LESS THAN (1998, 2, 1),
 partition feb98 VALUES LESS THAN (1998, 3, 1),
 partition mar98 VALUES LESS THAN (1998, 4, 1),
 partition apr98 VALUES LESS THAN (1998, 5, 1),
 partition may98 VALUES LESS THAN (1998, 6, 1),
 partition jun98 VALUES LESS THAN (1998, 7, 1));
```

You create a local nonprefixed index:

```
CREATE INDEX sales_index ON sales(invoice_no) LOCAL;
```

Initially, all partitions are empty, and are in the same default tablespace. Each month, you want to create one partition and attach it to the partitioned `sales` table.

Suppose it is July 1998, and you would like to load the July sales data into the partitioned table. In a staging database, you create a new tablespace, `ts_jul`. You also create a table, `jul_sales`, in that tablespace with exactly the same column types as the `sales` table. You can create the table `jul_sales` using the `CREATE TABLE ... AS SELECT` statement. After creating and populating `jul_sales`, you can also create an index, `jul_sale_index`, for the table, indexing the same column as the local index in the `sales` table. After building the index, transport the tablespace `ts_jul` to the data warehouse.

In the data warehouse, add a partition to the `sales` table for the July sales data. This also creates another partition for the local nonprefixed index:

```
ALTER TABLE sales ADD PARTITION jul98 VALUES LESS THAN (1998, 8, 1);
```

Attach the transported table `jul_sales` to the table `sales` by exchanging it with the new partition:

```
ALTER TABLE sales EXCHANGE PARTITION jul98 WITH TABLE jul_sales
    INCLUDING INDEXES
    WITHOUT VALIDATION;
```

This statement places the July sales data into the new partition `jul98`, attaching the new data to the partitioned table. This statement also converts the index `jul_sale_index` into a partition of the local index for the `sales` table. This statement should return immediately, because it only operates on the structural information and it simply switches database pointers. If you know that the data in the new partition does not overlap with data in previous partitions, you are advised to specify the `WITHOUT VALIDATION` option. Otherwise, the statement goes through all the new data in the new partition in an attempt to validate the range of that partition.

If all partitions of the `sales` table came from the same staging database (the staging database is never destroyed), the exchange statement always succeeds. In general, however, if data in a partitioned table comes from different databases, it's possible that the exchange operation may fail. For example, if the `jan98` partition of `sales` did not come from the same staging database, the above exchange operation can fail, returning the following error:

```
ORA-19728: data object number conflict between table JUL_SALES and partition
JAN98 in table SALES
```

To resolve this conflict, move the offending partition by issuing the following statement:

```
ALTER TABLE sales MOVE PARTITION jan98;
```

Then retry the exchange operation.

After the exchange succeeds, you can safely drop `jul_sales` and `jul_sale_index` (both are now empty). Thus you have successfully loaded the July sales data into your data warehouse.

Publishing Structured Data on CDs

Transportable tablespaces provide a way to publish structured data on CDs. A data provider can load a tablespace with data to be published, generate the transportable set, and copy the transportable set to a CD. This CD can then be distributed.

When customers receive this CD, they can plug it into an existing database without having to copy the datafiles from the CD to disk storage. For example, suppose on a Windows NT machine D: drive is the CD drive. You can plug in a transportable set with datafile `catalog.f` and export file `expdat.dmp` as follows:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='D:\catalog.f' FILE='D:\expdat.dmp'
```

You can remove the CD while the database is still up. Subsequent queries to the tablespace return an error indicating that Oracle cannot open the datafiles on the CD. However, operations to other parts of the database are not affected. Placing the CD back into the drive makes the tablespace readable again.

Removing the CD is the same as removing the datafiles of a read-only tablespace. If you shut down and restart the database, Oracle indicates that it cannot find the removed datafile and does not open the database (unless you set the initialization parameter `READ_ONLY_OPEN_DELAYED` to `TRUE`). When `READ_ONLY_OPEN_DELAYED` is set to `TRUE`, Oracle reads the file only when someone queries the plugged-in tablespace. Thus, when plugging in a tablespace on a CD, you should always set the `READ_ONLY_OPEN_DELAYED` initialization parameter to `TRUE`, unless the CD is permanently attached to the database.

Mounting the Same Tablespace Read-Only on Multiple Databases

You can use transportable tablespaces to mount a tablespace read-only on multiple databases. In this way, separate databases can share the same data on disk instead of duplicating data on separate disks. The tablespace datafiles must be accessible by all databases. To avoid database corruption, the tablespace must remain read-only in all the databases mounting the tablespace.

You can mount the same tablespace read-only on multiple databases in either of the following ways:

- Plug the tablespace into each of the databases on which you want to mount the tablespace. Generate a transportable set in a single database. Put the datafiles in the transportable set on a disk accessible to all databases. Import the structural information into each database.
- Generate the transportable set in one of the databases and plug it into other databases. If you use this approach, it is assumed that the datafiles are already on the shared disk, and they belong to an existing tablespace in one of the databases. You can make the tablespace read-only, generate the transportable set, and then plug the tablespace in to other databases while the datafiles remain in the same location on the shared disk.

You can make the disk accessible by multiple computers in several ways. You can use either a cluster file system or raw disk, because that is required by Oracle9i Real Application Clusters. Because Oracle reads only these type of datafiles on shared disk, you can also use NFS. Be aware, however, that if a user queries the shared tablespace while NFS is down, the database will hang until the NFS operation times out.

Later, you can drop the read-only tablespace in some of the databases. Doing so does not modify the datafiles for the tablespace. Thus, the drop operation does not corrupt the tablespace. Do not make the tablespace read-write unless only one database is mounting the tablespace.

Archive Historical Data Using Transportable Tablespaces

Since a transportable tablespace set is a self-contained set of files that can be plugged into any Oracle database, you can archive old/historical data in an enterprise data warehouse using the transportable tablespace procedures described in this chapter.

See Also: *Oracle9i Data Warehousing Guide* for more details

Using Transportable Tablespaces to Perform TSPITR

You can use transportable tablespaces to perform tablespace point-in-time recovery (TSPITR).

See Also: *Oracle9i User-Managed Backup and Recovery Guide* for information about how to perform TSPITR using transportable tablespaces

Viewing Tablespace Information

The following data dictionary and dynamic performance views provide useful information about the tablespaces of a database.

View	Description
V\$TABLESPACE	Name and number of all tablespaces from the control file.
DBA_TABLESPACES, USER_TABLESPACES	Descriptions of all (or user accessible) tablespaces.
DBA_SEGMENTS, USER_SEGMENTS	Information about segments within all (or user accessible) tablespaces.
DBA_EXTENTS, USER_EXTENTS	Information about data extents within all (or user accessible) tablespaces.
DBA_FREE_SPACE, USER_FREE_SPACE	Information about free extents within all (or user accessible) tablespaces.
V\$DATAFILE	Information about all datafiles, including tablespace number of owning tablespace.
V\$TEMPFILE	Information about all tempfiles, including tablespace number of owning tablespace.
DBA_DATA_FILES	Shows files (datafiles) belonging to tablespaces.
DBA_TEMP_FILES	Shows files (tempfiles) belonging to temporary tablespaces.
V\$TEMP_EXTENT_MAP	Information for all extents in all locally managed temporary tablespaces.
V\$TEMP_EXTENT_POOL	For locally managed temporary tablespaces: the state of temporary space cached and used for by each instance.
V\$TEMP_SPACE_HEADER	Shows space used/free for each tempfile.
DBA_USERS	Default and temporary tablespaces for all users.
DBA_TS_QUOTAS	Lists tablespace quotas for all users.
V\$SORT_SEGMENT	Information about every sort segment in a given instance. The view is only updated when the tablespace is of the <code>TEMPORARY</code> type.
V\$SORT_USER	Temporary sort space usage by user and temporary/permanent tablespace.

The following are just a few examples of using some of these views.

See Also: *Oracle9i Database Reference* for complete description of these views

Listing Tablespaces and Default Storage Parameters: Example

To list the names and default storage parameters of all tablespaces in a database, use the following query on the DBA_TABLESPACES view:

```
SELECT TABLESPACE_NAME "TABLESPACE",
       INITIAL_EXTENT "INITIAL_EXT",
       NEXT_EXTENT "NEXT_EXT",
       MIN_EXTENTS "MIN_EXT",
       MAX_EXTENTS "MAX_EXT",
       PCT_INCREASE
FROM DBA_TABLESPACES;
```

TABLESPACE	INITIAL_EXT	NEXT_EXT	MIN_EXT	MAX_EXT	PCT_INCREASE
RBS	1048576	1048576	2	40	0
SYSTEM	106496	106496	1	99	1
TEMP	106496	106496	1	99	0
TESTTBS	57344	16384	2	10	1
USERS	57344	57344	1	99	1

Listing the Datafiles and Associated Tablespaces of a Database: Example

To list the names, sizes, and associated tablespaces of a database, enter the following query on the DBA_DATA_FILES view:

```
SELECT FILE_NAME, BLOCKS, TABLESPACE_NAME
FROM DBA_DATA_FILES;
```

FILE_NAME	BLOCKS	TABLESPACE_NAME
/U02/ORACLE/IDDB3/RBS01.DBF	1536	RBS
/U02/ORACLE/IDDB3/SYSTEM01.DBF	6586	SYSTEM
/U02/ORACLE/IDDB3/TEMP01.DBF	6400	TEMP
/U02/ORACLE/IDDB3/TESTTBS01.DBF	6400	TESTTBS
/U02/ORACLE/IDDB3/USERS01.DBF	384	USERS

Displaying Statistics for Free Space (Extents) of Each Tablespace: Example

To produce statistics about free extents and coalescing activity for each tablespace in the database, enter the following query:

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
       COUNT(*)         "PIECES",
       MAX(blocks)      "MAXIMUM",
       MIN(blocks)      "MINIMUM",
       AVG(blocks)      "AVERAGE",
       SUM(blocks)      "TOTAL"
FROM   DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME, FILE_ID;
```

TABLESPACE	FILE_ID	PIECES	MAXIMUM	MINIMUM	AVERAGE	TOTAL
RBS	2	1	955	955	955	955
SYSTEM	1	1	119	119	119	119
TEMP	4	1	6399	6399	6399	6399
TESTTBS	5	5	6364	3	1278	6390
USERS	3	1	363	363	363	363

PIECES shows the number of free space extents in the tablespace file, **MAXIMUM** and **MINIMUM** show the largest and smallest contiguous area of space in database blocks, **AVERAGE** shows the average size in blocks of a free space extent, and **TOTAL** shows the amount of free space in each tablespace file in blocks. This query is useful when you are going to create a new object or you know that a segment is about to extend, and you want to make sure that there is enough space in the containing tablespace.

Managing Datafiles

This chapter describes the various aspects of datafile management, and contains the following topics:

- [Guidelines for Managing Datafiles](#)
- [Creating Datafiles and Adding Datafiles to a Tablespace](#)
- [Changing a Datafile's Size](#)
- [Altering Datafile Availability](#)
- [Renaming and Relocating Datafiles](#)
- [Dropping Datafiles](#)
- [Verifying Data Blocks in Datafiles](#)
- [Mapping Files to Physical Devices](#)
- [Viewing Datafile Information](#)

See Also: [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating datafiles and tempfiles that are both created and managed by the Oracle database server

Guidelines for Managing Datafiles

Datafiles are physical files of the operating system that store the data of all logical structures in the database. They must be explicitly created for each tablespace. Oracle assigns each datafile two associated file numbers, an absolute file number and a relative file number, that are used to uniquely identify it. These numbers are described in the following table:

Type of File Number	Description
Absolute	Uniquely identifies a datafile in the database. In earlier releases of Oracle, the absolute file number may have been referred to as simply, the "file number."
Relative	Uniquely identifies a datafile within a tablespace. For small and medium size databases, relative file numbers usually have the same value as the absolute file number. However, when the number of datafiles in a database exceeds a threshold (typically 1023), the relative file number differs from the absolute file number.

File numbers are displayed in many data dictionary views. You can optionally use file numbers instead of file names to identify datafiles or tempfiles in SQL statements. When using a file number, specify the file number that is displayed in the `FILE#` column of the `V$DATAFILE` or `V$TEMPFILE` view. This file number is also displayed in the `FILE_ID` column of the `DBA_DATA_FILES` or `DBA_TEMP_FILES` view.

This section describes aspects of managing datafiles, and contains the following topics:

- [Determine the Number of Datafiles](#)
- [Determine the Size of Datafiles](#)
- [Place Datafiles Appropriately](#)
- [Store Datafiles Separate from Redo Log Files](#)

Determine the Number of Datafiles

At least one datafile is required for the `SYSTEM` tablespace of a database. A small system might have a single datafile. The following are some guidelines to consider when determining the number of datafiles for your database.

Determine the Value of the DB_FILES Initialization Parameter

When starting an Oracle instance, the `DB_FILES` initialization parameter indicates the amount of SGA space to reserve for datafile information and thus, the maximum number of datafiles that can be created for the instance. This limit applies for the life of the instance. You can change the value of `DB_FILES` (by changing the initialization parameter setting), but the new value does not take effect until you shut down and restart the instance.

Note: The default value of `DB_FILES` is operating system specific.

When determining a value for `DB_FILES`, take the following into consideration:

- If the value of `DB_FILES` is too low, you cannot add datafiles beyond the `DB_FILES` limit without first shutting down the database.
- If the value of `DB_FILES` is too high, memory is unnecessarily consumed.

Limitations When Adding Datafiles to a Tablespace

You can add datafiles to tablespaces, subject to the following limitations:

- Operating systems often impose a limit on the number of files a process can open simultaneously. More datafiles cannot be created when the operating system limit of open files is reached.
- Operating systems impose limits on the number and size of datafiles.
- Oracle imposes a maximum limit on the number of datafiles for any Oracle database opened by any instance. This limit is operating system specific.
- You cannot exceed the number of datafiles specified by the `DB_FILES` initialization parameter.
- When you issue `CREATE DATABASE` or `CREATE CONTROLFILE` statements, the `MAXDATAFILES` parameter specifies an initial size of the datafile portion of the control file. However, if you attempt to add a new file whose number is greater than `MAXDATAFILES`, but less than or equal to `DB_FILES`, the control file will expand automatically so that the datafiles section can accommodate more files.

Consider the Performance Impact

The number of datafiles comprising a tablespace, and ultimately the database, can have an impact upon performance.

Oracle allows more datafiles in the database than the operating system defined limit. Oracle's DBWn processes can open all online datafiles. Oracle is capable of treating open file descriptors as a cache, automatically closing files when the number of open file descriptors reaches the operating system-defined limit. This can have a negative performance impact. When possible, adjust the operating system limit on open file descriptors so that it is larger than the number of online datafiles in the database.

See Also:

- Your operating system specific Oracle documentation for more information on operating system limits
- *Oracle9i SQL Reference* for more information about the MAXDATAFILES parameter of the CREATE DATABASE or CREATE CONTROLFILE statement

Determine the Size of Datafiles

The first datafile (in the original SYSTEM tablespace) must be at least 150M to contain the initial data dictionary and rollback segment. If you install other Oracle products, they may require additional space in the SYSTEM tablespace. See the installation instructions for these products for information about their space requirements.

Place Datafiles Appropriately

Tablespace location is determined by the physical location of the datafiles that constitute that tablespace. Use the hardware resources of your computer appropriately.

For example, if several disk drives are available to store the database, consider placing potentially contending datafiles on separate disks. This way, when users query information, both disk drives can work simultaneously, retrieving data at the same time.

Store Datafiles Separate from Redo Log Files

Datafiles should not be stored on the same disk drive that stores the database's redo log files. If the datafiles and redo log files are stored on the same disk drive and that disk drive fails, the files cannot be used in your database recovery procedures.

If you multiplex your redo log files, then the likelihood of losing all of your redo log files is low, so you can store datafiles on the same drive as some redo log files.

Creating Datafiles and Adding Datafiles to a Tablespace

When creating a tablespace, you should estimate the potential size of database objects and create sufficient datafiles. Later, if needed, you can create additional datafiles and add them to a tablespace to increase the total amount of disk space allocated to it, and consequently the database. Preferably, place datafiles on multiple devices, so as to ensure that data is spread evenly across all devices.

You can create datafiles and associate them with a tablespace using any of the statements listed in the following table. In all cases, you can either specify the file specifications for the datafiles being created, or you can use the Oracle Managed Files feature to create files that are created and managed by the database server. The table includes a brief description of the statement, as used to create datafiles, and references the section of this book where use of the statement is most completely described:

SQL Statement	Description	For more information...
CREATE TABLESPACE	Creates a tablespace and the datafiles that comprise it	"Creating Tablespaces" on page 11-3
CREATE TEMPORARY TABLESPACE	Creates a locally-managed temporary tablespace and the <i>tempfiles</i> (tempfiles are a special kind of datafile) that comprise it	"Creating a Locally Managed Temporary Tablespace" on page 11-13
ALTER TABLESPACE ... ADD DATAFILE	Creates and adds a datafile to a tablespace	"Altering a Dictionary-Managed Tablespace" on page 11-11
ALTER TABLESPACE ... ADD TEMPFILE	Creates and adds a tempfile to a temporary tablespace	"Creating a Locally Managed Temporary Tablespace" on page 11-13
CREATE DATABASE	Creates a database and associated datafiles	"Manually Creating an Oracle Database" on page 2-14
ALTER DATABASE ... CREATE DATAFILE	Creates a new empty datafile in place of an old one--useful to re-create a datafile that was lost with no backup.	Not discussed in this book. See <i>Oracle9i User-Managed Backup and Recovery Guide</i> .

If you add new datafiles to a tablespace and do not fully specify the filenames, Oracle creates the datafiles in the default database directory or the current

directory, depending upon your operating system. Oracle recommends you always specify a fully qualified name for a datafile. Unless you want to reuse existing files, make sure the new filenames do not conflict with other files. Old files that have been previously dropped will be overwritten.

If a statement that creates a datafile fails, Oracle removes any created operating system files. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands.

Changing a Datafile's Size

This section describes the various ways to alter the size of a datafile, and contains the following topics:

- [Enabling and Disabling Automatic Extension for a Datafile](#)
- [Manually Resizing a Datafile](#)

Enabling and Disabling Automatic Extension for a Datafile

You can create datafiles or alter existing datafiles so that they automatically increase in size when more space is needed in the database. The files increase in specified increments up to a specified maximum.

Setting your datafiles to extend automatically provides these advantages:

- Reduces the need for immediate intervention when a tablespace runs out of space
- Ensures applications will not halt because of failures to allocate extents

To determine whether a datafile is auto-extensible, query the `DBA_DATA_FILES` view and examine the `AUTOEXTENSIBLE` column.

You can specify automatic file extension by specifying an `AUTOEXTEND ON` clause when you create datafiles using the following SQL statements:

- `CREATE DATABASE`
- `CREATE TABLESPACE`
- `ALTER TABLESPACE`

You can enable or disable automatic file extension for existing datafiles, or manually resize a datafile using the `ALTER DATABASE` statement.

The following example enables automatic extension for a datafile added to the `users` tablespace:

```
ALTER TABLESPACE users
  ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
  AUTOEXTEND ON
  NEXT 512K
  MAXSIZE 250M;
```

The value of `NEXT` is the minimum size of the increments added to the file when it extends. The value of `MAXSIZE` is the maximum size to which the file can automatically extend.

The next example disables the automatic extension for the datafile.

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
  AUTOEXTEND OFF;
```

See Also: *Oracle9i SQL Reference* for more information about the SQL statements for creating or altering datafiles

Manually Resizing a Datafile

You can manually increase or decrease the size of a datafile using the `ALTER DATABASE` statement.

Because you can change the sizes of datafiles, you can add more space to your database without adding more datafiles. This is beneficial if you are concerned about reaching the maximum number of datafiles allowed in your database.

Manually reducing the sizes of datafiles enables you to reclaim unused space in the database. This is useful for correcting errors in estimates of space requirements.

In the next example, assume that the datafile `/u02/oracle/rbdb1/stuff01.dbf` has extended up to 250M. However, because its tablespace now stores smaller objects, the datafile can be reduced in size.

The following statement decreases the size of datafile

```
/u02/oracle/rbdb1/stuff01.dbf:
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
  RESIZE 100M;
```

Note: It is not always possible to decrease the size of a file to a specific value.

Altering Datafile Availability

You can take individual datafiles or tempfiles of a tablespace offline or similarly, bring them online. Offline datafiles are unavailable to the database and cannot be accessed until they are brought back online. You also have the option of taking all datafiles or tempfiles comprising a tablespace offline or online simply by specifying the name of a tablespace.

One example of where you might be required to alter the availability of a datafile is when Oracle has problems writing to a datafile and automatically takes the datafile offline. Later, after resolving the problem, you can bring the datafile back online manually.

The files of a read-only tablespace can independently be taken offline or brought online just as for read-write tablespaces. Bringing a datafile online in a read-only tablespace makes the file readable. No one can write to the file unless its associated tablespace is returned to the read-write state.

To take a datafile offline, or bring it online, you must have the `ALTER DATABASE` system privilege. To take all datafiles or tempfiles offline using the `ALTER TABLESPACE` statement, you must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege. In an Oracle Real Application Clusters environment, the database must be open in exclusive mode.

This section describes ways to alter datafile availability, and contains the following topics:

- [Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode](#)
- [Taking Datafiles Offline in NOARCHIVELOG Mode](#)
- [Altering the Availability of All Datafiles or Tempfiles in a Tablespace](#)

Note: You can make all datafiles in any tablespace, except the files in the `SYSTEM` tablespace, temporarily unavailable by taking the tablespace offline. You *must* leave these files in the tablespace to bring the tablespace back online.

For more information about taking a tablespace offline, see "[Taking Tablespaces Offline](#)" on page 11-21.

Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode

To bring an individual datafile online, issue the `ALTER DATABASE` statement and include the `DATAFILE` clause. The following statement brings the specified datafile online:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

To take the same file offline, issue the following statement:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

Note: To use this form of the `ALTER DATABASE` statement, the database must be in ARCHIVELOG mode. This requirement prevents you from accidentally losing the datafile, since taking the datafile offline while in NOARCHIVELOG mode is likely to result in losing the file.

Taking Datafiles Offline in NOARCHIVELOG Mode

To take a datafile offline when the database is in NOARCHIVELOG mode, use the `ALTER DATABASE` statement with both the `DATAFILE` and `OFFLINE DROP` clauses. This enables you to take the datafile offline and drop it immediately. It is useful, for example, if the datafile contains only data from temporary segments and has not been backed up and the database is in NOARCHIVELOG mode.

The following statement takes the specified datafile offline:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE DROP;
```

Altering the Availability of All Datafiles or Tempfiles in a Tablespace

Clauses of the `ALTER TABLESPACE` statement allow you to change the online or offline status of all of the datafiles or tempfiles within a tablespace. Specifically, the statements that affect online/offline status are:

- `ALTER TABLESPACE ... DATAFILE { ONLINE | OFFLINE }`
- `ALTER TABLESPACE ... TEMPFILE { ONLINE | OFFLINE }`

You are required only to enter the tablespace name, not the individual datafiles or tempfiles. All of the datafiles or tempfiles are affected, but the online/offline status of the tablespace itself is not changed.

In most cases the above `ALTER TABLESPACE` statements can be issued whenever the database is mounted, even if it is not open. However, the database *must not* be open if the tablespace is the system tablespace, an undo tablespace, or the default temporary tablespace. The `ALTER DATABASE DATAFILE` and `ALTER DATABASE TEMPFILE` statements also have `ONLINE/OFFLINE` clauses, however in those statements you must enter all of the filenames for the tablespace.

The syntax is different from the `ALTER TABLESPACE . . . ONLINE | OFFLINE` statement that alters a tablespace's availability, because that is a different operation. The `ALTER TABLESPACE` statement takes datafiles offline as well as the tablespace, but it cannot be used to alter the status of a temporary tablespace or its tempfile(s).

Renaming and Relocating Datafiles

You can rename datafiles to either change their names or relocate them. Some options, and procedures which you can follow, are described in the following sections:

- [Renaming and Relocating Datafiles for a Single Tablespace](#)

For example, renaming *filename1* and *filename2* in *tablespace1*, while the rest of the database is open.

- [Renaming and Relocating Datafiles for Multiple Tablespaces](#)

For example, renaming *filename1* in *tablespace1* and *filename2* in *tablespace2*, while the database is mounted but closed.

Note: To rename or relocate datafiles of the `SYSTEM` tablespace, you must use the second option, because you cannot take the `SYSTEM` tablespace offline.

When you rename and relocate datafiles with these procedures, only the pointers to the datafiles, as recorded in the database's control file, are changed. The procedures do not physically rename any operating system files, nor do they copy files at the operating system level. Renaming and relocating datafiles involves several steps. Read the steps and examples carefully before performing these procedures.

Renaming and Relocating Datafiles for a Single Tablespace

The section offers some procedures for renaming and relocating datafiles in a single tablespace. You must have the `ALTER TABLESPACE` system privilege to rename datafiles of a single tablespace.

Renaming Datafiles in a Single Tablespace

To rename datafiles from a single tablespace, complete the following steps:

1. Take the non-SYSTEM tablespace that contains the datafiles offline.

For example:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

2. Rename the datafiles using the operating system.
3. Use the `ALTER TABLESPACE` statement with the `RENAME DATAFILE` clause to change the filenames within the database.

For example, the following statement renames the datafiles

`/u02/oracle/rbdb1/user1.dbf` and `/u02/oracle/rbdb1/user2.dbf`
to `/u02/oracle/rbdb1/users01.dbf` and `/u02/oracle/rbdb1/users02.dbf`,
respectively:

```
ALTER TABLESPACE users
  RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
                  '/u02/oracle/rbdb1/user2.dbf'
  TO '/u02/oracle/rbdb1/users01.dbf',
    '/u02/oracle/rbdb1/users02.dbf';
```

The new files must already exist; this statement does not create the files. Also, always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile name exactly as it appears in the `DBA_DATA_FILES` view of the data dictionary.

4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Relocating and Renaming Datafiles in a Single Tablespace

Here is an example that illustrates the steps involved for relocating a datafile.

Assume the following conditions:

- An open database has a tablespace named `users` that is made up of datafiles all located on the same disk.

- The datafiles of the `users` tablespace are to be relocated to different and separate disk drives.
- You are currently connected with administrator privileges to the open database.
- You have a current backup of the database.

Complete the following steps:

1. Identify the datafile names of interest.

The following query of the data dictionary view `DBA_DATA_FILES` lists the datafile names and respective sizes (in bytes) of the `users` tablespace:

```
SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
WHERE TABLESPACE_NAME = 'USERS';
```

FILE_NAME	BYTES
/U02/ORACLE/RBDB1/USERS01.DBF	102400000
/U02/ORACLE/RBDB1/USERS02.DBF	102400000

2. Take the tablespace containing the datafiles offline, or shut down the database and restart and mount it, leaving it closed. Either option closes the datafiles of the tablespace.
3. Copy the datafiles to their new locations and rename them using the operating system.

Note: You can execute an operating system command to copy a file by using the SQL*Plus `HOST` command.

4. Rename the datafiles within Oracle.

The datafile pointers for the files that make up the `users` tablespace, recorded in the control file of the associated database, must now be changed from the old names to the new names.

If the tablespace is offline but the database is open, use the `ALTER TABLESPACE ... RENAME DATAFILE` statement. If the database is mounted but closed, use the `ALTER DATABASE ... RENAME FILE` statement.

```
ALTER TABLESPACE users
  RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
                 '/u02/oracle/rbdb1/users02.dbf'
  TO '/u03/oracle/rbdb1/users01.dbf',
```

```
' /u04/oracle/rbdb1/users02.dbf ' ;
```

5. Bring the tablespace online, or open the database.

If the `users` tablespace is offline and the database is open, bring the tablespace back online. If the database is mounted but closed, open the database.

6. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Renaming and Relocating Datafiles for Multiple Tablespaces

You can rename and relocate datafiles of one or more tablespaces using `ALTER DATABASE` statement with the `RENAME FILE` clause. This option is the only choice if you want to rename or relocate datafiles of several tablespaces in one operation, or rename or relocate datafiles of the `SYSTEM` tablespace. If the database must remain open, consider instead the procedure outlined in the previous section.

To rename datafiles of several tablespaces in one operation or to rename datafiles of the `SYSTEM` tablespace, you must have the `ALTER DATABASE` system privilege.

To rename datafiles in multiple tablespaces, follow these steps.

1. Ensure that the database is mounted but closed.
2. Copy the datafiles to be renamed to their new locations and new names, using the operating system.
3. Use `ALTER DATABASE` to rename the file pointers in the database's control file.

For example, the following statement renames the datafiles `/u02/oracle/rbdb1/sort01.dbf` and `/u02/oracle/rbdb1/user3.dbf` to `/u02/oracle/rbdb1/temp01.dbf` and `/u02/oracle/rbdb1/users03.dbf`, respectively:

```
ALTER DATABASE
  RENAME FILE ' /u02/oracle/rbdb1/sort01.dbf ' ,
             ' /u02/oracle/rbdb1/user3.dbf '
  TO ' /u02/oracle/rbdb1/temp01.dbf ' ,
     ' /u02/oracle/rbdb1/users03.dbf ' ;
```

The new files must already exist; this statement does not create the files. Also, always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile name exactly as it appears in the `DBA_DATA_FILES` view of the data dictionary.

4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Dropping Datafiles

There is no SQL statement that specifically drops a datafile. The only means of dropping a datafile is to drop the tablespace that contains the datafile. For example, if you want to remove a datafile from a tablespace, you could do the following:

1. Create a new tablespace
2. Move the data from the old tablespace to the new one
3. Drop the old tablespace

You can, however, drop a tempfile using the `ALTER DATABASE` statement. For example:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
INCLUDING DATAFILES;
```

See Also: [Dropping Tablespaces](#) on page 11-29

Verifying Data Blocks in Datafiles

If you want to configure Oracle to use checksums to verify data blocks, set the initialization parameter `DB_BLOCK_CHECKSUM` to `TRUE`. The value of this parameter can be changed dynamically, or set in the initialization parameter file. The default value of `DB_BLOCK_CHECKSUM` is `FALSE`. Regardless of the setting of this parameter, checksums are always used to verify data blocks in the system tablespace.

When you enable block checking, Oracle computes a checksum for each block written to disk. Checksums are computed for all data blocks, including temporary blocks.

The `DBWn` process calculates the checksum for each block and stores it in the block's header. Checksums are also computed by the direct loader.

The next time Oracle reads a data block, it uses the checksum to detect corruption in the block. If a corruption is detected, Oracle returns message `ORA-01578` and writes information about the corruption to a trace file.

Caution: Setting `DB_BLOCK_CHECKSUM` to `TRUE` can cause performance overhead. Set this parameter to `TRUE` only under the advice of Oracle Support personnel to diagnose data corruption problems.

See Also: Oracle9i Database Reference for information about checksums and the `DB_BLOCK_CHECKSUM` initialization parameter

Mapping Files to Physical Devices

In an environment where datafiles are simply file system files or are created directly on a raw device, it is relatively straight forward to see the association between a tablespace and the underlying device. Oracle provides views, such as `DBA_TABLESPACES`, `DBA_DATA_FILES`, and `V$DATAFILE`, that provide a mapping of files onto devices. These mappings, along with device statistics can be used to evaluate I/O performance.

However, with the introduction of host based Logical Volume Managers (LVM), and sophisticated storage subsystems that provide RAID (Redundant Array of Independent Disks) features, it is not easy to determine file to device mapping. This poses a problem because it becomes difficult to determine your "hottest" files when they are hidden behind a "black box". This section presents Oracle's approach to resolving this problem.

The following topics are contained in this section:

- [Overview of Oracle's File Mapping Interface](#)
- [How Oracle's File Mapping Interface Works](#)
- [Using Oracle's File Mapping Interface](#)
- [File Mapping Examples](#)

Note: This section presents an overview of Oracle's file mapping interface and explains how to use the `DBMS_STORAGE_MAP` package and dynamic performance views to expose the mapping of files onto physical devices. You can more easily access this functionality through the Oracle Enterprise Manager (OEM). It provides an easy to use graphical interface for mapping files to physical devices.

See the Oracle Enterprise Manager documentation set for more information.

Overview of Oracle's File Mapping Interface

To acquire an understanding of I/O performance, one must have detailed knowledge of the storage hierarchy in which files reside. Oracle provides a mechanism to show a complete mapping of a file to intermediate layers of logical volumes to actual physical devices. This is accomplished through a set of dynamic performance views (V\$ views). Using these views, you can locate the exact disk on which any block of a file resides.

To build these views, storage vendors must provide mapping libraries that are responsible for mapping their particular I/O stack elements. Oracle communicates with these libraries through an external non-Oracle process that is spawned by an Oracle background process called FMON. FMON is responsible for managing the mapping information. Oracle provides a PL/SQL package, `DBMS_STORAGE_MAP`, that you use to invoke mapping operations that populate the mapping views.

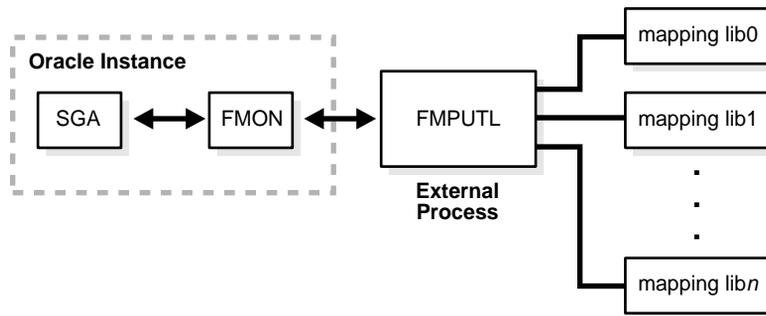
How Oracle's File Mapping Interface Works

This section describes the components of Oracle's file mapping interface and how the interface works. It contains the following topics:

- [Components of File Mapping](#)
- [Mapping Structures](#)
- [Example of Mapping Structures](#)
- [Configuration ID](#)

Components of File Mapping

The following figure shows the components of the file mapping mechanism.

Figure 12–1 Components of File Mapping

The following sections briefly describes these components and how they work together to populate the mapping views:

- [FMON](#)
- [External Process \(FMPUTL\)](#)
- [Mapping Libraries](#)

FMON FMON is a background process started by Oracle whenever the `FILE_MAPPING` initialization parameter is set to `TRUE`. FMON is responsible for:

- Building mapping information, which is stored in the SGA. This information is composed of the following structures:
 - Files
 - File system extents
 - Elements
 - Subelements

These structures are explained in "[Mapping Structures](#)" on page 12-18.

- Refreshing mapping information when a change occurs because of:
 - Changes to Oracle datafiles (size)
 - Addition or deletion of datafiles
 - Changes to the storage configuration (not frequent)
- Saving mapping information in the data dictionary to maintain a view of the information that is persistent across startup and shutdown operations

- Restoring mapping information into the SGA at instance startup. This avoids the need for a potentially expensive complete rebuild of the mapping information on every instance startup.

You help control this mapping using procedures that are invoked with the `DBMS_STORAGE_MAP` package.

External Process (FMPUTL) FMON spawns an external non-Oracle process called `FMPUTL`, that communicates directly with the vendor supplied mapping libraries. This process obtains the mapping information through all levels of the I/O stack, assuming that mapping libraries exist for all levels. On some platforms the external process requires that the `SETUID` bit is set to `ON` because root privileges are needed to map through all levels of the I/O mapping stack.

The external process is responsible for discovering the mapping libraries and dynamically loading them into its address space.

Mapping Libraries Oracle uses mapping libraries to discover mapping information for the elements that are owned by a particular mapping library. Through these mapping libraries information about individual I/O stack elements is communicated. This information is used to populate dynamic performance views that can be queried by users.

Mapping libraries need to exist for all levels of the stack for the mapping to be complete, and different libraries may own their own parts of the I/O mapping stack. For example, a VERITAS VxVM library would own the stack elements related to the VERITAS Volume Manager, and an EMC library would own all EMC storage specific layers of the I/O mapping stack.

Mapping libraries are vendor supplied. However, Oracle currently supplies a mapping library for EMC storage. The mapping libraries available to a database server are identified in a special file named `filemap.ora`.

Mapping Structures

The mapping structures and Oracle's representation of these structures are described in this section. You will need to understand this information in order to interpret the information in the mapping views.

The following are the primary structures that compose the mapping information:

- Files

A file mapping structure provides a set of attributes for a file, including file size, number of file system extents that the file is composed of, and the file type.

- File system extents

A file system extent mapping structure describes a contiguous chunk of blocks residing on one element. This includes the device offset, the extent size, the file offset, the type (data or parity), and the name of the element where the extent resides.

Note: File system extents are not the same as Oracle extents. File system extents are physical contiguous blocks of data written to a device as managed by the file system. Oracle extents are logical structures managed by Oracle, such as tablespace extents.

- Elements

An element mapping structure is the abstract mapping structure that describes a storage component within the I/O stack. Elements may be mirrors, stripes, partitions, RAID5, concatenated elements, and disks. These structures are the mapping building blocks.

- Subelements

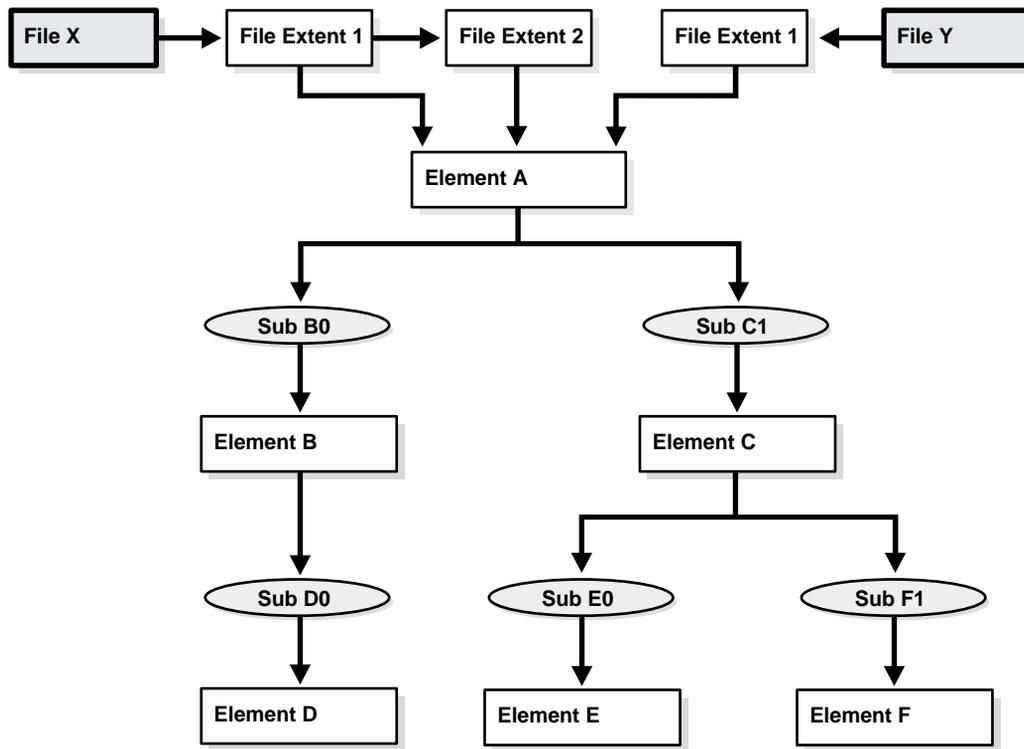
A subelement mapping structure describes the link between an element and the next elements in the I/O mapping stack. This structure contains the subelement number, size, the element name where the subelement exists, and the element offset.

All of these mapping structures are illustrated in the following example.

Example of Mapping Structures

Consider an Oracle database which is composed of two data files X and Y. Both files X and Y reside on a file system mounted on volume A. File X is composed of two extents while file Y is composed of only one extent. Element A is striped over two elements B and C. Element B is a partition of element D and element C is mirrored over elements E and F. Note that elements D, E, and F are physical disks. Subelement B0 connects the parent element A to element B, subelement C1 connects A to C,

All of the mapping structures are illustrated in [Figure 12-2](#).

Figure 12–2 Illustration of Mapping Structures

Note that the mapping structures represented are sufficient to describe the entire mapping information for the Oracle instance and consequently to map every logical block within the file into a (element name, element offset) tuple (or more in case of mirroring) at each level within the I/O stack.

Configuration ID

The configuration ID captures the version information associated with elements or files. The vendor library provides the configuration ID and updates it whenever a change occurs. Without a configuration ID, there is no way for Oracle to tell whether the mapping has changed.

There are two kinds of configuration IDs:

- Persistent

These configuration IDs are persistent across instance shutdown

- Non-persistent

The configuration IDs are not persistent across instance shutdown. Oracle is only capable of refreshing the mapping information while the instance is up.

Using Oracle's File Mapping Interface

This section discusses how to use Oracle's file mapping interface. It contains the following topics:

- [Enabling File Mapping](#)
- [Using the DBMS_STORAGE_MAP Package](#)
- [Obtaining Information from the File Mapping Views](#)

Enabling File Mapping

The following steps enable the file mapping feature:

1. Ensure that a valid `filemap.ora` file exists in the `$ORACLE_HOME/rdbms/filemap/etc` directory.

Caution: While the format and content of the `filemap.ora` file is discussed here, it is for informational reasons only. The `filemap.ora` file is created by Oracle when your system is installed. Until such time that vendors supply their own libraries, there will be only one entry in the `filemap.ora` file, and that is the Oracle supplied EMC library. This file should be modified manually by uncommenting this entry *only* if an EMC Symmetrix array is available.

The `filemap.ora` file is the configuration file that describes all of the available mapping libraries. FMON requires that a `filemap.ora` file exists and that it points to a valid path to mapping libraries. Otherwise, it will not start successfully.

The following row needs to be included for each library :

```
lib=vendor_name:mapping_library_path
```

where:

- *vendor_name* should be Oracle for the EMC Symmetric library
- *mapping_library_path* is the full path of the mapping library

Note that the ordering of the libraries in this file is extremely important. The libraries are queried based on their order in the configuration file.

The file mapping service can be even started even if no mapping libraries are available. The `filemap.ora` file still needs to be present even though it is empty. In this case, the mapping service is constrained in the sense that new mapping information cannot be discovered. Only restore and drop operations are allowed in such a configuration.

2. Set the `FILE_MAPPING` initialization parameter to `TRUE`.

```
FILE_MAPPING=TRUE
```

The instance does not have to be shut down to set this parameter. It can be set using an `ALTER SYSTEM` statement.

3. Invoke the appropriate `DBMS_STORAGE_MAP` mapping procedure. You have two options:
 - In a cold startup scenario, the Oracle database is just started and no mapping operation has been invoked yet. You execute the `DBMS_STORAGE_MAP.MAP_ALL` procedure to build the mapping information for the entire I/O subsystem associated with the Oracle database.
 - In a warm start scenario where the mapping information is already built, you have the option to invoke the `DBMS_STORAGE_MAP.MAP_SAVE` procedure to save the mapping information in the data dictionary. (Note that this procedure is invoked in `DBMS_STORAGE_MAP.MAP_ALL()` by default.) This forces all of the mapping information in the SGA to be flushed to disk.

Once you restart the database, use `DBMS_STORAGE_MAP.RESTORE()` to restore the mapping information into the SGA. If needed, `DBMS_STORAGE_MAP.MAP_ALL()` can be called to refresh the mapping information.

Using the `DBMS_STORAGE_MAP` Package

The `DBMS_STORAGE_MAP` package enables you control the mapping operations. The various procedures available to you are described in the following table.

Procedure	Use to:
<code>MAP_OBJECT</code>	Build the mapping information for the Oracle object identified by object name, owner, and type
<code>MAP_ELEMENT</code>	Build mapping information for the specified element

Procedure	Use to:
MAP_FILE	Build mapping information for the specified filename
MAP_ALL	Build entire mapping information for all types of Oracle files (excluding archive logs)
DROP_ELEMENT	Drop the mapping information for a specified element
DROP_FILE	Drop the file mapping information for the specified filename
DROP_ALL	Drop all mapping information in the SGA for this instance
SAVE	Save into the data dictionary the required information needed to regenerate the entire mapping
RESTORE	Load the entire mapping information from the data dictionary into the shared memory of the instance
LOCK_MAP	Lock the mapping information in the SGA for this instance
UNLOCK_MAP	Unlock the mapping information in the SGA for this instance

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of the DBMS_STORAGE_MAP package
- "[File Mapping Examples](#)" on page 12-25 for an example of using the DBMS_STORAGE_MAP package

Obtaining Information from the File Mapping Views

Mapping information generated by DBMS_STORAGE_MAP package is captured in dynamic performance views. Brief descriptions of these views are presented here.

View	Description
V\$MAP_LIBRARY	Contains a list of all mapping libraries that have been dynamically loaded by the external process
V\$MAP_FILE	Contains a list of all file mapping structures in the shared memory of the instance
V\$MAP_FILE_EXTENT	Contains a list of all file system extent mapping structures in the shared memory of the instance
V\$MAP_ELEMENT	Contains a list of all element mapping structures in the SGA of the instance

View	Description
V\$MAP_EXT_ELEMENT	Contains supplementary information for all element mapping
V\$MAP_SUBELEMENT	Contains a list of all subelement mapping structures in the shared memory of the instance
V\$MAP_COMP_LIST	Contains supplementary information for all element mapping structures.
V\$MAP_FILE_IO_STACK	The hierarchical arrangement of storage containers for the file displayed as a series of rows. Each row represents a level in the hierarchy.

See Also: *Oracle9i Database Reference* contains complete descriptions of the dynamic performance views

However, the information generated by the `DBMS_STORAGE_MAP.MAP_OBJECT` procedure is captured in a global temporary table named `MAP_OBJECT`. This table displays the hierarchical arrangement of storage containers for objects. Each row in the table represents a level in the hierarchy. A description of the `MAP_OBJECT` table follows.

Column	Datatype	Description
OBJECT_NAME	VARCHAR2(2000)	Name of the object
OBJECT_OWNER	VARCHAR2(2000)	Owner of the object
OBJECT_TYPE	VARCHAR2(2000)	Object type
FILE_MAP_IDX	NUMBER	File index (corresponds to <code>FILE_MAP_IDX</code> in <code>V\$MAP_FILE</code>)
DEPTH	NUMBER	Element depth within the I/O stack
ELEM_IDX	NUMBER	Index corresponding to element
CU_SIZE	NUMBER	Contiguous set of logical blocks of the file, in HKB units, that is resident contiguously on the element
STRIDE	NUMBER	Number of HKB between contiguous units (CU) in the file that are contiguous on this element. Used in RAID5 and striped files.

Column	Datatype	Description
NUM_CU	NUMBER	Number of contiguous units that are adjacent to each other on this element that are separated by STRIDE HKB in the file. In RAID5, the number of contiguous units also include the parity stripes.
ELEM_OFFSET	NUMBER	Element offset in HKB units
FILE_OFFSET	NUMBER	Offset in HKB units from the start of the file to the first byte of the contiguous units
DATA_TYPE	VARCHAR2(2000)	Datatype (DATA, PARITY, or DATA AND PARITY)
PARITY_POS	NUMBER	Position of the parity. Only for RAID5. This field is needed to distinguish the parity from the data part.
PARITY_PERIOD	NUMBER	Parity period. Only for RAID5.

File Mapping Examples

The following examples illustrates some of the powerful capabilities of Oracle's file mapping feature. This includes :

- The ability to map all Oracle files that span a particular device
- The ability to map a particular file into its corresponding devices
- The ability to map a particular Oracle object, including its block distribution at all levels within the I/O stack

Consider an Oracle instance which is composed of two datafiles:

- t_db1.f
- t_db2.f

These files are created on a Solaris UFS file system mounted on a VERITAS VxVM host based striped volume, /dev/vx/dsk/ipfdg/ipf-voll, that consists of the following host devices as externalized from an EMC Symmetrix array:

- /dev/vx/rdmp/c2t1d0s2
- /dev/vx/rdmp/c2t1d1s2

Note that the following examples require the execution of a MAP_ALL() operation.

Example 1: Map All Oracle Files that Span a Device

The following query returns all Oracle files associated with the /dev/vx/rdmp/c2t1d1s2 host device:

```
SELECT UNIQUE me.ELEM_NAME, mf.FILE_NAME
  FROM V$MAP_FILE_IO_STACK fs, V$MAP_FILE mf, V$MAP_ELEMENT me
 WHERE mf.FILE_MAP_IDX = fs.FILE_MAP_IDX
 AND me.ELEM_IDX = fs.ELEM_IDX
 AND me.ELEM_NAME = /dev/vx/rdmp/c2t1d1s2;
```

The query results are:

ELEM_NAME	FILE_NAME
/dev/vx/rdmp/c2t1d1s2	/oracle/dbs/t_db1.f
/dev/vx/rdmp/c2t1d1s2	/oracle/dbs/t_db2.f

Example 2: Map a File into Its Corresponding Devices

The following query displays a topological graph of the /oracle/dbs/t_db1.f datafile:

```
WITH fv AS
  (SELECT FILE_MAP_IDX, FILE_NAME FROM V$MAP_FILE
   WHERE FILE_NAME = /oracle/dbs/t_db1.f)
SELECT fv.FILE_NAME, LPAD(' ', 4 * (LEVEL - 1)) || el.ELEM_NAME ELEM_NAME
  FROM V$MAP_SUBELEMENT sb, V$MAP_ELEMENT el, fv,
       (SELECT UNIQUE ELEM_IDX FROM V$MAP_FILE_IO_STACK io, fv
        WHERE io.FILE_MAP_IDX = fv.FILE_MAP_IDX) fs
 WHERE el.ELEM_IDX = sb.CHILD_IDX
 AND fs.ELEM_IDX = el.ELEM_IDX
 START WITH sb.PARENT_IDX IN
  (SELECT DISTINCT ELEM_IDX
   FROM V$MAP_FILE_EXTENT fe, fv
   WHERE fv.FILE_MAP_IDX = fe.FILE_MAP_IDX)
 CONNECT BY PRIOR sb.CHILD_IDX = sb.PARENT_IDX;
```

The resulting topological graph is:

FILE_NAME	ELEM_NAME
/oracle/dbs/t_db1.f	_sym_plex_/dev/vx/rdsk/ipfdg/ipf-voll_-1_-1
/oracle/dbs/t_db1.f	_sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-voll_0_0_0
/oracle/dbs/t_db1.f	/dev/vx/rdmp/c2t1d0s2
/oracle/dbs/t_db1.f	_sym_symdev_000183600407_00C
/oracle/dbs/t_db1.f	_sym_hyper_000183600407_00C_0

```

/oracle/dbs/t_dbl.f                               _sym_hyper_000183600407_00C_1
/oracle/dbs/t_dbl.f                               _sym_subdisk_/dev/vx/rdisk/ipfdg/ipf-voll_0_1_0
/oracle/dbs/t_dbl.f                               /dev/vx/rdmp/c2t1d1s2
/oracle/dbs/t_dbl.f                               _sym_symdev_000183600407_00D
/oracle/dbs/t_dbl.f                               _sym_hyper_000183600407_00D_0
/oracle/dbs/t_dbl.f                               _sym_hyper_000183600407_00D_1

```

Example 3: Map an Oracle Object

This example displays the block distribution at all levels within the I/O stack for the `scott.bonus` table.

A `MAP_OBJECT()` operation must first be executed as follows:

```
EXECUTE DBMS_STORAGE_MAP.MAP_OBJECT('BONUS','SCOTT','TABLE');
```

The query is as follows:

```

SELECT io.OBJECT_NAME o_name, io.OBJECT_OWNER o_owner, io.OBJECT_TYPE o_type,
       mf.FILE_NAME, me.ELEM_NAME, io.DEPTH,
       (SUM(io.CU_SIZE * (io.NUM_CU - DECODE(io.PARITY_PERIOD, 0, 0,
       TRUNC(io.NUM_CU / io.PARITY_PERIOD)))) / 2) o_size
FROM MAP_OBJECT io, V$MAP_ELEMENT me, V$MAP_FILE mf
WHERE io.OBJECT_NAME = 'BONUS'
AND io.OBJECT_OWNER = 'SCOTT'
AND io.OBJECT_TYPE = 'TABLE'
AND me.ELEM_IDX = io.ELEM_IDX
AND mf.FILE_MAP_IDX = io.FILE_MAP_IDX
GROUP BY io.ELEM_IDX, io.FILE_MAP_IDX, me.ELEM_NAME, mf.FILE_NAME, io.DEPTH,
         io.OBJECT_NAME, io.OBJECT_OWNER, io.OBJECT_TYPE
ORDER BY io.DEPTH;

```

The following is the result of the query. Note that the `o_size` column is expressed in KB.

O_NAME	O_OWNER	O_TYPE	FILE_NAME	ELEM_NAME	DEPTH	O_SIZE
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	/dev/vx/dsk/ipfdg/ipf-voll	0	20
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	_sym_plex_/dev/vx/rdisk/ipfdg/ipf-voll_1_1_1	1	20
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	_sym_subdisk_/dev/vx/rdisk/ipfdg/ipf-voll_0_1_0	2	12
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	_sym_subdisk_/dev/vx/rdisk/ipfdg/ipf-voll_0_2_0	2	8
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	/dev/vx/rdmp/c2t1d1s2	3	12
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	/dev/vx/rdmp/c2t1d2s2	3	8
BONUS	SCOTT	TABLE	/oracle/dbs/t_dbl.f	_sym_symdev_000183600407_00D	4	12

BONUS	SCOTT	TABLE	/oracle/dbs/t_db1.f	_sym_symdev_000183600407_00E	4	8
BONUS	SCOTT	TABLE	/oracle/dbs/t_db1.f	_sym_hyper_000183600407_00D_0	5	12
BONUS	SCOTT	TABLE	/oracle/dbs/t_db1.f	_sym_hyper_000183600407_00D_1	5	12
BONUS	SCOTT	TABLE	/oracle/dbs/t_db1.f	_sym_hyper_000183600407_00E_0	6	8
BONUS	SCOTT	TABLE	/oracle/dbs/t_db1.f	_sym_hyper_000183600407_00E_1	6	8

Viewing Datafile Information

The following data dictionary views provide useful information about the datafiles of a database:

View	Description
DBA_DATA_FILES	Provides descriptive information about each datafile, including the tablespace to which it belongs and the file id. The file id can be used to join with other views for detail information.
DBA_EXTENTS USER_EXTENTS	DBA view describes the extents comprising all segments in the database. Contains the file id of the datafile containing the extent. USER view describes extents of the segments belonging to objects owned by the current user.
DBA_FREE_SPACE USER_FREE_SPACE	DBA view lists the free extents in all tablespaces. Includes the file id of the datafile containing the extent. USER view lists the free extents in the tablespaces accessible to the current user.
V\$DATAFILE	Contains datafile information from the control file
V\$DATAFILE_HEADER	Contains information from datafile headers

This example illustrates the use of one of these views, V\$DATAFILE.

```
SELECT NAME,
       FILE#,
       STATUS,
       CHECKPOINT_CHANGE# "CHECKPOINT"
FROM   V$DATAFILE;
```

NAME	FILE#	STATUS	CHECKPOINT
-----	----	-----	-----
/u01/oracle/rbdb1/system01.dbf	1	SYSTEM	3839
/u02/oracle/rbdb1/temp01.dbf	2	ONLINE	3782
/u02/oracle/rbdb1/users03.dbf	3	OFFLINE	3782

FILE# lists the file number of each datafile; the first datafile in the SYSTEM tablespace created with the database is always file 1. STATUS lists other information

about a datafile. If a datafile is part of the `SYSTEM` tablespace, its status is `SYSTEM` (unless it requires recovery). If a datafile in a non-`SYSTEM` tablespace is online, its status is `ONLINE`. If a datafile in a non-`SYSTEM` tablespace is offline, its status can be either `OFFLINE` or `RECOVER`. `CHECKPOINT` lists the final SCN (system change number) written for a datafile's most recent checkpoint.

See Also: *Oracle9i Database Reference* for a complete descriptions of these views

Managing Undo Space

This chapter describes how to manage undo space, either by using undo tablespaces or by using rollback segments. It contains the following topics:

- [What is Undo?](#)
- [Specifying the Mode for Undo Space Management](#)
- [Managing Undo Tablespaces](#)
- [Managing Rollback Segments](#)

See Also:

- [Chapter 3, "Using Oracle-Managed Files"](#) for information about creating an undo tablespace whose datafiles are both created and managed by the Oracle database server
- *Oracle9i Real Application Clusters Administration* for information about managing undo space in an Oracle Real Application Clusters environment.

What is Undo?

Every Oracle database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Oracle refers to these records collectively as **undo**.

Undo records are used to:

- Roll back transactions when a `ROLLBACK` statement is issued
- Recover the database
- Provide read consistency

When a rollback statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

Historically, Oracle has used rollback segments to store undo. Space management for these rollback segments has proven to be quite complex. Oracle now offers another method of storing undo that eliminates the complexities of managing rollback segment space, and enables DBAs to exert control over how long undo is retained before being overwritten. This method uses an undo tablespace. Both of these methods of managing undo space are discussed in this chapter.

You cannot use both methods in the same database instance, although for migration purposes it is possible, for example, to create undo tablespaces in a database that is using rollback segments, or to drop rollback segments in a database that is using undo tablespaces. However, you must shut down and restart your database in order to effect the switch to another method of managing undo.

Note: Oracle always uses a `SYSTEM` rollback segment for performing system transactions. There is only one `SYSTEM` rollback segment and it is created automatically at `CREATE DATABASE` time and is always brought online at instance startup. You are not required to perform any operations to manage the `SYSTEM` rollback segment.

See Also: *Oracle9i Database Concepts* for more information about undo and managing undo space

Specifying the Mode for Undo Space Management

If you use the rollback segment method of managing undo space, you are said to be operating in the manual undo management mode. If you use the undo tablespace method, you are operating in the automatic undo management mode. You determine the mode at instance startup using the `UNDO_MANAGEMENT` initialization parameter.

Starting an Instance in Automatic Undo Management Mode

The following initialization parameter setting causes the `STARTUP` command to start an instance in automatic undo management mode:

```
UNDO_MANAGEMENT = AUTO
```

An undo tablespace must be available, into which Oracle will store undo records. The default undo tablespace is created at database creation, or an undo tablespace can be created explicitly. The methods of creating an undo tablespace are explained in "[Creating an Undo Tablespace](#)" on page 13-6

When the instance starts up, Oracle automatically selects for use the first available undo tablespace. If there is no undo tablespace available, the instance starts, but uses the `SYSTEM` rollback segment. This is not recommended in normal circumstances, and an alert message is written to the alert file to warn that the system is running without an undo tablespace.

You can optionally specify at startup that you want an Oracle instance to use a specific undo tablespace. This is done by setting the `UNDO_TABLESPACE` initialization parameter. For example:

```
UNDO_TABLESPACE = undotbs_01
```

In this case, if you have not already created the undo tablespace (in this example, `undotbs_01`), the `STARTUP` command will fail. The `UNDO_TABLESPACE` parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

The following is a summary of the initialization parameters for automatic undo management mode:

Initialization Parameter	Description
UNDO_MANAGEMENT	If <code>AUTO</code> , use automatic undo management mode. If <code>MANUAL</code> , use manual undo management mode.
UNDO_TABLESPACE	A dynamic parameter specifying the name of an undo tablespace to use.
UNDO_RETENTION	A dynamic parameter specifying the length of time to retain undo. Default is 900 seconds.
UNDO_SUPPRESS_ERRORS	If <code>TRUE</code> , suppress error messages if manual undo management SQL statements are issued when operating in automatic undo management mode. If <code>FALSE</code> , issue error message. This is a dynamic parameter.

If the initialization parameter file contains parameters relating to manual undo management, they are ignored.

To learn how to manage undo tablespaces, see "[Managing Undo Tablespaces](#)" on page 13-5.

See Also: *Oracle9i Database Reference* for complete descriptions of initialization parameters used in automatic undo management mode

Starting an Instance in Manual Undo Management Mode

The following initialization parameter setting causes the `STARTUP` command to start an instance in manual undo management mode:

```
UNDO_MANAGEMENT = MANUAL
```

If the `UNDO_MANAGEMENT` initialization parameter is not specified, the instance starts in manual undo management mode. If an `UNDO_TABLESPACE` initialization parameter is found, it is ignored. For DBAs who want to run their databases in manual undo management mode, their existing initialization parameter file can be used without any changes.

When the instance starts up, it brings online a number of rollback segments as determined by either of the following:

- The `ROLLBACK_SEGMENTS` initialization parameter
- The `TRANSACTIONS` and `TRANSACTIONS_PER_ROLLBACK_SEGMENT` initialization parameters

The following is a summary of initialization parameters that can be specified with manual undo management mode.

Initialization Parameter	Description
ROLLBACK_SEGMENTS	Specifies the rollback segments to be acquired at instance startup
TRANSACTIONS	Specifies the maximum number of concurrent transactions
TRANSACTIONS_PER_ROLLBACK_SEGMENT	Specifies the number of concurrent transactions that each rollback segment is expected to handle
MAX_ROLLBACK_SEGMENTS	Specifies the maximum number of rollback segments that can be online for any instance

To learn how to manage rollback segments, see "[Managing Rollback Segments](#)" on page 13-13.

See Also: *Oracle9i Database Reference* for complete descriptions of initialization parameters used in manual undo management mode

Managing Undo Tablespaces

Oracle strongly recommends operating in automatic undo management mode. The database server can manage undo more efficiently, and automatic undo management mode is less complex to implement and manage. The following sections guide you in the management of undo tablespaces:

- [Creating an Undo Tablespace](#)
- [Altering an Undo Tablespace](#)
- [Dropping an Undo Tablespace](#)
- [Switching Undo Tablespaces](#)
- [Establishing User Quotas for Undo Space](#)
- [Specifying the Retention Period for Undo Information](#)
- [Viewing Information About Undo Space](#)

See Also: *Oracle9i SQL Reference* for complete descriptions of the SQL statements discussed in the following sections

Creating an Undo Tablespace

There are two methods of creating an undo tablespace. The first method creates the undo tablespace when the `CREATE DATABASE` statement is issued. This occurs when you are creating a new database, and the instance is started in automatic undo management mode (`UNDO_MANAGEMENT = AUTO`). The second method is used with an existing database. It uses the `CREATE UNDO TABLESPACE` statement.

You cannot create database objects in an undo tablespace. It is reserved for system-managed undo data.

Using `CREATE DATABASE` to Create an Undo Tablespace

You can create a specific undo tablespace using the `UNDO TABLESPACE` clause of the `CREATE DATABASE` statement. But, this clause is not required.

If the `UNDO TABLESPACE` clause is not specified and the `CREATE DATABASE` statement is executed in automatic undo management mode, a default undo tablespace is created with the name `SYS_UNDOTBS`. This tablespace is allocated from the default set of files used by the `CREATE DATABASE` statement and its attributes are determined by Oracle. The initial size is 10M, and it is autoextensible. This method of creating an undo tablespace is only recommended to users who do not have any specific requirements for allocation of undo space.

The following statement illustrates using the `UNDO TABLESPACE` clause in a `CREATE DATABASE` statement. The undo tablespace is named `undotbs_01` and one datafile, `/u01/oracle/rbdb1/undo0101.dbf`, is allocated for it.

```
CREATE DATABASE rbdb1
  CONTROLFILE REUSE
  .
  .
  .
  UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf' ;
```

If the undo tablespace cannot be created successfully during `CREATE DATABASE`, the entire `CREATE DATABASE` operation fails. You must clean up the database files, correct the error and retry the `CREATE DATABASE` operation.

Using the `CREATE UNDO TABLESPACE` Statement

The `CREATE UNDO TABLESPACE` statement is the same as the `CREATE TABLESPACE` statement, but the `UNDO` keyword is specified. Oracle determines most of the attributes of the undo tablespace, you can specify only the `DATAFILE` clause.

This example creates the `undotbs_02` undo tablespace:

```
CREATE UNDO TABLESPACE undotbs_02
  DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

Altering an Undo Tablespace

Undo tablespaces are altered using the `ALTER TABLESPACE` statement. However, since most aspects of undo tablespaces are system managed, you need only be concerned with the following actions:

- Adding a datafile
- Renaming a datafile
- Bringing a datafile online or taking it offline
- Beginning or ending an open backup on a datafile

These are also the only attributes you are permitted to alter.

If an undo tablespace runs out of space, or you want to prevent it from doing so, you can add more files to it or resize existing datafiles.

The following example adds another datafile to undo tablespace `undotbs_01`:

```
ALTER TABLESPACE undotbs_01
  ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND ON NEXT 1M
  MAXSIZE UNLIMITED;
```

You can use the `ALTER DATABASE . . . DATAFILE` statement to resize or extend a datafile.

See Also: ["Changing a Datafile's Size"](#) on page 12-6

Dropping an Undo Tablespace

Use the `DROP TABLESPACE` statement to drop an undo tablespace. The following example drops the undo tablespace `undotbs_01`:

```
DROP TABLESPACE undotbs_01;
```

An undo tablespace can only be dropped if it is not currently used by any instance. If the undo tablespace contains any outstanding transactions (for example, a transaction died but has not yet been recovered), the `DROP TABLESPACE` statement fails. However, since `DROP TABLESPACE` drops an undo tablespace even if it contains unexpired undo information (within retention period), you must be careful

not to drop an undo tablespace if undo information is needed by some existing queries.

`DROP TABLESPACE` for undo tablespaces behaves like `DROP TABLESPACE . . . INCLUDING CONTENTS`. All contents of the undo tablespace are removed.

Switching Undo Tablespaces

You can switch from using one undo tablespace to another. Because the `UNDO_TABLESPACE` initialization parameter is a dynamic parameter, the `ALTER SYSTEM SET` statement can be used to assign a new undo tablespace.

The following statement effectively switches to a new undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

Assuming `undotbs_01` is the current undo tablespace, after this command successfully executes, the instance uses `undotbs_02` in place of `undotbs_01` as its undo tablespace.

If any of the following conditions exist for the tablespace being switched to, an error is reported and no switching occurs:

- The tablespace does not exist,
- The tablespace is not an undo tablespace
- The tablespace is already being used by another instance

The database is online while the switch operation is performed, and user transactions can be executed while this command is being executed. When the switch operation completes successfully, all transactions started after the switch operation began are assigned to transaction tables in the new undo tablespace.

The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a `PENDING OFFLINE` mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

An undo tablespace can exist in this `PENDING OFFLINE` mode, even after the switch operation completes successfully. A `PENDING OFFLINE` undo tablespace cannot be used by another instance, nor can it be dropped. Eventually, after all active transactions have committed, the undo tablespace automatically goes from the `PENDING OFFLINE` mode to the `OFFLINE` mode. From then on, the undo

tablespace is available for other instances (in an Oracle Real Application Cluster environment).

If the parameter value for `UNDO TABLESPACE` is set to '' (two single quotes), the current undo tablespace will be switched out without switching in any other undo tablespace. This can be used, for example, to unassign an undo tablespace in the event that you want to revert to manual undo management mode.

The following example unassigns the current undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

Establishing User Quotas for Undo Space

Oracle's Database Resource Manager can be used to establish user quotas for undo space. The Database Resource Manager directive, `UNDO_POOL`, allows DBAs to limit the amount of undo space consumed by a group of users (resource consumer group).

You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current `UPDATE` transaction generating the redo is terminated. No other members of the consumer group can perform further updates until undo space is freed from the pool.

When no `UNDO_POOL` directive is explicitly defined, users are allowed unlimited undo space.

See Also: [Chapter 27, "Using the Database Resource Manager"](#)

Specifying the Retention Period for Undo Information

Committed undo information normally is lost when its undo space is overwritten by a newer transaction. But for consistent read purposes, long running queries might require old undo information for undoing changes and producing older images of data blocks. The initialization parameter, `UNDO_RETENTION`, provides a means of explicitly specifying the amount of undo information to retain. With a proper setting, long running queries can complete without risk of receiving the "snapshot too old" error.

Setting the `UNDO_RETENTION` Initialization Parameter

Retention is specified in units of seconds, for example 500 seconds. It is persistent and can survive system crashes. That is, undo generated before an instance crash, is retained until its retention time has expired even across restarting the instance.

When the instance is recovered, undo information will be retained based on the current setting of the `UNDO_RETENTION` initialization parameter.

The `UNDO_RETENTION` parameter can be set initially in the initialization parameter file that is used by the `STARTUP` process:

```
UNDO_RETENTION = 10
```

The `UNDO_RETENTION` parameter value can be changed dynamically at any time using the `ALTER SYSTEM` command:

```
ALTER SYSTEM SET UNDO_RETENTION = 5;
```

The effect of the `UNDO_RETENTION` parameter is immediate, but it can only be honored if the current undo tablespace has enough space for the active transactions. If an active transaction requires undo space and the undo tablespace does not have available space, the system starts reusing unexpired undo space. Such action can potentially cause some queries to fail with the "snapshot too old" error.

If the `UNDO_RETENTION` initialization parameter is not specified, the default value is 900 seconds.

Choosing the Retention Period for Flashback Queries

The retention period for undo information is an important factor in the execution of flashback queries. Oracle's flashback query feature enables you to see a consistent version of the database as of a specified time in the past. You can execute queries, or even applications, as of a previous time in the database. The Oracle supplied `DBMS_FLASHBACK` package implements this functionality at the session level. At the object level, flashback queries use the `AS OF` clause of the `SELECT` statement to specify the previous point in time for which you wish to view data.

The retention period determines how far back in time a database version can be established for flashback queries. Specifically, you must choose an undo retention interval that is long enough that it enables you to construct a snapshot of the database for the oldest version of the database that you are interested in. For example, if an application requires that a version of the database be available reflecting its content 12 hours previously, then `UNDO_RETENTION` must be set to 43200.

When using automatic undo management, the `RETENTION` value for `LOB` columns is set to the value of `UNDO_RETENTION`.

See Also:

- *Oracle9i Application Developer's Guide - Fundamentals* for information about using the flashback query feature
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of the `DBMS_FLASHBACK` package
- *Oracle9i SQL Reference* for a description of the `AS OF` clause of the `SELECT` statement

Calculating the Space Requirements For Undo Retention

Given a specific `UNDO_RETENTION` parameter setting and some system statistics, the amount of undo space required to satisfy the undo retention requirement can be estimated using the following formula:

$$\text{UndoSpace} = \text{UR} * \text{UPS} + \text{overhead}$$

where:

- `UndoSpace` is the number of undo blocks
- `UR` is `UNDO_RETENTION` in seconds
- `UPS` is undo blocks for each second
- `overhead` is the small overhead for metadata (transaction tables, bitmaps, and so forth)

As an example, if `UNDO_RETENTION` is set to 2 hours, and the transaction rate (`UPS`) is 200 undo blocks for each second, with a 4K block size, the required undo space is computed as follows:

$$(2 * 3600 * 200 * 4K) = 5.8\text{GBs.}$$

Such computation can be performed by using information in the `V$UNDOSTAT` view. In the steady state, you can query the view to obtain the transaction rate. The overhead figure can also be obtained from the view.

Viewing Information About Undo Space

This section lists views that are useful for viewing information about undo space in the automatic undo management mode. In addition to views listed here, you can obtain information from the views available for viewing tablespace and datafile information.

See Also:

- ["Viewing Tablespace Information"](#) on page 11-50
- ["Viewing Datafile Information"](#) on page 12-28

Undo Space Views

The following views are available for obtaining undo space information:

View	Description
V\$UNDOSTAT	Contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. Oracle also uses this information to help tune undo usage in the system. This view is available in both the automatic undo management and the manual undo management modes.
V\$ROLLSTAT	For automatic undo management mode, information reflects behavior of the undo segments in the undo tablespace
V\$TRANSACTION	Contains undo segment information
DBA_UNDO_EXTENTS	Shows the commit time for each extent in the undo tablespace.

See Also: *Oracle9i Database Reference* for complete descriptions of the views used in automatic undo management mode

Monitoring Undo Space

The V\$UNDOSTAT view is useful for monitoring the effects of transaction execution on undo space in the current instance. Statistics are available for undo space consumption, transaction concurrency, and length of queries in the instance.

Each row in the view contains statistics collected in the instance for a ten-minute interval. The rows are in descending order by the BEGIN_TIME column value. Each row belongs to the time interval marked by (BEGIN_TIME, END_TIME). Each column represents the data collected for the particular statistic in that time interval. The first row of the view contains statistics for the (partial) current time period. The view contains a total of 1008 rows, spanning a 7 day cycle.

The following example shows the results of a query on the V\$UNDOSTAT view.

```
SELECT BEGIN_TIME, END_TIME, UNDOTSN, UNDOBLKS, TXNCOUNT,
       MAXCONCURRENCY AS "MAXCON"
```

```
FROM V$UNDOSTAT;
```

The results are:

BEGIN_TIME	END_TIME	UNDOTSN	UNDOBLKS	TXNCOUNT	MAXCON
07/28/2000 18:26:28	07/28/2000 18:32:13	2	709	55	2
07/28/2000 18:16:28	07/28/2000 18:26:28	2	448	12	2
07/28/2000 14:36:28	07/28/2000 18:16:28	1	0	0	0
07/28/2000 14:26:28	07/28/2000 14:36:28	1	1	1	1
07/28/2000 14:16:28	07/28/2000 14:26:28	1	10	1	1
...					

The above example shows how undo space is consumed in the system for the previous 24 hours from the time 18:32:13.

Managing Rollback Segments

If you choose to use rollback segments to store undo, the following sections guide you in their management:

- [Guidelines for Managing Rollback Segments](#)
- [Creating Rollback Segments](#)
- [Altering Rollback Segments](#)
- [Explicitly Assigning a Transaction to a Rollback Segment](#)
- [Dropping Rollback Segments](#)
- [Viewing Rollback Segment Information](#)

Note: The use of rollback segments for managing undo space will be deprecated in a future release. Oracle strongly recommends that you use automatic undo management and manage undo space using an UNDO_TABLESPACE.

Guidelines for Managing Rollback Segments

This section describes guidelines to consider before creating or managing the rollback segments of your databases, and contains the following topics:

- [Use Multiple Rollback Segments](#)

- [Choose Between Public and Private Rollback Segments](#)
- [Specify Rollback Segments to Acquire Automatically](#)
- [Approximate Rollback Segment Sizes](#)
- [Create Rollback Segments with Many Equally Sized Extents](#)
- [Set an Optimal Number of Extents for Each Rollback Segment](#)
- [Place Rollback Segments in a Separate Tablespace](#)

See Also: *Oracle9i Database Concepts* for additional information about rollback segments

Use Multiple Rollback Segments

Using multiple rollback segment distributes rollback segment contention across many segments and improves system performance. Oracle assigns transactions to rollback segments in round-robin fashion. This results in a fairly even distribution of the number of transactions for each rollback segment. It is also possible to assign a transaction to a specific rollback segment, but this is usually not done.

When a database is created, a single rollback segment named `SYSTEM` is created in the `SYSTEM` tablespace. This rollback segment is used in special ways by the Oracle database server, and is not intended for general use. Before you write to objects created in non-`SYSTEM` tablespaces, you must create and bring online at least one additional rollback segment in a non-`SYSTEM` tablespace.

Note: When you are initially creating the database, and in order to create additional tablespaces and rollback segments, you must create a second rollback segment in the `SYSTEM` tablespace. Once these additional rollback segments are created, you should activate the new rollback segments and make the second rollback segment unavailable.

At startup, an instance always acquires (brings online) the `SYSTEM` rollback segment in addition to any other rollback segments it needs or is directed to acquire. When there are multiple rollback segments, Oracle tries to use the `SYSTEM` rollback segment only for special system transactions and distributes user transactions among other rollback segments. If there are too many transactions for the non-`SYSTEM` rollback segments, Oracle uses the `SYSTEM` segment; plan your number of rollback segments to avoid this.

There are a couple of options for activating multiple rollback segments when you start up an instance:

- Use public rollback segments and include the `TRANSACTIONS` and `TRANSACTIONS_PER_ROLLBACK_SEGMENT` initialization parameters in your initialization parameter file
- Use private or public rollback segments and specify their names in the `ROLLBACK_SEGMENTS` initialization parameter

These options are discussed in other guidelines that follow.

There is a limit on the number of rollback segments that can be open simultaneously. This limit is set by the `MAX_ROLLBACK_SEGMENTS` initialization parameter. Ensure that this parameter is set to a value higher than the number of rollback segments specified in the `ROLLBACK_SEGMENTS` initialization parameter.

See Also: *Oracle9i Database Reference* for additional information about the `TRANSACTIONS`, `TRANSACTIONS_PER_ROLLBACK_SEGMENT`, and `ROLLBACK_SEGMENT` initialization parameters

Choose Between Public and Private Rollback Segments

A **private rollback segment** must be acquired explicitly by an instance. This can occur at database startup when the rollback segments name is included in the `ROLLBACK_SEGMENTS` parameter in the initialization parameter file. A private rollback segment can also be acquired by specifically bringing it online by manually issuing the statement to do so. In an Oracle Real Application Clusters environment, private rollback segments allow an instance to acquire specific rollback segments.

Public rollback segments form a pool of rollback segments that any instance requiring a rollback segment can use. An instance decides how many of these rollback segments to automatically acquire at instance startup based on the values of the `TRANSACTIONS` and `TRANSACTIONS_PER_ROLLBACK_SEGMENT` initialization parameters. Public rollback segments can be shared between Oracle Real Application Cluster instances.

If you are not using the Oracle9i Real Application Clusters feature, private and public rollback segments function similarly.

Specify Rollback Segments to Acquire Automatically

When many transactions are concurrently proceeding, they simultaneously generate rollback information. A way of specifying that an appropriate number of rollback segments be acquired automatically at instance startup is to include the

`TRANSACTIONS` and `TRANSACTIONS_PER_ROLLBACK_SEGMENT` initialization parameters. You must also be using public rollback segments.

You can indicate the number of concurrent transactions you expect for the instance with the initialization parameter `TRANSACTIONS`, and the number of transactions you expect each rollback segment will need to handle with the initialization parameter `TRANSACTIONS_PER_ROLLBACK_SEGMENT`. Then, when an instance opens a database, it attempts to acquire at least n rollback segments, where $n = \text{TRANSACTIONS} / \text{TRANSACTIONS_PER_ROLLBACK_SEGMENT}$. When creating your database, or subsequently, you should have created at least n public rollback segments.

If you choose to use private rollback segments, these rollback segments will be acquired automatically by an instance at startup if you specify the rollback segments by name in the `ROLLBACK_SEGMENTS` initialization parameter in the instance's parameter file.

If you use both private and public rollback segments the following might occur. An instance acquires all the rollback segments listed in the `ROLLBACK_SEGMENTS` initialization parameter, even if more than $\text{TRANSACTIONS} / \text{TRANSACTIONS_PER_ROLLBACK_SEGMENT}$ segments are specified.

Approximate Rollback Segment Sizes

Total rollback segment size should be set based on the size of the most common transactions issued against a database. In general, short transactions experience better performance when the database has many smaller rollback segments, while long-running transactions, like batch jobs, perform better with larger rollback segments. Generally, rollback segments can handle transactions of any size easily. However, in extreme cases when a transaction is either very short or very long, a user might want to use an appropriately sized rollback segment.

If a system is running only short transactions, rollback segments should be small so that they are always cached in main memory. If the rollback segments are small enough, they are more likely to be cached in the SGA according to the LRU algorithm, and database performance is improved because less disk I/O is necessary. The main disadvantage of small rollback segments is the increased likelihood of the error "snapshot too old" when running a long query involving records that are frequently updated by other transactions. This error occurs because the rollback entries needed for read consistency are overwritten as other update entries wrap around the rollback segment. Consider this issue when designing an application's transactions, and make them short atomic units of work so that you can avoid this problem.

In contrast, long-running transactions work better with larger rollback segments, because the rollback entries for a long-running transaction can fit in preallocated extents of a large rollback segment.

When database systems applications concurrently issue a mix of very short and very long transactions, performance can be optimized if transactions are explicitly assigned to a rollback segment based on the transaction/rollback segment size. You can minimize dynamic extent allocation and truncation for rollback segments. This is not required for most systems and is intended for extremely large or small transactions.

To optimize performance when issuing a mix of extremely small and large transactions, make a number of rollback segments of appropriate size for each type of transaction (such as small, medium, and large). Most rollback segments should correspond to the typical transactions, with a fewer number of rollback segments for the atypical transactions. Then set `OPTIMAL` for each such rollback segment so that the rollback segment returns to its intended size if it has to grow.

You should tell users about the different sets of rollback segments that correspond to the different types of transactions. Often, it is *not* beneficial to assign a transaction explicitly to a specific rollback segment. However, you can assign an atypical transaction to an appropriate rollback segment created for such transactions. For example, you can assign a transaction that contains a large batch job to a large rollback segment.

When a mix of transactions is not prevalent, each rollback segment should be 10% of the size of the database's largest table because most SQL statements affect 10% or less of a table. A rollback segment of this size should be sufficient to store the actions performed by most SQL statements.

Generally speaking, you should set a high `MAXEXTENTS` for rollback segments. This allows a rollback segment to allocate subsequent extents as it needs them.

Create Rollback Segments with Many Equally Sized Extents

Each rollback segment's total allocated space should be divided among many equally sized extents. In general, optimal rollback I/O performance is observed if each rollback segment for an instance has 10 to 20 equally sized extents.

After determining the desired total initial size of a rollback segment and the number of initial extents for the segment, use the following formula to calculate the size (*s*) of each extent of the rollback segment:

$$s = T / n$$

where:

s = calculated size, in bytes, of each extent initially allocated

T = total initial rollback segment size, in bytes

n = number of extents initially allocated

After s is calculated, create the rollback segment and specify the storage parameters `INITIAL` and `NEXT` as s , and `MINEXTENTS` to n . `PCTINCREASE` cannot be specified for rollback segments and therefore defaults to 0. Also, if the size s of an extent is not an exact multiple of the data block size, it is rounded up to the next multiple.

Set an Optimal Number of Extents for Each Rollback Segment

You should carefully assess the kind of transactions the system runs when setting the `OPTIMAL` parameter for each rollback segment. For a system that executes long-running transactions frequently, `OPTIMAL` should be large so that Oracle does not have to shrink and allocate extents frequently. Also, for a system that executes long queries on active data, `OPTIMAL` should be large to avoid "snapshot too old" errors. `OPTIMAL` should be smaller for a system that mainly executes short transactions and queries so that the rollback segments remain small enough to be cached in memory, thus improving system performance.

The `V$ROLLNAME` and `V$ROLLSTAT` dynamic performance views can be monitored to collect statistics useful in determining appropriate settings for `OPTIMAL`. See ["Monitoring Rollback Segment Statistics"](#) on page 13-27.

Place Rollback Segments in a Separate Tablespace

If possible, create one or more tablespaces specifically to hold all rollback segments. This way, all rollback segment data is stored separately from other types of data. Creating this "rollback segment" tablespace can provide the following benefits:

- A tablespace holding rollback segments can always be kept online, thus maximizing the combined storage capacity of rollback segments at all times. If some rollback segments are not available, the overall database operation can be affected.
- Because tablespaces with active rollback segments cannot be taken offline, designating a tablespace to hold all rollback segments of a database ensures that the data stored in other tablespaces can be taken offline without concern for the database's rollback segments.
- A tablespace's free extents are likely to be more fragmented if the tablespace contains rollback segments that frequently allocate and deallocate extents.

Creating Rollback Segments

To create rollback segments, you must have the `CREATE ROLLBACK SEGMENT` system privilege. You use the `CREATE ROLLBACK SEGMENT` statement. The tablespace to contain the new rollback segments must be online. Rollback segments are usually created as part of the database creation script or process, but you may add more at a later time.

The following topics relating to creating rollback segments are contained in this section:

- [The CREATE ROLLBACK SEGMENT Statement](#)
- [Bringing New Rollback Segments Online](#)
- [Setting Storage Parameters When Creating a Rollback Segment](#)

The CREATE ROLLBACK SEGMENT Statement

The following statement creates a rollback segment named `rbs_02` in the `rbsspace` tablespace, using the default storage parameters of that tablespace. Since this is not an Oracle Real Application Clusters environment, it is not necessary to specify `PRIVATE` or `PUBLIC`. The default is `PRIVATE`.

```
CREATE ROLLBACK SEGMENT rbs_02 TABLESPACE rbsspace;
```

See Also: *Oracle9i SQL Reference* for exact syntax, restrictions, and authorization requirements for the SQL statements used in managing rollback segments

Bringing New Rollback Segments Online

New rollback segments are initially offline. You must issue an `ALTER ROLLBACK SEGMENT` statement to bring them online and make them available for use by transactions of an instance. This is described in "[Changing the ONLINE/OFFLINE Status of Rollback Segments](#)" on page 13-22.

If you create a private rollback segment, add the name of this new rollback segment to the `ROLLBACK_SEGMENTS` initialization parameter in the initialization parameter file for the database. Doing so enables the private rollback segment to be acquired automatically by the instance at instance startup. For example, if two new private rollback segments are created and named `rbs_01` and `rbs_02`, then the `ROLLBACK_SEGMENTS` initialization parameter can be specified as follows:

```
ROLLBACK_SEGMENTS = (rbs_01, rbs_02)
```

Setting Storage Parameters When Creating a Rollback Segment

Suppose you wanted to create a rollback segment `rbs_01` with storage parameters and optimal size set as follows:

- The rollback segment is allocated an initial extent of 100K.
- The rollback segment is allocated the second extent of 100K.
- The optimal size of the rollback segment is 4M.
- The minimum number of extents and the number of extents initially allocated when the segment is created is 20.
- The maximum number of extents that the rollback segment can allocate, including the initial extent, is 100.

The following statement creates a rollback segment with these characteristics:

```
CREATE ROLLBACK SEGMENT rbs_01
        TABLESPACE rbsspace
        STORAGE (
            INITIAL 100K
            NEXT 100K
            OPTIMAL 4M
            MINEXTENTS 20
            MAXEXTENTS 100 );
```

You cannot set a value for the storage parameter `PCTINCREASE`. It is always 0 for rollback segments. The `OPTIMAL` storage parameter is unique to rollback segments. For a discussion of storage parameters see ["Setting Storage Parameters"](#) on page 14-8.

Oracle Corporation makes the following recommendations:

- Set `INITIAL` and `NEXT` to the same value to ensure that all extents are the same size.
- Create a large number of initial extents to minimize the possibility of dynamic extension. `MINEXTENTS = 20` is a good value.
- Avoid setting `MAXEXTENTS = UNLIMITED` as this could cause unnecessary extension of a rollback segment and possibly of data files due to a programming error. If you do specify `UNLIMITED`, be aware that extents for that segment must have a minimum of four data blocks. Also, if you later want to convert a rollback segment whose `MAXEXTENTS` are limited to `UNLIMITED`, that rollback segment cannot be converted if it has less than four data blocks in any extent. If you want to convert from limited to `UNLIMITED`, and have less than four data

blocks in an extent, your only choice is to drop and re-create the rollback segment.

See Also: *Oracle9i SQL Reference* for a detailed description of storage parameters

Altering Rollback Segments

This section discusses various actions you can take to maintain your rollback segments. All of these maintenance activities use the `ALTER ROLLBACK SEGMENT` statement. You must have the `ALTER ROLLBACK SEGMENT` system privilege to use this statement.

The following topics are discussed:

- [Changing Rollback Segment Storage Parameters](#)
- [Shrinking a Rollback Segment Manually](#)
- [Changing the ONLINE/OFFLINE Status of Rollback Segments](#)

Changing Rollback Segment Storage Parameters

You can change some of a rollback segment's storage parameters after creating it. You may want to change the values of `OPTIMAL` or `MAXEXTENTS`. The following statement alters the maximum number of extents that the `rbbs_01` rollback segment can allocate:

```
ALTER ROLLBACK SEGMENT rbbs_01
  STORAGE (MAXEXTENTS 120);
```

You can alter the settings for the `SYSTEM` rollback segment, including the `OPTIMAL` parameter, just as you can alter those of any rollback segment.

Shrinking a Rollback Segment Manually

You can manually decrease the size of a rollback segment using the `ALTER ROLLBACK SEGMENT` statement. The rollback segment you are trying to shrink must be online.

The following statement shrinks rollback segment `rbbs1` to 100K:

```
ALTER ROLLBACK SEGMENT rbbs1 SHRINK TO 100K;
```

This statement attempts to reduce the size of the rollback segment to the specified size, but stops short if an extent cannot be deallocated because it is active.

Changing the **ONLINE/OFFLINE** Status of Rollback Segments

This section describes aspects of bringing rollback segments online and taking them offline, and contains the following topics:

- [Bringing Rollback Segments Online Manually](#)
- [Bringing Rollback Segment Online Automatically](#)
- [Taking Rollback Segments Offline](#)

A rollback segment is either *online* and available to transactions, or *offline* and unavailable to transactions. Generally, rollback segments are online and available for use by transactions.

You may want to take online rollback segments offline in the following situations:

- You want to take a tablespace offline, and the tablespace contains rollback segments. You cannot take a tablespace offline if it contains rollback segments that transactions are currently using. To prevent associated rollback segments from being used, you can take them offline before taking the tablespace offline.
- You want to drop a rollback segment, but cannot because transactions are currently using it. To prevent the rollback segment from being used, you can take it offline before dropping it.

Note: You cannot take the `SYSTEM` rollback segment offline.

You might later want to bring an offline rollback segment back online so that transactions can use it. When a rollback segment is created, it is initially offline, and you must explicitly bring a newly created rollback segment online before it can be used by an instance's transactions. You can bring an offline rollback segment online using any instance accessing the database that contains the rollback segment.

Bringing Rollback Segments Online Manually You can only bring a rollback segment online if its current status (as shown in the `DBA_ROLLBACK_SEGS` data dictionary view) is `OFFLINE` or `PARTLY AVAILABLE`. To bring an offline rollback segment online, use the `ALTER ROLLBACK SEGMENT` statement with the `ONLINE` option.

The following statement brings the rollback segment `user_rs_2` online:

```
ALTER ROLLBACK SEGMENT user_rs_2 ONLINE;
```

After you bring a rollback segment online, its status in the data dictionary view `DBA_ROLLBACK_SEGS` is `ONLINE`. To see a query for checking rollback segment status, see "[Displaying Rollback Segment Information](#)" on page 13-26.

A rollback segment in the `PARTLY AVAILABLE` state contains data for an in-doubt or recovered distributed transaction, or for yet to be recovered transactions. You can view its status in the data dictionary view `DBA_ROLLBACK_SEGS` as `PARTLY AVAILABLE`. The rollback segment usually remains in this state until the transaction is resolved either automatically by `RECO`, or manually by a DBA.

You might find that all rollback segments are `PARTLY AVAILABLE`. In this case, you can bring the `PARTLY AVAILABLE` segment online. Some resources used by the rollback segment for the in-doubt transaction remain inaccessible until the transaction is resolved. As a result, the rollback segment may have to grow if other transactions assigned to it need additional space.

As an alternative to bringing a `PARTLY AVAILABLE` segment online, you might find it more efficient to create a new rollback segment temporarily, until the in-doubt transaction is resolved.

Bringing Rollback Segment Online Automatically If you would like a rollback segment to be automatically brought online whenever you start up the database, add the segment's name to the `ROLLBACK_SEGMENTS` parameter in the database's parameter file. Or, you can use public rollback segments and use the `TRANSACTIONS` and `TRANSACTIONS_PER_ROLLBACK_SEGMENT` initialization parameters.

These options are discussed in "[Specify Rollback Segments to Acquire Automatically](#)" on page 13-15.

Taking Rollback Segments Offline To take an online rollback segment offline, use the `ALTER ROLLBACK SEGMENT` statement with the `OFFLINE` option. The rollback segment's status in the `DBA_ROLLBACK_SEGS` data dictionary view must be `ONLINE`, and the rollback segment must be acquired by the current instance.

The following example takes the rollback segment `user_rs_2` offline:

```
ALTER ROLLBACK SEGMENT user_rs_2 OFFLINE;
```

If you attempt to take a rollback segment that does not contain active rollback entries offline, Oracle immediately takes the segment offline and changes its status to `OFFLINE`.

In contrast, if you try to take a rollback segment that contains rollback data for active transactions (local, remote, or distributed) offline, Oracle makes the rollback

segment unavailable to future transactions and takes it offline after all the active transactions using the rollback segment complete. Until the transactions complete, the rollback segment cannot be brought online by any instance other than the one that was trying to take it offline.

During this period that the rollback segment is waiting to go offline, the rollback segment's status in the view `DBA_ROLLBACK_SEGS` remains `ONLINE`. However, the rollback segment's status in the view `V$ROLLSTAT` is `PENDING OFFLINE`. For information on viewing rollback segment status, see "[Displaying Rollback Segment Information](#)" on page 13-26.

The instance that tried to take a rollback segment offline and caused it to change to `PENDING OFFLINE` can bring it back online at any time. If the rollback segment is brought back online, it functions normally.

After you take a public or private rollback segment offline, it remains offline until you explicitly bring it back online *or* you restart the instance.

Explicitly Assigning a Transaction to a Rollback Segment

A transaction can be explicitly assigned to a specific rollback segment. Reasons for doing this include:

- You can predict the amount of rollback information generated by a transaction. You can assign the transaction to a rollback segment where you know that the rollback information will fit in the current extents of the segment. Thus, you can reduce the overhead of additional extents being dynamically allocated, and subsequently truncated.
- You know that no long running queries are concurrently reading the same tables, so if you assign small transactions to small rollback segments, those segments will most likely remain in memory.
- You have transactions that modify tables that are concurrently being read by long-running queries. You can assign these transactions to large rollback segments so that the rollback information needed for the read-consistent queries is not overwritten.

To assign a transaction to a rollback segment explicitly, use the `SET TRANSACTION` statement with the `USE ROLLBACK SEGMENT` clause. The rollback segment must be online for the current instance, and the `SET TRANSACTION USE ROLLBACK SEGMENT` statement must be the first statement of the transaction. If a specified rollback segment is not online or a `SET TRANSACTION USE ROLLBACK SEGMENT` clause is not the first statement in a transaction, an error is returned.

For example, if you are about to begin a transaction that contains a significant amount of work (more than most transactions), you can assign the transaction to a large rollback segment, as follows:

```
SET TRANSACTION USE ROLLBACK SEGMENT large_rs1;
```

After the transaction is committed, Oracle automatically assigns the next transaction to any available rollback segment unless the new transaction is explicitly assigned to a specific rollback segment by the user.

Dropping Rollback Segments

You can drop rollback segments when the extents of a segment become too fragmented on disk, or the segment needs to be relocated in a different tablespace. Before dropping a rollback segment, make sure that the status of the rollback segment is `OFFLINE`. If the rollback segment that you want to drop is any other status, you cannot drop it. If the status is `INVALID`, the segment has already been dropped.

To drop a rollback segment, use the `DROP ROLLBACK SEGMENT` statement. You must have the `DROP ROLLBACK SEGMENT` system privilege. The following statement drops the `rbs1` rollback segment:

```
DROP ROLLBACK SEGMENT rbs1;
```

Note: If a rollback segment specified in `ROLLBACK_SEGMENTS` is dropped, be sure to edit the parameter files of the database to remove the name of the dropped rollback segment from the list in the `ROLLBACK_SEGMENTS` parameter. If this step is not performed before the next instance startup, startup fails because it cannot acquire the dropped rollback segment.

After a rollback segment is dropped, its status changes to `INVALID`. The next time a rollback segment is created, it takes the row vacated by a dropped rollback segment, if one is available, and the dropped rollback segment's row no longer appears in the `DBA_ROLLBACK_SEGS` view.

Viewing Rollback Segment Information

This section presents views that can be used to obtain and monitor rollback segment information, and provides information and examples relating to their use.

The following topics are included:

- [Rollback Segment Views](#)
- [Displaying Rollback Segment Information](#)
- [Monitoring Rollback Segment Statistics](#)
- [Displaying All Rollback Segments](#)
- [Displaying Whether a Rollback Segment Has Gone Offline](#)

See Also: *Oracle9i Database Reference* for more information about the data dictionary views discussed in this chapter

Rollback Segment Views

The following views are useful for displaying information about rollback segments:

View	Description
DBA_ROLLBACK_SEGS	Describes the rollback segments, including names and tablespaces
DBA_SEGMENTS	Identifies a segment as a rollback segment and contains additional segment information
V\$ROLLNAME	Lists the names of all online rollback segments
V\$ROLLSTAT	Contains rollback segment statistics
V\$TRANSACTION	Contains undo segment information

Displaying Rollback Segment Information

The DBA_ROLLBACK_SEGS data dictionary view stores information about the rollback segments of a database. For example, the following query lists the name, associated tablespace, and status of each rollback segment in a database:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, STATUS
       FROM DBA_ROLLBACK_SEGS;
```

```
SEGMENT_NAME  TABLESPACE_NAME  STATUS
-----
SYSTEM        SYSTEM             ONLINE
PUBLIC_RS     SYSTEM             ONLINE
USERS_RS      USERS              ONLINE
```

In addition, the following data dictionary views contain information about the segments of a database, including rollback segments:

- USER_SEGMENTS
- DBA_SEGMENTS

Monitoring Rollback Segment Statistics

The `V$ROLLSTAT` dynamic performance view can be queried to monitor rollback segment statistics. It must be joined with the `V$ROLLNAME` view to map its segment number to its name.

Some specific columns of interest in the `V$ROLLSTAT` view include:

Name	Description
USN	Rollback segment number. If this view is joined with the <code>V\$ROLLNAME</code> view, the rollback segment name can be determined.
WRITES	The number of bytes of entries written to the rollback segment.
XACTS	The number of active transactions.
GETS	The number of rollback segment header requests.
WAITS	The number of rollback segment header requests that resulted in waits.
OPTSIZE	The value of the optimal parameter for the rollback segment.
HWMSIZE	The highest value (high water mark), in bytes, of the rollback segment size reached during usage.
SHRINKS	The number of shrinks that the rollback segment has had to perform in order to stay at the optimal size.
WRAPS	The number of times a rollback segment entry has wrapped from one extent to another.
EXTENDS	The number of times that the rollback segment had to acquire a new extent.
AVESHRINK	The average number of bytes freed during a shrink.
AVEACTIVE	The average number of bytes in active extents in the rollback segment, measured over time.

These statistics are reset at system startup.

Ad hoc querying of this view can help in determining the most advantageous setting for the `OPTIMAL` parameter. Assuming that an instance has equally sized rollback segments with comparably sized extents, `OPTIMAL` for a given rollback segment should be set slightly higher than `AVEACTIVE`. The following chart provides additional information on how to interpret the statistics given in this view.

SHRINKS	AVESHRINK	Analysis and Recommendation
Low	Low	If <code>AVEACTIVE</code> is close to <code>OPTSIZE</code> , then the <code>OPTIMAL</code> setting is correct. Otherwise, <code>OPTIMAL</code> is too large (not many shrinks are being performed.)
Low	High	Excellent: a good setting for <code>OPTIMAL</code> .
High	Low	<code>OPTIMAL</code> is too small: too many shrinks are being performed.
High	High	Periodic long transactions are probably causing these statistics. Set the <code>OPTIMAL</code> parameter higher until <code>SHRINKS</code> is low.

Displaying All Rollback Segments

The following query returns the name of each rollback segment, the tablespace that contains it, and its size:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
FROM DBA_SEGMENTS
WHERE SEGMENT_TYPE = 'ROLLBACK';
```

SEGMENT_NAME	TABLESPACE_NAME	BYTES	BLOCKS	EXTENTS
SYSTEM	SYSTEM	409600	200	8
RB_TEMP	SYSTEM	1126400	550	11
RB1	RBS	614400	300	3
RB2	RBS	614400	300	3
RB3	RBS	614400	300	3
RB4	RBS	614400	300	3
RB5	RBS	614400	300	3
RB6	RBS	614400	300	3
RB7	RBS	614400	300	3
RB8	RBS	614400	300	3

10 rows selected.

Displaying Whether a Rollback Segment Has Gone Offline

When you take a rollback segment offline, it does not actually go offline until all active transactions in it have completed. Between the time when you attempt to take it offline and when it actually is offline, its status in `V$ROLLSTAT` is `PENDING OFFLINE` and it is not used for new transactions. To determine whether any rollback segments for an instance are in this state, use the following query:

```
SELECT NAME, XACTS "ACTIVE TRANSACTIONS"
       FROM V$ROLLNAME, V$ROLLSTAT
       WHERE STATUS = 'PENDING OFFLINE'
          AND V$ROLLNAME.USN = V$ROLLSTAT.USN;
```

```
NAME          ACTIVE TRANSACTIONS
-----
RS2                      3
```

If your instance is part of an Oracle Real Application Clusters configuration, this query displays information for rollback segments of the current instance only, not those of other instances.

Part III

Schema Objects

Part III describes the creation and maintenance of schema objects in the Oracle database. It includes the following chapters:

- [Chapter 14, "Managing Space for Schema Objects"](#)
- [Chapter 15, "Managing Tables"](#)
- [Chapter 16, "Managing Indexes"](#)
- [Chapter 17, "Managing Partitioned Tables and Indexes"](#)
- [Chapter 18, "Managing Clusters"](#)
- [Chapter 19, "Managing Hash Clusters"](#)
- [Chapter 20, "Managing Views, Sequences, and Synonyms"](#)
- [Chapter 21, "General Management of Schema Objects"](#)
- [Chapter 22, "Detecting and Repairing Data Block Corruption"](#)

Managing Space for Schema Objects

This chapter offers guidelines for managing space for schema objects. It contains the following topics:

- [Managing Space in Data Blocks](#)
- [Setting Storage Parameters](#)
- [Managing Resumable Space Allocation](#)
- [Deallocating Space](#)
- [Understanding Space Use of Datatypes](#)

You should familiarize yourself with the concepts in this chapter before attempting to manage specific schema objects as described in later chapters.

Managing Space in Data Blocks

This section describes aspects of managing space in data blocks. Data blocks are the finest level of granularity of the structure in which database data is stored on disk. The size of a data block is specified (or defaulted) at database creation.

The `PCTFREE` and `PCTUSED` parameters are physical attributes that can be specified when a schema object is created or altered. These parameters allow you to control the use of the free space within a data block. This free space is available for inserts and updates of rows of data.

The `PCTFREE` and `PCTUSED` parameters allow you to:

- Improve performance when writing and retrieving data
- Decrease the amount of unused space in data blocks
- Decrease the amount of row chaining between data blocks

The `INITRANS` and `MAXTRANS` parameters are also physical attributes that can be specified when schema objects are created or altered. These parameters control the number of concurrent update transactions allocated for data blocks of a schema object, which in turn affects space usage in data block headers and can have an impact upon data block free space.

The following topics are contained in this section:

- [Specifying the PCTFREE Parameter](#)
- [Specifying the PCTUSED Parameter](#)
- [Selecting Associated PCTUSED and PCTFREE Values](#)
- [Specifying the Transaction Entry Parameters: INITRANS and MAXTRANS](#)

See Also:

- *Oracle9i Database Concepts* for more information on data blocks
- *Oracle9i SQL Reference* for syntax and other details of the `PCTFREE`, `PCTUSED`, `INITRANS`, and `MAXTRANS` physical attributes parameters

Specifying the PCTFREE Parameter

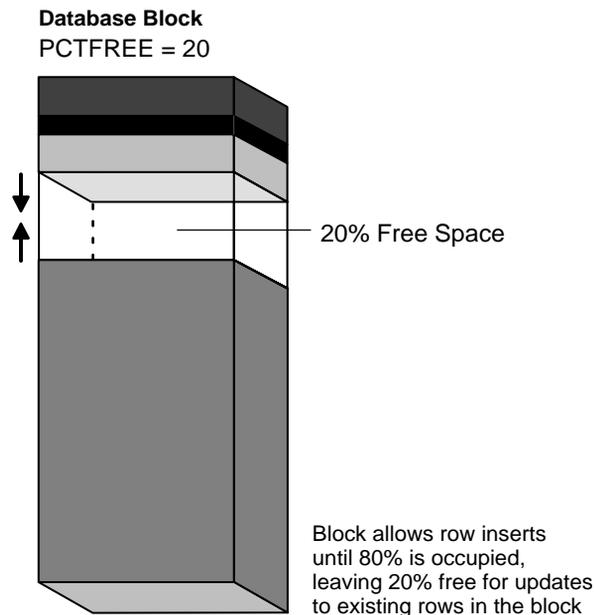
The `PCTFREE` parameter is used to set the percentage of a block to be reserved for possible updates to rows that already are contained in that block. For example,

assume that you specify the following parameter within a `CREATE TABLE` statement:

```
PCTFREE 20
```

This indicates that 20% of each data block used for this table's data segment will be kept free and available for possible updates to the existing rows already within each block. [Figure 14-1](#) illustrates `PCTFREE`.

Figure 14-1 `PCTFREE`



Notice that before the block reaches `PCTFREE`, the free space of the data block is filled by both the insertion of new rows and by the growth of the data block header.

Ensure that you understand the nature of a table or index data before setting `PCTFREE`. Updates can cause rows to grow. New values might not be the same size as values they replace. If there are many updates in which data values get larger, `PCTFREE` should be increased. If updates to rows do not affect the total row width, `PCTFREE` can be low. Your goal is to find a satisfactory trade-off between densely packed data and good update performance.

The default for `PCTFREE` is 10 percent. You can use any integer between 0 and 99, inclusive, as long as the sum of `PCTFREE` and `PCTUSED` does not exceed 100.

Effects of Specifying a Smaller `PCTFREE`

A smaller `PCTFREE` has the following effects:

- Reserves less room for updates to expand existing table rows
- Allows inserts to fill the block more completely
- May save space, because the total data for a table or index is stored in fewer blocks (more rows or entries for each block)

A small `PCTFREE` might be suitable, for example, for a segment that is rarely changed.

Effects of Specifying a Larger `PCTFREE`

A larger `PCTFREE` has the following effects:

- Reserves more room for future updates to existing table rows
- May require more blocks for the same amount of inserted data (inserting fewer rows for each block)
- May improve update performance, because Oracle does not need to chain row pieces as frequently, if ever

A large `PCTFREE` is suitable, for example, for segments that are frequently updated.

`PCTFREE` for Nonclustered Tables

If the data in the rows of a nonclustered table is likely to increase in size over time, reserve some space for these updates. Otherwise, updated rows are likely to be chained among blocks.

`PCTFREE` for Clustered Tables

The discussion for nonclustered tables also applies to clustered tables. However, if `PCTFREE` is reached, new rows from *any* table contained in the same cluster key go into a new data block that is chained to the existing cluster key.

`PCTFREE` for Indexes

You can specify `PCTFREE` only when initially creating an index.

Specifying the PCTUSED Parameter

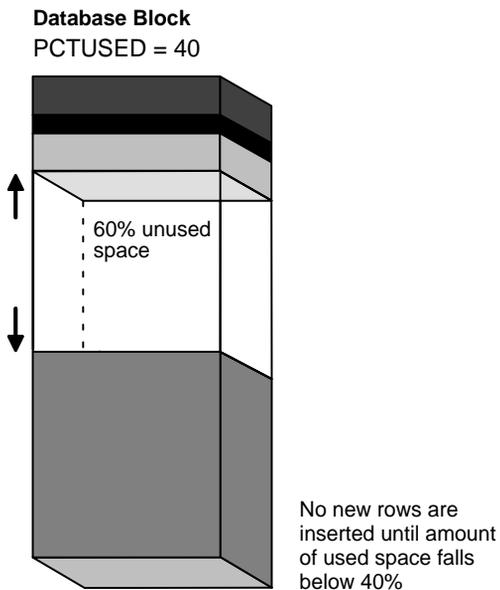
Note: The `PCTUSED` parameter is ignored for objects created in locally managed tablespaces with segment space management specified as `AUTO`. This form of segment space management is discussed in ["Specifying Segment Space Management in Locally Managed Tablespaces"](#) on page 11-8.

After a data block becomes full as determined by `PCTFREE`, Oracle does not consider the block for the insertion of new rows until the percentage of the block being used falls below the parameter `PCTUSED`. Before this value is achieved, Oracle uses the free space of the data block only for updates to rows already contained in the data block. For example, assume that you specify the following parameter within a `CREATE TABLE` statement:

```
PCTUSED 40
```

In this case, a data block used for this table's data segment is not considered for the insertion of any new rows until the amount of used space in the block falls to 39% or less (assuming that the block's used space has previously reached `PCTFREE`). [Figure 14-2](#) illustrates this.

Figure 14–2 PCTUSED



The default value for PCTUSED is 40 percent. After the free space in a data block reaches PCTFREE, no new rows are inserted in that block until the percentage of space used falls below PCTUSED. The percent value is for the block space available for data after overhead is subtracted from total space.

You can specify any integer between 0 and 99 (inclusive) for PCTUSED, as long as the sum of PCTUSED and PCTFREE does not exceed 100.

Effects of Specifying a Smaller PCTUSED

A smaller PCTUSED has the following effects:

- Reduces processing costs incurred during UPDATE and DELETE statements for moving a block to the free list when it has fallen below that percentage of usage
- Increases the unused space in a database

Effects of Specifying a Larger PCTUSED

A larger PCTUSED has the following effects:

- Improves space efficiency

- Increases processing cost during `INSERT` and `UPDATE`

Selecting Associated `PCTUSED` and `PCTFREE` Values

If you decide not to use the default values for `PCTFREE` or `PCTUSED`, keep the following guidelines in mind:

- The sum of `PCTFREE` and `PCTUSED` must be equal to or less than 100.
- If the sum equals 100, then Oracle attempts to keep no more than `PCTFREE` free space, and processing costs are highest.
- The smaller the difference between 100 and the sum of `PCTFREE` and `PCTUSED` (as in `PCTUSED` of 75, `PCTFREE` of 20), the more efficient space usage is, at some performance cost.

The following table contains examples that show how and why specific values for `PCTFREE` and `PCTUSED` are specified for tables.

Example	Scenario	Settings	Explanation
1	Common activity includes <code>UPDATE</code> statements that increase the size of the rows.	<code>PCTFREE=20</code> <code>PCTUSED=40</code>	<code>PCTFREE</code> is set to 20 to allow enough room for rows that increase in size as a result of updates. <code>PCTUSED</code> is set to 40 so that less processing is done during high update activity, thus improving performance.
2	Most activity includes <code>INSERT</code> and <code>DELETE</code> statements, and <code>UPDATE</code> statements that do not increase the size of affected rows.	<code>PCTFREE=5</code> <code>PCTUSED=60</code>	<code>PCTFREE</code> is set to 5 because most <code>UPDATE</code> statements do not increase row sizes. <code>PCTUSED</code> is set to 60 so that space freed by <code>DELETE</code> statements is used soon, yet processing is minimized.
3	The table is very large and storage is a primary concern. Most activity includes read-only transactions.	<code>PCTFREE=5</code> <code>PCTUSED=40</code>	<code>PCTFREE</code> is set to 5 because this is a large table and you want to completely fill each block.

Specifying the Transaction Entry Parameters: INITRANS and MAXTRANS

`INITRANS` specifies the number of DML transaction entries for which space is initially reserved in the data block header. Space is reserved in the headers of all data blocks in the associated segment.

As multiple transactions concurrently access the rows of the same data block, space is allocated for each DML transaction's entry in the block. Once the space reserved by `INITRANS` is depleted, space for additional transaction entries is allocated out of the free space in a block, if available. Once allocated, this space effectively becomes a permanent part of the block header. The `MAXTRANS` parameter limits the number of transaction entries that can concurrently use data in a data block. Therefore, you can limit the amount of free space that can be allocated for transaction entries in a data block using `MAXTRANS`.

The `INITRANS` and `MAXTRANS` parameters for the data blocks allocated to a specific schema object should be set individually for each schema object based on the following criteria:

- The space you would like to reserve for transaction entries compared to the space you would reserve for database data
- The number of concurrent transactions that are likely to touch the same data blocks at any given time

For example, if a table is very large and only a small number of users simultaneously access the table, the chances of multiple concurrent transactions requiring access to the same data block is low. Therefore, `INITRANS` can be set low, especially if space is at a premium in the database.

Alternatively, assume that a table is usually accessed by many users at the same time. In this case, you might consider preallocating transaction entry space by using a high `INITRANS`. This eliminates the overhead of having to allocate transaction entry space, as required when the object is in use. Also, allow a higher `MAXTRANS` so that no user has to wait to access necessary data blocks.

Setting Storage Parameters

This section describes the storage parameters that you can set for various data structures. These storage parameters apply to the following types of structures and schema objects:

- Tablespace (used as storage parameter defaults for all segments)

- Tables, partitions, clusters, materialized views, and materialized view logs (data segments)
- Indexes (index segments)
- Rollback segments

The following topics are discussed:

- [Identifying the Storage Parameters](#)
- [Setting Default Storage Parameters for Segments in a Tablespace](#)
- [Setting Storage Parameters for Data Segments](#)
- [Setting Storage Parameters for Index Segments](#)
- [Setting Storage Parameters for LOBs, Varrays, and Nested Tables](#)
- [Changing Values for Storage Parameters](#)
- [Understanding Precedence in Storage Parameters](#)
- [Example of How Storage Parameters Effect Space Allocation](#)

Identifying the Storage Parameters

Storage parameters determine space allocation for objects when they are created in a dictionary-managed tablespace. Locally managed tablespaces provide a simpler means of space allocation, and most storage parameters have no meaning in their context.

When you create a dictionary-managed tablespace you can specify default storage parameters. These values override the system defaults to become the defaults for objects created in that tablespace only. You specify the default storage values in the `DEFAULT STORAGE` clause of a `CREATE` or `ALTER TABLESPACE` statement.

Furthermore, for objects created in dictionary-managed tablespaces, you can specify storage parameters for each individual schema object. These parameter settings override any default storage settings. Use the `STORAGE` clause of the `CREATE` or `ALTER` statement for specifying storage parameters for the individual object. The following example illustrates specifying storage parameters when a table is being created:

```
CREATE TABLE players
  (code NUMBER(10) PRIMARY KEY,
   lastname VARCHAR(20),
   firstname VARCHAR(15),
```

```

        position VARCHAR2(20),
        team VARCHAR2(20))
PCTFREE 10
PCTUSED 40
STORAGE
  (INITIAL 25K
   NEXT 10K
   MAXEXTENTS 10
   MINEXTENTS 3);

```

Not all storage parameters can be specified for every type of database object, and not all storage parameters can be specified in both the `CREATE` and `ALTER` statements.

The following table contains a brief description of each storage parameter. For a complete description of these parameters, including their default, minimum, and maximum settings, see the *Oracle9i SQL Reference*.

Parameter	Description
INITIAL	The size, in bytes, of the first extent allocated when a segment is created. This parameter cannot be specified in an <code>ALTER</code> statement.
NEXT	The size, in bytes, of the next incremental extent to be allocated for a segment. The second extent is equal to the original setting for <code>NEXT</code> . From there forward, <code>NEXT</code> is set to the previous size of <code>NEXT</code> multiplied by $(1 + \text{PCTINCREASE}/100)$.
PCTINCREASE	The percentage by which each incremental extent grows over the last incremental extent allocated for a segment. If <code>PCTINCREASE</code> is 0, then all incremental extents are the same size. If <code>PCTINCREASE</code> is greater than zero, then each time <code>NEXT</code> is calculated, it grows by <code>PCTINCREASE</code> . <code>PCTINCREASE</code> cannot be negative. The new <code>NEXT</code> equals $1 + \text{PCTINCREASE}/100$, multiplied by the size of the last incremental extent (the old <code>NEXT</code>) and rounded up to the next multiple of a block size.
MINEXTENTS	The total number of extents to be allocated when the segment is created. This allows for a large allocation of space at creation time, even if contiguous space is not available.
MAXEXTENTS	The total number of extents, including the first, that can ever be allocated for the segment.

Parameter	Description
FREELIST GROUPS	<p>The number of groups of free lists for the database object you are creating. Oracle uses the instance number of Oracle Real Application Cluster instances to map each instance to one free list group. For information on the use of this parameter, see <i>Oracle9i Real Application Clusters Administration</i>.</p> <p>Note: This parameter is ignored for objects created in locally managed tablespaces with segment space management specified as <code>AUTO</code>.</p>
FREELISTS	<p>Specifies the number of free lists for each of the free list groups for the schema object. Not valid for tablespaces. The use of this parameter is discussed in <i>Oracle9i Database Performance Tuning Guide and Reference</i>.</p> <p>Note: This parameter is ignored for objects created in locally managed tablespaces with segment space management specified as <code>AUTO</code>.</p>
OPTIMAL	<p>Relevant only to rollback segments. See Chapter 13, "Managing Undo Space" for information on the use of this parameter.</p>
BUFFER POOL	<p>Defines a default buffer pool (cache) for a schema object. Not valid for tablespaces or rollback segments. For information on the use of this parameter, see <i>Oracle9i Database Performance Tuning Guide and Reference</i>.</p>

Setting Default Storage Parameters for Segments in a Tablespace

You can set default storage parameters for each tablespace of a database. Any storage parameter that you do not explicitly set when creating or subsequently altering a segment in a tablespace automatically is set to the corresponding default storage parameter for the tablespace in which the segment resides.

When specifying `MINEXTENTS` at the tablespace level, any extent allocated in the tablespace is rounded to a multiple of the number of minimum extents.

Setting Storage Parameters for Data Segments

You set the storage parameters for the data segment of a nonclustered table, materialized view, or materialized view log using the `STORAGE` clause of the `CREATE` or `ALTER` statement for tables, materialized views, or materialized view logs.

In contrast, you set the storage parameters for the data segments of a cluster using the `STORAGE` clause of the `CREATE CLUSTER` or `ALTER CLUSTER` statement,

rather than the individual `CREATE` or `ALTER` statements that put tables and materialized views into the cluster. Storage parameters specified when creating or altering a *clustered* table or materialized view are ignored. The storage parameters set for the cluster override the table's storage parameters.

With partitioned tables, you can set default storage parameters at the table level. When creating a new partition of the table, the default storage parameters are inherited from the table level (unless you specify them for the individual partition). If no storage parameters are specified at the table level, then they are inherited from the tablespace.

Setting Storage Parameters for Index Segments

Storage parameters for an index segment created for a table index can be set using the `STORAGE` clause of the `CREATE INDEX` or `ALTER INDEX` statement.

Storage parameters of an index segment created for the index used to enforce a primary key or unique key constraint can be set in either of the following ways:

- In the `ENABLE . . . USING INDEX` clause of the `CREATE TABLE` or `ALTER TABLE` statement
- In the `STORAGE` clause of the `ALTER INDEX` statement

Setting Storage Parameters for LOBs, Varrays, and Nested Tables

A table or materialized view can contain `LOB`, `varray`, or `nested table` column types. These entities can be stored in their own segments. `LOBs` and `varrays` are stored in `LOB` segments, while a `nested table` is stored in a storage table. You can specify a `STORAGE` clause for these segments that will override storage parameters specified at the table level.

See Also:

- *Oracle9i Application Developer's Guide - Large Objects (LOBs)*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i SQL Reference*

All of the above books contain more information about creating tables containing `LOBs`, `varrays`, and `nested tables`.

Changing Values for Storage Parameters

You can alter default storage parameters for tablespaces and specific storage parameters for individual segments if you so choose. Default storage parameters can be reset for a tablespace. However, changes affect only new objects created in the tablespace, or new extents allocated for a segment.

The `INITIAL` and `MINEXTENTS` storage parameters cannot be altered for an existing table, cluster, index, or rollback segment. If only `NEXT` is altered for a segment, the next incremental extent is the size of the new `NEXT`, and subsequent extents can grow by `PCTINCREASE` as usual.

If both `NEXT` and `PCTINCREASE` are altered for a segment, the next extent is the new value of `NEXT`, and from that point forward, `NEXT` is calculated using `PCTINCREASE` as usual.

Understanding Precedence in Storage Parameters

The storage parameters in effect at a given time are determined by the following types of SQL statements, listed in order of precedence (where higher numbers take precedence over lower numbers):

1. `ALTER [TABLE | CLUSTER | MATERIALIZED VIEW | MATERIALIZED VIEW LOG | INDEX | ROLLBACK] SEGMENT statement`
2. `CREATE [TABLE | CLUSTER | MATERIALIZED VIEW | MATERIALIZED VIEW LOG | INDEX | ROLLBACK] SEGMENT statement`
3. `ALTER TABLESPACE statement`
4. `CREATE TABLESPACE statement`
5. Oracle default values

Any storage parameter specified at the object level overrides the corresponding option set at the tablespace level. When storage parameters are not explicitly set at the object level, they default to those at the tablespace level. When storage parameters are not set at the tablespace level, Oracle system defaults apply. If storage parameters are altered, the new options apply only to the extents not yet allocated.

Note: The storage parameters for temporary segments always use the default storage parameters set for the associated tablespace.

Example of How Storage Parameters Effect Space Allocation

Assume the following statement has been executed:

```
CREATE TABLE test_storage
( . . . )
STORAGE (INITIAL 100K NEXT 100K
MINEXTENTS 2 MAXEXTENTS 5
PCTINCREASE 50);
```

Also assume that the initialization parameter `DB_BLOCK_SIZE` is set to 2K. The following table shows how extents are allocated for the `TEST_STORAGE` table. Also shown is the value for the incremental extent, as can be seen in the `NEXT` column of the `USER_SEGMENTS` or `DBA_SEGMENTS` data dictionary views:

Table 14–1 Extent Allocations

Extent#	Extent Size	Value for NEXT
1	50 blocks or 102400 bytes	50 blocks or 102400 bytes
2	50 blocks or 102400 bytes	75 blocks or 153600 bytes
3	75 blocks or 153600 bytes	113 blocks or 231424 bytes
4	115 blocks or 235520 bytes	170 blocks or 348160 bytes
5	170 blocks or 348160 bytes	No next value, MAXEXTENTS=5

If you change the `NEXT` or `PCTINCREASE` storage parameters with an `ALTER` statement (such as `ALTER TABLE`), the specified value replaces the current value stored in the data dictionary. For example, the following statement modifies the `NEXT` storage parameter of the `test_storage` table before the third extent is allocated for the table:

```
ALTER TABLE test_storage STORAGE (NEXT 500K);
```

As a result, the third extent is 500K when allocated, the fourth is $(500K * 1.5) = 750K$, and so forth.

Managing Resumable Space Allocation

Oracle provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. This enables you to take corrective action instead of the Oracle database server returning an error to the user. After the error condition is corrected, the suspended operation automatically

resumes. This feature is called **resumable space allocation**. The statements that are affected are called resumable statements.

This section contains the following topics:

- [Resumable Space Allocation Overview](#)
- [Enabling and Disabling Resumable Space Allocation](#)
- [Detecting Suspended Statements](#)
- [Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger](#)

Resumable Space Allocation Overview

This section provides an overview of resumable space allocation. It describes how resumable statements work, and specifically defines qualifying statements and error conditions.

How Resumable Statements Work

The following is an overview of how resumable statements work. Details are contained in later sections.

1. A statement executes in a resumable mode only when the client explicitly enables resumable semantics for the session using the `ALTER SESSION` statement.
2. A resumable statement is suspended when one of the following conditions occur (these conditions result in corresponding errors being signalled for nonresumable statements):
 - Out of space condition
 - Maximum extents reached condition
 - Space quota exceeded condition.
3. On suspending a resumable statement's execution, there are mechanisms to perform user supplied operations, log errors, and to query the status of the statement execution. When a resumable statement is suspended the following actions are taken:
 - The error is reported in the alert log.
 - If the user registered a trigger on the `AFTER SUSPEND` system event, the user trigger is executed. A user supplied PL/SQL procedure can access the

error message data using the `DBMS_RESUMABLE` package and `DBA/USER_RESUMABLE` view.

4. Suspending a statement automatically results in suspending the transaction. Thus all transactional resources are held through a statement suspend and resume.
5. When the error condition disappears (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution.
6. A suspended statement can be forced to throw the exception using the `DBMS_RESUMABLE.ABORT()` procedure. This procedure can be called by a DBA, or by the user who issued the statement.
7. A suspension time out interval is associated with resumable statements. A resumable statement that is suspended for the timeout interval (the default is two hours) wakes up and returns the exception to the user.
8. A resumable statement can be suspended and resumed multiple times during execution.

What Operations are Resumable?

Note: Resumable space allocation is fully supported when using locally managed tablespaces. There are certain limitations when using dictionary-managed tablespaces. See ["Resumable Space Allocation Limitations for Dictionary-Managed Tablespaces"](#) on page 14-18 for details.

The following operations are resumable:

- **Queries**
`SELECT` statements that run out of temporary space (for sort areas) are candidates for resumable execution. When using OCI, the calls `LNOCISmtExecute()` and `LNOCISmtFetch()` are candidates.
- **DML**
`INSERT`, `UPDATE`, and `DELETE` statements are candidates. The interface used to execute them does not matter; it can be OCI, JSQL, PL/SQL, or another interface. Also, `INSERT INTO . . . SELECT` from external tables can be resumable.

- **Import/Export**

As for SQL*Loader, a command line parameter controls whether statements are resumable after recoverable errors.

- **DDL**

The following statements are candidates for resumable execution:

- CREATE TABLE ... AS SELECT
- CREATE INDEX
- ALTER INDEX ... REBUILD
- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER INDEX ... REBUILD PARTITION
- ALTER INDEX ... SPLIT PARTITION
- CREATE MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW LOG

What Errors are Correctable?

There are three classes of correctable errors:

- **Out of space condition**

The operation cannot acquire any more extents for a table/index/temporary segment/rollback segment/undo segment/cluster/LOB/table partition/index partition in a tablespace. For example, the following errors fall in this category:

```
ORA-1650 unable to extend rollback segment ... in tablespace ...  
ORA-1653 unable to extend table ... in tablespace ...  
ORA-1654 unable to extend index ... in tablespace ...
```

- **Maximum extents reached condition**

The number of extents in a table/index/temporary segment/rollback segment/undo segment/cluster/LOB/table partition/index partition equals the maximum extents defined on the object. For example, the following errors fall in this category:

```
ORA-1628 max # extents ... reached for rollback segment ...  
ORA-1631 max # extents ... reached in table ...  
ORA-1654 max # extents ... reached in index ...
```

- Space quota exceeded condition

The user has exceeded his assigned space quota in the tablespace. Specifically, this is noted by the following error:

```
ORA-1536 space quote exceeded for tablespace string
```

Resumable Space Allocation Limitations for Dictionary-Managed Tablespaces

There are certain limitations of resumable space allocation when using dictionary-managed tablespaces. These limitations are listed below:

1. If a DDL operation such as `CREATE TABLE` or `CREATE INDEX` is executed with an explicit `MAXEXTENTS` setting which causes an out of space error during its execution, the operation will not be suspended. Instead, it will be aborted. This error is treated as not repairable because the properties of an object (for example, `MAXEXTENTS`) cannot be altered before its creation. However if a DML operation causes an already existing table or index to reach the `MAXEXTENTS` limit, it will be suspended and can be resumed later. This restriction can be overcome either by setting the `MAXEXTENTS` clause to `UNLIMITED` or by using locally managed tablespaces.
2. If rollback segments are located in dictionary managed tablespaces, then space allocation for rollback segments is not resumable. However, space allocation for user objects (tables, indexes, and the likes) would still be resumable. To workaround the limitation, we recommend using automatic undo management or placing the rollback segments in locally managed tablespaces.

Resumable Statements and Distributed Operations

Remote operations are not supported in resumable mode.

Parallel Execution and Resumable Statements

In parallel execution, if one of the parallel execution server processes encounters a correctable error, that server process suspends its execution. Other parallel execution server processes will continue executing their respective tasks, until either they encounter an error or are blocked (directly or indirectly) by the suspended server process. When the correctable error is resolved, the suspended process resumes execution and the parallel operation continues execution. If the suspended operation is terminated, the parallel operation aborts, throwing the error to the user.

Different parallel execution server processes may encounter one or more correctable errors. This may result in firing an `AFTER SUSPEND` trigger multiple times, in

parallel. Also, if a parallel execution server process encounters a noncorrectable error while another parallel execution server process is suspended, the suspended statement is immediately aborted.

For parallel execution, every parallel execution coordinator and server process has its own entry in `DBA/USER_RESUMABLE` view.

Enabling and Disabling Resumable Space Allocation

Resumable space allocation is only possible when statements are executed within a session that has resumable mode enabled.

To enable resumable mode for a session, use the following SQL statement:

```
ALTER SESSION ENABLE RESUMABLE;
```

Because suspended statements can hold up some system resources, users must be granted the `RESUMABLE` system privilege before they are allowed to enable and execute resumable statements.

To disable resumable mode, issue the following statement:

```
ALTER SESSION DISABLE RESUMABLE;
```

The default for a new session is resumable mode disabled.

You can also specify a timeout interval, and you can provide a name used to identify a resumable statement. These are discussed separately in following sections.

See Also: ["Setting Default Resumable Mode"](#) on page 14-20

Specifying a Timeout Interval

When you enable resumable mode for a session, you can also specify a timeout interval, after which a suspended statement will error if no intervention has taken place. The following statement specifies that resumable transactions will time out and error after 3600 seconds:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
```

The value of `TIMEOUT` remains in effect until it is changed by another `ALTER SESSION ENABLE RESUMABLE` statement, it is changed by another means, or the session ends. The default timeout interval is 7200 seconds.

See Also: ["Changing the Timeout Interval"](#) on page 14-20 for other methods of changing the timeout interval for resumable statements

Naming Resumable Statements

Resumable statements can be identified by name. The following statement assigns a name to resumable statements:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'insert into table';
```

The `NAME` value remains in effect until it is changed by another `ALTER SESSION ENABLE RESUMABLE` statement, or the session ends. The default value for `NAME` is:

```
User USERNAME(USERID), Session SESSIONID, Instance INSTANCEID
```

The name of the statement is used to identify the resumable statement in the `DBA_RESUMABLE` and `USER_RESUMABLE` views.

Setting Default Resumable Mode

To set default resumable mode, a DBA can register a database level `LOGON` trigger to alter a user's session to enable resumable and set a timeout interval.

Note: If there are multiple triggers registered that change default mode and timeout for resumable statements, the result will be unspecified because Oracle does not guarantee the order of trigger invocation.

Changing the Timeout Interval

In addition to the `ALTER SESSION ENABLE RESUMABLE` statement, there are other methods for setting or changing the timeout interval.

The `DBMS_RESUMABLE` package contains procedures for setting the timeout period for a specific session or for the current session. A DBA can change the default system timeout by creating a system wide `AFTER SUSPEND` trigger that calls `DBMS_RESUMABLE` to set it. For example, the following code sample sets a system wide default timeout to one hour:

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
    DBMS_RESUMABLE.SET_TIMEOUT(3600);
```

END;

Detecting Suspended Statements

When a resumable statement is suspended, the error is not raised to the client. In order for corrective action to be taken, Oracle provides alternative methods for notifying users of the error and for providing information about the circumstances.

AFTER SUSPEND System Event and Trigger

When a resumable statement encounter a correctable error, the system internally generates the `AFTER SUSPEND` system event. Users can register triggers for this event at both the database and schema level. If a user registers a trigger to handle this system event, the trigger is executed after a SQL statement has been suspended.

SQL statements executed within a `AFTER SUSPEND` trigger are always nonresumable and are always autonomous. Transactions started within the trigger use the `SYSTEM` rollback segment. These conditions are imposed to overcome deadlocks and reduce the chance of the trigger experiencing the same error condition as the statement.

Users can use the `USER_RESUMABLE` or `DBA_RESUMABLE` views, or the `DBMS_RESUMABLE.SPACE_ERROR_INFO` function, within triggers to get information about the resumable statements.

Triggers can also call the `DBMS_RESUMABLE` package to abort suspended statements and modify resumable timeout values.

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for information about system events, triggers, and attribute functions

Views Containing Information About Resumable Statements

The following views can be queried to obtain information about the status of resumable statements:

View	Description
<code>DBA_RESUMABLE</code> <code>USER_RESUMABLE</code>	These views contain rows for all currently executing or suspended resumable statements. They can be used by a DBA, <code>AFTER SUSPEND</code> trigger, or another session to monitor the progress of, or obtain specific information about, resumable statements.

View	Description
V\$SESSION_WAIT	When a statement is suspended the session invoking the statement is put into a wait state. A row is inserted into this view for the session with the EVENT column containing "statement suspended, wait error to be cleared".

See Also: *Oracle9i Database Reference* for specific information about the columns contained in these views

DBMS_RESUMABLE Package

The DBMS_RESUMABLE package helps control resumable statements. The following procedures are available:

Procedure	Description
ABORT(sessionID)	<p>This procedure aborts a suspended resumable statement. The parameter sessionID is the session ID in which the statement is executing. For parallel DML/DDI, sessionID is any session ID which participates in the parallel DML/DDI.</p> <p>Oracle guarantees that the ABORT operation always succeeds. It may be called either inside or outside of the AFTER SUSPEND trigger.</p> <p>The caller of ABORT must be the owner of the session with sessionID, have ALTER SYSTEM privilege, or have DBA privileges.</p>
GET_SESSION_TIMEOUT(sessionID)	This function returns the current timeout value of resumable statements for the session with sessionID. This returned timeout is in seconds. If the session does not exist, this function returns -1.
SET_SESSION_TIMEOUT(sessionID, timeout)	This procedure sets the timeout interval of resumable statements for the session with sessionID. The parameter timeout is in seconds. The new timeout setting will apply to the session immediately. If the session does not exist, no action is taken.
GET_TIMEOUT()	This function returns the current timeout value of resumable statements for the current session. The returned value is in seconds.
SET_TIMEOUT(timeout)	This procedure sets a timeout value for resumable statements for the current session. The parameter timeout is in seconds. The new timeout setting applies to the session immediately.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference*

Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger

In the following example, a system wide AFTER SUSPEND trigger is created and registered as user SYS at the database level. Whenever a resumable statement is suspended in any session, this trigger can have either of two effects:

- If the rollback segment has reached its space limit, then a message is sent to the DBA and the statement is aborted.
- If any other recoverable error has occurred, the timeout interval is reset to 8 hours.

Here are the statements for this example:

```
CREATE OR REPLACE TRIGGER resumable_default
AFTER SUSPEND
ON DATABASE
DECLARE
  /* declare transaction in this trigger is autonomous */
  /* this is not required because transactions within a trigger
     are always autonomous */
  PRAGMA AUTONOMOUS_TRANSACTION;
  cur_sid          NUMBER;
  cur_inst         NUMBER;
  errno           NUMBER;
  err_type        VARCHAR2;
  object_owner    VARCHAR2;
  object_type     VARCHAR2;
  table_space_name VARCHAR2;
  object_name     VARCHAR2;
  sub_object_name VARCHAR2;
  error_txt       VARCHAR2;
  msg_body       VARCHAR2;
  ret_value       BOOLEAN;
  mail_conn      UTL_SMTP.CONNECTION;
BEGIN
  -- Get session ID
  SELECT DISTINCT(SID) INTO cur_SID FROM V$MYSTAT;

  -- Get instance number
  cur_inst := userenv('instance');
```

```

-- Get space error information
ret_value :=
DBMS_RESUMABLE.SPACE_ERROR_INFO(err_type,object_type,object_owner,
    table_space_name,object_name, sub_object_name);
/*
-- If the error is related to rollback segments, log error, send email
-- to DBA, and abort the statement. Otherwise, set timeout to 8 hours.
--
-- sys.rbs_error is created by DBA manually and defined as
-- sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
-- suspend_time DATE)
*/

IF OBJECT_TYPE = 'ROLLBACK SEGMENT' THEN
    /* LOG ERROR */
    INSERT INTO sys.rbs_error (
        SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME
        FROM DBMS_RESUMABLE
        WHERE SESSION_ID = cur_sid AND INSTANCE_ID = cur_inst
    );
    SELECT ERROR_MSG INTO error_txt FROM DBMS_RESUMABLE
        WHERE SESSION_ID = cur_sid and INSTANCE_ID = cur_inst;

    -- Send email to recipient via UTL_SMTP package
    msg_body:='Subject: Space Error Occurred

        Space limit reached for rollback segment ' || object_name ||
        on ' || TO_CHAR(SYSDATE, 'Month dd, YYYY, HH:MIam') ||
        '. Error message was ' || error_txt;

    mail_conn := UTL_SMTP.OPEN_CONNECTION('localhost', 25);
    UTL_SMTP.HELO(mail_conn, 'localhost');
    UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
    UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
    UTL_SMTP.DATA(mail_conn, msg_body);
    UTL_SMTP.QUIT(mail_conn);

    -- Abort the statement
    DBMS_RESUMABLE.ABORT(cur_sid);
ELSE
    -- Set timeout to 8 hours
    DBMS_RESUMABLE.SET_TIMEOUT(28800);
END IF;

/* commit autonomous transaction */

```

```
COMMIT;  
END;
```

Deallocating Space

It is not uncommon to allocate space to a segment, only to find out later that it is not being used. For example, you can set `PCTINCREASE` to a high value, which could create a large extent that is only partially used. Or, you could explicitly overallocate space by issuing the `ALTER TABLE ... ALLOCATE EXTENT` statement. If you find that you have unused or overallocated space, you can release it so that the unused space can be used by other segments.

This section describes aspects of deallocating unused space.

Viewing the High Water Mark

Prior to deallocation, you can use the `DBMS_SPACE` package, which contains a procedure (`UNUSED_SPACE`) that returns information about the position of the high water mark and the amount of unused space in a segment.

Within a segment, the high water mark indicates the amount of used space, or space that had been formatted to receive data. You cannot release space below the high water mark (even if there is no data in the space you want to deallocate). However, if the segment is completely empty, you can release space using the `TRUNCATE ... DROP STORAGE` statement.

For segments in locally managed tablespaces with segment space management specified as `AUTO`, the following output parameters still determine the high water mark, but their meaning is somewhat altered:

- `LAST_USED_EXTENT_FILE_ID`
- `LAST_USED_EXTENT_BLOCK_ID`
- `LAST_USED_BLOCK`

Specifically, it is possible for some blocks below the high water mark to be unformatted. Neither the `UNUSED_SPACE` nor the `FREE_SPACE` procedure of `DBMS_SPACE` accurately accounts for unused space when segment space management is specified as `AUTO`. Use the `SPACE_USAGE` procedure instead.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* contains the description of the `DBMS_SPACE` package

Issuing Space Deallocation Statements

The following statements deallocate unused space in a segment (table, index or cluster). The `KEEP` clause is *optional*.

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;  
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;  
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

When you explicitly identify an amount of unused space to `KEEP`, this space is retained while the remaining unused space is deallocated. If the remaining number of extents becomes smaller than `MINEXTENTS`, the `MINEXTENTS` value changes to reflect the new number. If the initial extent becomes smaller, the `INITIAL` value changes to reflect the new size of the initial extent.

If you do not specify the `KEEP` clause, all unused space (everything above the high water mark) is deallocated, as long as the size of the initial extent and `MINEXTENTS` are preserved. Thus, even if the high water mark occurs within the `MINEXTENTS` boundary, `MINEXTENTS` remains and the initial extent size is not reduced.

You can verify the deallocated space is freed by examining the `DBA_FREE_SPACE` view.

See Also:

- *Oracle9i SQL Reference* for details on the syntax and options associated with deallocating unused space
- *Oracle9i Database Reference* for more information about the `DBA_FREE_SPACE` view

Examples of Deallocating Space

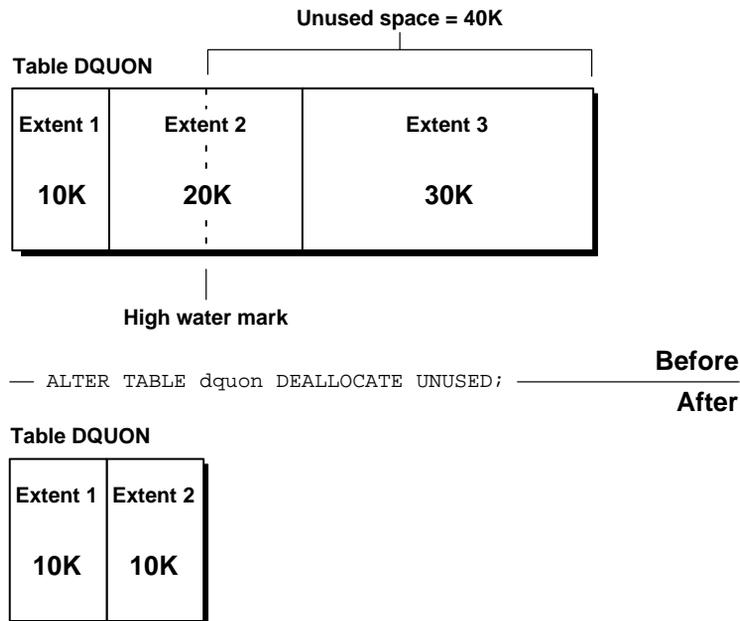
This section provides some space deallocation examples.

Deallocating Space Example 1:

A table consists of three extents. The first extent is 10K, the second is 20K, and the third is 30K. The high water mark is in the middle of the second extent, and there is 40K of unused space. [Figure 14-3](#) illustrates the effect of issuing the following statement:

```
ALTER TABLE dquon DEALLOCATE UNUSED;
```

All unused space is deallocated, leaving table `dquon` with two remaining extents. The third extent disappears, and the second extent size is 10K.

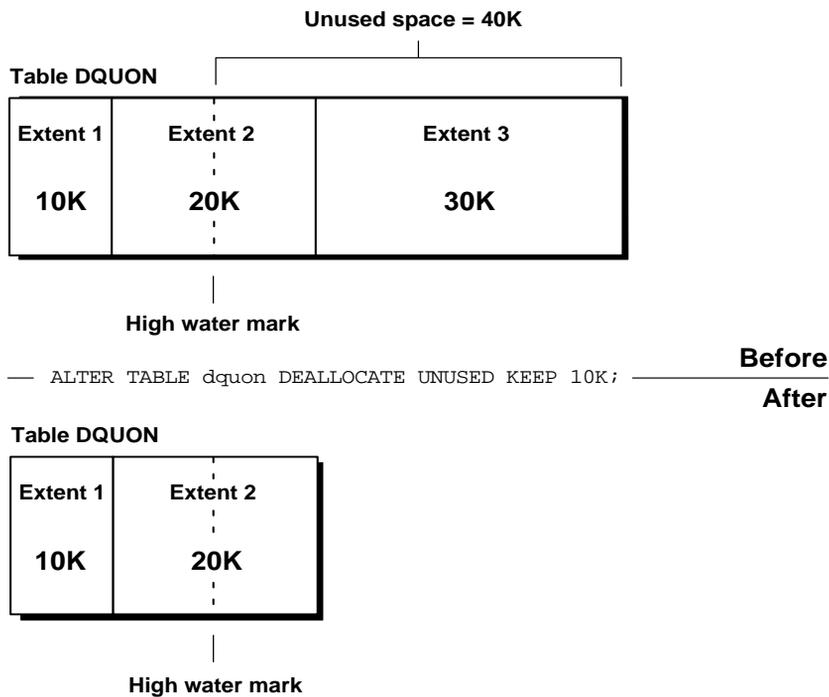
Figure 14–3 Deallocating All Unused Space

But, if you had issued the following statement specifying the `KEEP` keyword, then 10K above the high water mark would be kept, and the rest of the unused space would be deallocated from `dquon`.

```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 10K;
```

In effect, the third extent is deallocated and the second extent remains intact.

[Figure 14–4](#) illustrates this situation.

Figure 14–4 Deallocating Unused Space, KEEP 10K

Further, if you deallocate all unused space from `dquon` and keep 20K, as specified in the following statement, the third extent is cut to 10K, and the size of the second extent remains the same.

```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 20K;
```

Deallocating Space Example 2:

Consider the situation illustrated by [Figure 14–3](#). Extent 3 is completely deallocated, and the second extent is left with 10K. Further, the size of the next allocated extent defaults to the size of the last completely deallocated extent, which in this case, is 30K. If this is not what you want, you can explicitly set the size of the next extent using the `ALTER TABLE` statement, specifying a new value for `NEXT` in the storage clause.

The following statement sets the next extent size for table `dquon` to 20K:

```
ALTER TABLE dquon STORAGE (NEXT 20K);
```

Deallocating Space Example 3:

To preserve the `MINEXTENTS` number of extents, `DEALLOCATE` can retain extents that were originally allocated to a segment. This capacity is influenced by the `KEEP` parameter and was explained earlier.

If table `dquon` has a `MINEXTENTS` value of 2, the statements illustrated in [Figure 14-3](#) and [Figure 14-4](#) still yield the same results as shown, and further, the initial value of `MINEXTENTS` is preserved.

However, if the `MINEXTENTS` value is 3, then the statement illustrated in [Figure 14-4](#) produces the same result as shown (the third extent is removed), but the value of `MINEXTENTS` is changed to 2. However, the statement illustrated in [Figure 14-3](#) does not produce the same result. In this case, the statement has no effect.

Understanding Space Use of Datatypes

When creating tables and other data structures, you need to know how much space they will require. Each datatype has different space requirements. The *PL/SQL User's Guide and Reference* and *Oracle9i SQL Reference* contain extensive descriptions of datatypes and their space requirements.

Managing Tables

This chapter describes the various aspects of managing tables, and includes the following topics:

- [Guidelines for Managing Tables](#)
- [Creating Tables](#)
- [Altering Tables](#)
- [Redefining Tables Online](#)
- [Dropping Tables](#)
- [Managing Index-Organized Tables](#)
- [Managing External Tables](#)
- [Viewing Information About Tables](#)

See Also:

- [Chapter 14, "Managing Space for Schema Objects"](#) is recommended reading before attempting tasks in this chapter.
- [Chapter 21, "General Management of Schema Objects"](#) presents additional aspects of managing tables, such as specifying integrity constraints and analyzing tables.
- [Chapter 17, "Managing Partitioned Tables and Indexes"](#) discusses partitioned tables.

Guidelines for Managing Tables

This section describes guidelines to follow when managing tables. Following these guidelines can make the management of your tables easier, and improve performance both when creating the table and later querying or updating it.

The following topics are discussed:

- [Design Tables Before Creating Them](#)
- [Specify How Data Block Space Is to Be Used](#)
- [Specify the Location of Each Table](#)
- [Consider Parallelizing Table Creation](#)
- [Consider Using NOLOGGING When Creating Tables](#)
- [Estimate Table Size and Set Storage Parameters](#)
- [Plan for Large Tables](#)
- [Table Restrictions](#)

Design Tables Before Creating Them

Usually, the application developer is responsible for designing the elements of an application, including the tables. Database administrators are responsible for setting storage parameters and defining clusters for tables, based on information from the application developer about how the application works and the types of data expected.

Working with your application developer, carefully plan each table so that the following occurs:

- Tables are normalized.
- Each column is of the proper datatype.
- Columns that allow nulls are defined last, to conserve storage space.
- Tables are clustered whenever appropriate, to conserve storage space and optimize performance of SQL statements. Clustered tables are the subject of [Chapter 18, "Managing Clusters"](#).

Specify How Data Block Space Is to Be Used

By specifying the `PCTFREE` and `PCTUSED` parameters during the creation of each table, you can affect the efficiency of space utilization and amount of space reserved

for updates to the current data in the data blocks of a table's data segment. The `PCTFREE` and `PCTUSED` parameters are discussed in ["Managing Space in Data Blocks"](#) on page 14-2.

Note: When you create a table in a locally managed tablespace for which automatic segment-space management is enabled, the need to specify the `PCTFREE` (or `FREELISTS`) parameter is eliminated. Automatic segment-space management is specified at the tablespace level. The Oracle database server automatically and efficiently manages free and used space within objects created in such tablespaces.

Locally managed tablespaces and automatic segment space management are discussed in ["Locally Managed Tablespaces"](#) on page 15-2.

Specify the Location of Each Table

If you have the proper privileges and tablespace quota, you can create a new table in any tablespace that is currently online. It is advisable to specify the `TABLESPACE` clause in a `CREATE TABLE` statement to identify the tablespace that is to store the new table. If you do not specify a tablespace in a `CREATE TABLE` statement, the table is created in your default tablespace.

When specifying the tablespace to contain a new table, make sure that you understand implications of your selection. By properly specifying a tablespace during the creation of each table, you can:

- Increase the performance of the database system
- Decrease the time needed for database administration

The following situations illustrate how specifying incorrect storage locations for schema objects can affect a database:

- If users' objects are created in the `SYSTEM` tablespace, the performance of Oracle can suffer, since both data dictionary objects and user objects must contend for the same datafiles.
- If an application's associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for that application's data can be increased.

[Chapter 24, "Managing Users and Resources"](#) contains information about assigning default tablespaces and tablespace quotas to users.

Consider Parallelizing Table Creation

You can utilize parallel execution when creating tables using a subquery (`AS SELECT`) in the `CREATE TABLE` statement. Because multiple processes work together to create the table, performance of the table creation operation is improved.

Parallelizing table creation is discussed in the section ["Parallelizing Table Creation"](#) on page 15-8.

Consider Using NOLOGGING When Creating Tables

To create a table most efficiently use the `NOLOGGING` clause in the `CREATE TABLE . . . AS SELECT` statement. The `NOLOGGING` clause causes minimal redo information to be generated during the table creation. This has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the table is decreased.
- Performance improves for parallel creation of large tables.

The `NOLOGGING` clause also specifies that subsequent direct loads using `SQL*Loader` and direct load `INSERT` operations are not logged. Subsequent DML statements (`UPDATE`, `DELETE`, and conventional path insert) are unaffected by the `NOLOGGING` attribute of the table and generate redo.

If you cannot afford to lose the table after you have created it (for example, you will no longer have access to the data used to create the table) you should take a backup immediately after the table is created. In some situations, such as for tables that are created for temporary use, this precaution may not be necessary.

In general, the relative performance improvement of specifying `NOLOGGING` is greater for larger tables than for smaller tables. For small tables, `NOLOGGING` has little effect on the time it takes to create a table. However, for larger tables the performance improvement can be significant, especially when you are also parallelizing the table creation.

Estimate Table Size and Set Storage Parameters

Estimating the sizes of tables before creating them is useful for the following reasons:

- You can use the combined estimated size of tables, along with estimates for indexes, undo space, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.
- You can use the estimated size of an individual table to better manage the disk space that the table will use. When a table is created, you can set appropriate storage parameters and improve I/O performance of applications that use the table. For example, assume that you estimate the maximum size of a table before creating it. If you then set the storage parameters when you create the table, fewer extents are allocated for the table's data segment, and all of the table's data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this table.

Whether or not you estimate table size before creation, you can explicitly set storage parameters when creating each table. (Clustered tables, discussed in [Chapter 18, "Managing Clusters"](#), automatically use the storage parameters of the cluster.) Any storage parameter that you do not explicitly set when creating or subsequently altering a table automatically uses the corresponding default storage parameter set for the tablespace in which the table resides. Storage parameters are discussed in ["Setting Storage Parameters"](#) on page 14-8.

If you explicitly set the storage parameters for the extents of a table's data segment, try to store the table's data in a small number of large extents rather than a large number of small extents.

Plan for Large Tables

There are no limits on the physical size of tables and extents. You can specify the keyword `UNLIMITED` for `MAXEXTENTS`, thereby simplifying your planning for large objects, reducing wasted space and fragmentation, and improving space reuse. However, when the number of extents in a table grows very large, you can see an impact on performance when performing any operation requiring that table.

Note: You cannot alter data dictionary tables to have `MAXEXTENTS` greater than the allowed block maximum.

If you have large tables in your database, consider the following recommendations:

- Separate the table from its indexes.

Place indexes in separate tablespaces from other objects, and on separate disks if possible. If you ever must drop and re-create an index on a very large table (such as when disabling and enabling a constraint, or re-creating the table), indexes isolated into separate tablespaces can often find contiguous space more easily than those in tablespaces that contain other objects.

- Allocate sufficient temporary space.

If applications that access the data in a very large table perform large sorts, ensure that enough space is available for large temporary segments (temporary segments always use the default `STORAGE` settings for their tablespaces).

Table Restrictions

Here are some restrictions to be aware of before you create tables:

- Tables containing object types cannot be imported into a pre-Oracle8 database.
- You cannot move types and extent tables to a different schema when the original data still exists in the database.
- You cannot merge an exported table into a preexisting table having the same name in a different schema.
- Oracle has a limit on the total number of columns that a table (or attributes that an object type) can have. See *Oracle9i Database Reference* for this limit.

Further, when you create a table that contains user-defined type data, Oracle maps columns of user-defined type to relational columns for storing the user-defined type data. This causes additional relational columns to be created. This results in "hidden" relational columns that are not visible in a `DESCRIBE` table statement and are not returned by a `SELECT *` statement. Therefore, when you create an object table, or a relational table with columns of `REF`, `varray`, nested table, or object type, be aware that the total number of columns that Oracle actually creates for the table can be more than those you specify.

See Also: *Oracle9i Application Developer's Guide - Object-Relational Features* for more information about user-defined types

Creating Tables

To create a new table in your schema, you must have the `CREATE TABLE` system privilege. To create a table in another user's schema, you must have the `CREATE ANY TABLE` system privilege. Additionally, the owner of the table must have a

quota for the tablespace that contains the table, or the `UNLIMITED TABLESPACE` system privilege.

Create tables using the SQL statement `CREATE TABLE`.

This section contains the following topics:

- [Creating a Table](#)
- [Creating a Temporary Table](#)
- [Parallelizing Table Creation](#)
- [Automatically Collecting Statistics on Tables](#)

See Also: *Oracle9i SQL Reference* for exact syntax of the `CREATE TABLE` and other SQL statements discussed in this chapter

Creating a Table

When you issue the following statement, you create a table named `admin_emp` in the `hr` schema and store it in the `admin_tbs` tablespace:

```
CREATE TABLE      hr.admin_emp (
    empno          NUMBER(5) PRIMARY KEY,
    ename          VARCHAR2(15) NOT NULL,
    job            VARCHAR2(10),
    mgr            NUMBER(5),
    hiredate       DATE DEFAULT (sysdate),
    sal            NUMBER(7,2),
    comm           NUMBER(7,2),
    deptno         NUMBER(3) NOT NULL
                  CONSTRAINT admin_dept_fkey REFERENCES hr.departments
                  (department_id))
TABLESPACE admin_tbs
STORAGE ( INITIAL 50K
          NEXT 50K
          MAXEXTENTS 10
          PCTINCREASE 25 );
```

In this example, integrity constraints are defined on several columns of the table. Integrity constraints are discussed in "[Managing Integrity Constraints](#)" on page 21-14. Several segment attributes are also explicitly specified for the table. These are explained in [Chapter 14, "Managing Space for Schema Objects"](#).

Creating a Temporary Table

It is also possible to create a temporary table. The definition of a temporary table is visible to all sessions, but the data in a temporary table is visible only to the session that inserts the data into the table. You use the `CREATE GLOBAL TEMPORARY TABLE` statement to create a temporary table. The `ON COMMIT` keywords indicate if the data in the table is **transaction-specific** (the default) or **session-specific**:

- `ON COMMIT DELETE ROWS` specifies that the temporary table is transaction specific and Oracle truncates the table (delete all rows) after each commit.
- `ON COMMIT PRESERVE ROWS` specifies that the temporary table is session specific and Oracle truncates the table when you terminate the session.

This example creates a temporary table that is transaction specific:

```
CREATE GLOBAL TEMPORARY TABLE admin_work_area
    (startdate DATE,
     enddate DATE,
     class CHAR(20))
    ON COMMIT DELETE ROWS;
```

Indexes can be created on temporary tables. They are also temporary and the data in the index has the same session or transaction scope as the data in the underlying table.

See Also:

- *Oracle9i Database Concepts* for more information about temporary tables
- *Oracle9i Application Developer's Guide - Fundamentals* for more examples of temporary table use

Parallelizing Table Creation

When you specify the `AS SELECT` clause when creating a table, you can utilize parallel execution. The `CREATE TABLE ... AS SELECT` statement contains two parts: a `CREATE` part (DDL) and a `SELECT` part (query). Oracle can parallelize both parts of the statement. The `CREATE` part is parallelized if *one* of the following is true:

- A `PARALLEL` clause is included in the `CREATE TABLE ... AS SELECT` statement
- An `ALTER SESSION FORCE PARALLEL DDL` statement is specified

The query part is parallelized if *all* of the following are true:

- The query includes a parallel hint specification (`PARALLEL` or `PARALLEL_INDEX`) or the `CREATE` part includes the `PARALLEL` clause or the schema objects referred to in the query have a `PARALLEL` declaration associated with them.
- At least one of the tables specified in the query requires either a full table scan or an index range scan spanning multiple partitions.

If you parallelize the creation of a table, that table then has a parallel declaration (the `PARALLEL` clause) associated with it. Any subsequent DML or queries on the table, for which parallelization is possible, will attempt to use parallel execution.

The following simple example parallelizes the creation of a table:

```
CREATE TABLE hr.admin_emp_dept
  PARALLEL
  AS SELECT * FROM hr.employees
  WHERE department_id = 10;
```

In this example the `PARALLEL` clause tells Oracle to select an optimum number of parallel execution servers when creating the table.

See Also:

- *Oracle9i Database Concepts* for more information about parallel execution
- *Oracle9i Data Warehousing Guide* for a more detailed discussion about using parallel execution
- ["Managing Processes for Parallel Execution"](#) on page 5-18

Automatically Collecting Statistics on Tables

The PL/SQL package `DBMS_STATS` lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics. You can also use this package to identify or name statistics that have been gathered.

You enable `DBMS_STATS` to automatically gather statistics for a table by specifying the `MONITORING` clause in the `CREATE` (or `ALTER`) `TABLE` statement. Then, you can effect automated statistics gathering by, for example, setting up a recurring job (perhaps by using job queues) that invokes `DBMS_STATS.GATHER_TABLE_STATS` with the `GATHER STALE` option at an appropriate interval for your application.

Monitoring tracks the approximate number of `INSERT`, `UPDATE`, and `DELETE` operations for the table since the last time statistics were gathered. Information

about how many rows are affected is maintained in the SGA, until periodically (about every three hours) SMON incorporates the data into the data dictionary. This data dictionary information is made visible through the `DBA_TAB_MODIFICATIONS`, `ALL_TAB_MODIFICATIONS`, or `USER_TAB_MODIFICATIONS` views. Oracle uses these views to identify tables with stale statistics.

Using the `MONITORING` clause and the `DBMS_STATS` package enables the optimizer to generate accurate execution plans.

To disable monitoring of a table, specify the `NOMONITORING` clause.

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for a discussion of the exact mechanism for using the `MONITORING` clause and the `DBMS_STATS` package for gathering statistics

Altering Tables

You alter a table using the `ALTER TABLE` statement. To alter a table, the table must be contained in your schema, or you must have either the `ALTER` object privilege for the table or the `ALTER ANY TABLE` system privilege.

The following are many of the reasons for altering a table:

- To modify physical characteristics (`PCTFREE`, `PCTUSED`, `INITRANS`, `MAXTRANS`, or storage parameters)
- To move the table to a new segment or tablespace
- To explicitly allocate an extent or deallocate unused space
- To add, drop, or rename columns, or modify an existing column's definition (datatype, length, default value, and `NOT NULL` integrity constraint)
- To modify the logging attributes of the table
- To modify the `CACHE/NOCACHE` attributes
- To add, modify or drop integrity constraints associated with the table
- To enable or disable integrity constraints or triggers associated with the table
- To modify the degree of parallelism for the table
- To enable or disable statistics collection (`MONITORING/NOMONITORING`)
- To rename a table
- To add or modify index-organized table characteristics

- To alter the characteristics of an external table
- To add or modify LOB columns
- To add or modify object type, nested table, or varray columns

Some of the usages of the `ALTER TABLE` statement are presented in the following sections:

- [Altering Physical Attributes of a Table](#)
- [Moving a Table to a New Segment or Tablespace](#)
- [Manually Allocating Storage for a Table](#)
- [Modifying an Existing Column's Definition](#)
- [Adding Table Columns](#)
- [Renaming Table Columns](#)
- [Dropping Table Columns](#)

Caution: Before altering a table, familiarize yourself with the consequences of doing so. The *Oracle9i SQL Reference* lists many of these consequences in the descriptions of the `ALTER TABLE` clauses.

If a view, materialized view, trigger, domain index, function-based index, check constraint, function, procedure of package depends on a base table, the alteration of the base table or its columns can affect the dependent object. See "[Managing Object Dependencies](#)" on page 21-23 for information about how Oracle manages dependencies.

Altering Physical Attributes of a Table

When altering the data block space usage parameters (`PCTFREE` and `PCTUSED`) of a table, note that new settings apply to all data blocks used by the table, including blocks already allocated and subsequently allocated for the table. However, the blocks already allocated for the table are not immediately reorganized when space usage parameters are altered, but as necessary after the change. The data block storage parameters are described in "[Managing Space in Data Blocks](#)" on page 14-2.

When altering the transaction entry settings (`INITRANS`, `MAXTRANS`) of a table, note that a new setting for `INITRANS` applies only to data blocks subsequently allocated

for the table, while a new setting for `MAXTRANS` applies to all blocks (already and subsequently allocated blocks) of a table. To better understand these transaction entry setting parameters, see ["Specifying the Transaction Entry Parameters: `INITRANS` and `MAXTRANS`"](#) on page 14-8.

The storage parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters (for example, `NEXT`, `PCTINCREASE`) affect only extents subsequently allocated for the table. The size of the next extent allocated is determined by the current values of `NEXT` and `PCTINCREASE`, and is not based on previous values of these parameters. Storage parameters are discussed in ["Setting Storage Parameters"](#) on page 14-8.

Moving a Table to a New Segment or Tablespace

The `ALTER TABLE ... MOVE` statement enables you to relocate data of a nonpartitioned table into a new segment, and optionally into a different tablespace for which you have quota. This statement also allows you to modify any of the table's storage attributes, including those which cannot be modified using `ALTER TABLE`.

The following statement moves the `hr.admin_emp` table to a new segment, specifying new storage parameters:

```
ALTER TABLE hr.admin_emp MOVE
  STORAGE ( INITIAL 20K
           NEXT 40K
           MINEXTENTS 2
           MAXEXTENTS 20
           PCTINCREASE 0 );
```

If the table includes `LOB` column(s), this statement can be used to move the table along with `LOB` data and `LOB` index segments (associated with this table) which the user explicitly specifies. If not specified, the default is to not move the `LOB` data and `LOB` index segments.

Manually Allocating Storage for a Table

Oracle dynamically allocates additional extents for the data segment of a table, as required. However, perhaps you want to allocate an additional extent for a table explicitly. For example, in an Oracle Real Application Clusters environment, an extent of a table can be allocated explicitly for a specific instance.

A new extent can be allocated for a table using the `ALTER TABLE ... ALLOCATE EXTENT` clause.

You can also explicitly deallocate unused space using the `DEALLOCATE UNUSED` clause of `ALTER TABLE`. This is described in "[Deallocating Space](#)" on page 14-25.

See Also: *Oracle9i Real Application Clusters Administration* for information about using the `ALLOCATE EXTENT` clause in an Oracle Real Application Clusters environment

Modifying an Existing Column's Definition

Use the `ALTER TABLE ... MODIFY` statement to modify an existing column's definition. You can modify a column's datatype, default value, or column constraint.

You can increase the length of an existing column, or decrease it, if all existing data satisfies the new length. You can change a column from byte semantics to `CHAR` semantics or vice versa. You must set the initialization parameter `BLANK_TRIMMING=TRUE` to decrease the length of a nonempty `CHAR` column.

If you are modifying a table to increase the length of a column of datatype `CHAR`, realize that this can be a time consuming operation and can require substantial additional storage, especially if the table contains many rows. This is because the `CHAR` value in each row must be blank-padded to satisfy the new column length.

See Also: *Oracle9i SQL Reference* for additional information about modifying table columns and additional restrictions

Adding Table Columns

To add a column to an existing table, use the `ALTER TABLE ... ADD` statement.

The following statement alters the `hr.admin_emp` table to add a new column named `bonus`:

```
ALTER TABLE hr.admin_emp
  ADD (bonus NUMBER (7,2));
```

If a new column is added to a table, the column is initially `NULL` unless you specify the `DEFAULT` clause. When you specify a default value, Oracle updates each row in the new column with the values specified.

You can add a column with a `NOT NULL` constraint to a table only if the table does not contain any rows, or you specify a default value.

See Also: *Oracle9i SQL Reference* for additional information about adding table columns and additional restrictions

Renaming Table Columns

Oracle allows you to rename existing columns in a table. Use the `RENAME COLUMN` clause of the `ALTER TABLE` statement to rename a column. The new name must not conflict with the name of any existing column in the table. No other clauses are allowed in conjunction with the `RENAME COLUMN` clause.

The following statement renames the `comm` column of the `hr.admin_emp` table.

```
ALTER TABLE hr.admin_emp
    RENAME COLUMN comm TO commission;
```

As noted earlier, altering a table's column can invalidate dependent objects. However, when you rename a column, Oracle updates associated data dictionary tables to ensure that function-based indexes and check constraints remain valid.

Oracle also allows you to rename column constraints. This is discussed in ["Renaming Constraints"](#) on page 21-19.

Note: The `RENAME TO` clause of `ALTER TABLE` appears similar in syntax to the `RENAME COLUMN` clause, but is used for renaming the table itself.

Dropping Table Columns

You can drop columns that are no longer needed from a table, including an index-organized table. This provides a convenient means to free space in a database, and avoids your having to export/import data then re-create indexes and constraints.

You cannot drop all columns from a table, nor can you drop columns from a table owned by `SYS`. Any attempt to do so results in an error.

See Also: *Oracle9i SQL Reference* for information about additional restrictions and options for dropping columns from a table

Removing Columns from Tables

When you issue an `ALTER TABLE ... DROP COLUMN` statement, the column descriptor and the data associated with the target column are removed from each row in the table. You can drop multiple columns with one statement. The following statements are examples of dropping columns from the `hr.admin_emp` table.

This statement drops only the `sal` column:

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

The following statement drops both the `bonus` and `comm` columns:

```
ALTER TABLE hr.admin_emp DROP (bonus, commission);
```

Marking Columns Unused

If you are concerned about the length of time it could take to drop column data from all of the rows in a large table, you can use the `ALTER TABLE ... SET UNUSED` statement. This statement marks one or more columns as unused, but does not actually remove the target column data or restore the disk space occupied by these columns. However, a column that is marked as unused is not displayed in queries or data dictionary views, and its name is removed so that a new column can reuse that name. All constraints, indexes, and statistics defined on the column are also removed.

To mark the `hiredate` and `mgr` columns as unused, execute the following statement:

```
ALTER TABLE hr.admin_emp SET UNUSED (hiredate, mgr);
```

You can later remove columns that are marked as unused by issuing an `ALTER TABLE ... DROP UNUSED COLUMNS` statement. Unused columns are also removed from the target table whenever an explicit drop of any particular column or columns of the table is issued.

The data dictionary views `USER_UNUSED_COL_TABS`, `ALL_UNUSED_COL_TABS`, or `DBA_UNUSED_COL_TABS` can be used to list all tables containing unused columns. The `COUNT` field shows the number of unused columns in the table.

```
SELECT * FROM DBA_UNUSED_COL_TABS;
```

OWNER	TABLE_NAME	COUNT
HR	ADMIN_EMP	2

Removing Unused Columns

The `ALTER TABLE ... DROP UNUSED COLUMNS` statement is the only action allowed on unused columns. It physically removes unused columns from the table and reclaims disk space.

In the example that follows the optional keyword `CHECKPOINT` is specified. This option causes a checkpoint to be applied after processing the specified number of rows, in this case 250. Checkpointing cuts down on the amount of undo logs

accumulated during the drop column operation to avoid a potential exhaustion of undo space.

```
ALTER TABLE hr.admin_emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

Redefining Tables Online

In highly available systems, it is occasionally necessary to redefine large "hot" tables to improve the performance of queries or DML performed against these tables. Oracle provide a mechanism to redefine tables online. This mechanism provides a significant increase in availability compared to traditional methods of redefining tables that require tables to be taken offline.

When a table is redefined online, it is accessible to DML during much of the redefinition process. The table is locked in the exclusive mode only during a very small window which is independent of the size of the table and the complexity of the redefinition.

This section contains the following topics:

- [Features of Online Table Redefinition](#)
- [The DBMS_REDEFINITION Package](#)
- [Steps for Online Redefinition of Tables](#)
- [Intermediate Synchronization](#)
- [Abort and Cleanup After Errors](#)
- [Example of Online Table Redefinition](#)
- [Restrictions](#)

Features of Online Table Redefinition

Online table redefinition enables you to:

- Modify the storage parameters of the table
- Move the table to a different tablespace in the same schema
- Add support for parallel queries
- Add or drop partitioning support
- Re-create the table to reduce fragmentation

- Change the organization of a normal table (heap organized) to an index-organized table and vice versa
- Add or drop a column

The DBMS_REDEFINITION Package

The mechanism for performing online redefinition is the PL/SQL package `DBMS_REDEFINITION`. Execute privileges on this package is granted to `EXECUTE_CATALOG_ROLE`. In addition to having execute privileges on this package, you must be granted the following privileges:

- `CREATE ANY TABLE`
- `ALTER ANY TABLE`
- `DROP ANY TABLE`
- `LOCK ANY TABLE`
- `SELECT ANY TABLE`

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference*

Steps for Online Redefinition of Tables

In order to perform an online redefinition of a table the user must perform the following steps.

1. Choose one of the following two methods of redefinition:
 - The first method of redefinition, and the preferred method, is to use the primary keys to perform the redefinition. For this method, the pre-redefinition and the post-redefinition versions of the tables should have the same primary key columns. This is the default method of redefinition.
 - The second method of redefinition is to use `ROWIDS`. For this method, the table to be redefined should not be an index organized table. Also, in this method of redefinition, a hidden column (`M_ROW$$`) is added to the post-redefined version of the table and it is recommended that this column be marked as unused or dropped after the redefinition is completed.
2. Verify that the table can be online redefined by invoking the `DBMS_REDEFINITION.CAN_REDEF_TABLE()` procedure and specifying the method of redefinition to be used. If the table is not a candidate for online redefinition,

then this procedure raises an error indicating why the table cannot be online redefined.

3. Create an empty interim table (in the same schema as the table to be redefined) with all of the desired attributes. If columns are to be dropped, do not include them in the definition of the interim table. If a column is to be added, then add the column definition to the interim table.

It is possible to perform table redefinition in parallel. If you specify a degree of parallelism on both of the tables and you ensure that parallel execution is enabled for the session, Oracle will use parallel execution whenever possible to perform the redefinition.

4. Start the redefinition process by calling `DBMS_REDEFINITION.START_REDEF_TABLE()`, providing the following:
 - The table to be redefined
 - The interim table name
 - The column mapping
 - The method of redefinition

If the column mapping information is not supplied, then it is assumed that all the columns (with their names unchanged) are to be included in the interim table. If the column mapping is supplied, then only those columns specified explicitly in the column mapping are considered. If the method of redefinition is not specified, then the default method of redefinition using primary keys is assumed.

5. Create any triggers, indexes, grants and constraints on the interim table. Any referential constraints involving the interim table (that is, the interim table is either a parent or a child table of the referential constraint) must be created disabled. Until the redefinition process is either completed or aborted, any trigger defined on the interim table will not execute.

When the redefinition is completed, the triggers, constraints, indexes and grants associated with the interim table replace those on the table being redefined. The referential constraints involving the interim table (created disabled) transfer to the table being redefined and become enabled after the redefinition is complete.

6. Execute the `DBMS_REDEFINITION.FINISH_REDEF_TABLE()` procedure to complete the redefinition of the table. During this procedure, the original table is locked in the exclusive mode for a very small window. This window is independent of the amount of data in the original table. Also, as part of this procedure, the following occurs:

- a. The original table is redefined such that it has all the attributes, indexes, constraints, grants and triggers of the interim table
 - b. The referential constraints involving the interim table now involve the post redefined table and are enabled.
7. Optionally, rename any indexes, triggers, and constraints that were created on the interim table and that are now defined on the redefined table. If the redefinition was done using ROWIDS, the post-redefined table will have a hidden column (M_ROW\$\$) and it is recommended that the user set this hidden column to unused as follows:

```
ALTER TABLE table_name SET UNUSED (M_ROW$$)
```

8. The following is the end result of the redefinition process:
- The original table is redefined with the attributes and features of the interim table.
 - The triggers, grants, indexes and constraints defined on the interim table after `START_REDEF_TABLE()` and before `FINISH_REDEF_TABLE()` are now defined on the post-redefined table. Any referential constraints involving the interim table before the redefinition process was finished now involve the post-redefinition table and are enabled.
 - Any indexes, triggers, grants and constraints defined on the original table (prior to redefinition) are transferred to the interim table and are dropped when the user drops the interim table. Any referential constraints involving the original table before the redefinition now involve the interim table and are disabled.
 - Any PL/SQL procedures and cursors defined on the original table (prior to redefinition) are invalidated. They are automatically revalidated (this revalidation can fail if the shape of the table was changed as a result of the redefinition process) whenever they are used next.

Intermediate Synchronization

After the redefinition process has been started by calling `START_REDEF_TABLE()` and before `FINISH_REDEF_TABLE()` has been called, it is possible that a large number of DML statements have been executed on the original table. If you know this is the case, it is recommended that you periodically synchronize the interim table with the original table. This is done by calling the `DBMS_REDEFINITION.SYNC_INTERIM_TABLE()` procedure. Calling this procedure

reduces the time taken by `FINISH_REDEF_TABLE()` to complete the redefinition process.

The small amount of time that the original table is locked during `FINISH_REDEF_TABLE()` is independent of whether `SYNC_INTERIM_TABLE()` has been called.

Abort and Cleanup After Errors

In the event that an error is raised during the redefinition process, or if you choose to abort the redefinition process, call `DBMS_REDEFINITION.ABORT_REDEF_TABLE()`. This procedure drops temporary logs and tables associated with the redefinition process. After this procedure is called, the user can drop the interim table and its associated objects.

Example of Online Table Redefinition

This example illustrates online redefinition of the previously created table `hr.admin_emp`, which at this point only contains columns: `empno`, `ename`, `job`, `deptno`. The table is redefined as follows:

- New columns `mgr`, `hiredate`, `sal`, and `bonus` (these existed in the original table but were dropped in previous examples) are added.
- The new column `bonus` is initialized to 0
- The column `deptno` has its value increased by 10.
- The redefined table is partitioned by range on `empno`.

The steps in this redefinition are illustrated below.

1. Verify that the table is a candidate for online redefinition.

```
BEGIN
DBMS_REDEFINITION.CAN_REDEF_TABLE('hr','admin_emp',
                                dbms_redefinition.cons_use_pk);
END;
/
```

2. Create an interim table `hr.int_admin_emp`.

```
CREATE TABLE hr.int_admin_emp
(empno      NUMBER(5) PRIMARY KEY,
 ename      VARCHAR2(15) NOT NULL,
 job        VARCHAR2(10),
 mgr        NUMBER(5),
 hiredate   DATE DEFAULT (sysdate),
```

```

        sal          NUMBER(7,2),
        deptno       NUMBER(3) NOT NULL,
        bonus        NUMBER (7,2) DEFAULT(1000)
PARTITION BY RANGE(empno)
(PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE admin_tbs,
PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE admin_tbs2);

```

3. Start the redefinition process.

```

BEGIN
DBMS_REDEFINITION.START_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp',
'empno empno, ename ename, job job, deptno+10 deptno, 0 bonus',
dbms_redefinition.cons_use_pk);
END;
/

```

4. Create any triggers, indexes and constraints on `hr.int_admin_emp`. During the final step of redefinition, these are transferred back to the original table. Any referential constraints involved on `hr.int_admin_emp` should be disabled. You can define any grants associated with the interim table. These replace the grants on the original table after the redefinition.

```

ALTER TABLE hr.int_admin_emp ADD CONSTRAINT admin_dept_fkey2
FOREIGN KEY (deptno) REFERENCES hr.departments (department_id);
ALTER TABLE hr.int_admin_emp MODIFY CONSTRAINT admin_dept_fkey2
DISABLE KEEP INDEX;

```

The disabled constraint, `admin_dept_fkey2`, will be enabled automatically as part of the finish redefinition process and will then involve the newly redefined `admin_emp` table.

5. Optionally, synchronize the interim table `hr.int_admin_emp`.

```

BEGIN
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/

```

6. Complete the redefinition.

```

BEGIN
DBMS_REDEFINITION.FINISH_REDEF_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/

```

The table `hr.admin_emp` is locked in the exclusive mode only for a small window toward the end of this step. After this call the table `hr.admin_emp` is redefined such that it has all the attributes of the `hr.int_admin_emp` table.

7. Drop the interim table.

Restrictions

The following restrictions apply to the online redefinition of tables:

- If the table is to be redefined using primary keys, then the table to be redefined and the post-redefinition table must have the same primary key columns. If the table is to be redefined using `ROWIDS`, then the table must not be an index-organized table.
- Tables that have materialized views and materialized view logs defined on them cannot be online redefined.
- Tables that are materialized view container tables and Advanced Queuing tables cannot be online redefined.
- The overflow table of an index-organized table cannot be online redefined.
- Tables with user-defined types (objects, `REFs`, collections, typed tables) cannot be online redefined.
- Tables with `BFILE` columns cannot be online redefined.
- Tables with `LONG` columns cannot be online redefined. Tables with `LOB` columns are acceptable.
- The table to be redefined cannot be part of a cluster.
- Tables in the `SYS` and `SYSTEM` schema cannot be online redefined.
- Temporary tables cannot be redefined.
- There is no horizontal subsetting support.
- Only simple deterministic expressions can be used when mapping the columns in the interim table to those of the original table. For example, subqueries are not allowed.
- If new columns (which are not instantiated with existing data for the original table) are being added as part of the redefinition, then they must not be declared `NOT NULL` until the redefinition is complete.
- There cannot be any referential constraints between the table being redefined and the interim table.

- Table redefinition cannot be done NOLOGGING.

Dropping Tables

To drop a table, the table must be contained in your schema or you must have the `DROP ANY TABLE` system privilege.

To drop a table that is no longer needed, use the `DROP TABLE` statement. The following statement drops the `hr.int_admin_emp` table:

```
DROP TABLE hr.int_admin_emp;
```

If the table to be dropped contains any primary or unique keys referenced by foreign keys of other tables and you intend to drop the `FOREIGN KEY` constraints of the child tables, include the `CASCADE` clause in the `DROP TABLE` statement, as shown below:

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

Caution: Before dropping a table, familiarize yourself with the consequences of doing so:

- Dropping a table removes the table definition from the data dictionary. All rows of the table are no longer accessible.
 - All indexes and triggers associated with a table are dropped.
 - All views and PL/SQL program units dependent on a dropped table remain, yet become invalid (not usable). See "[Managing Object Dependencies](#)" on page 21-23 for information about how Oracle manages dependencies.
 - All synonyms for a dropped table remain, but return an error when used.
 - All extents allocated for a table that is dropped are returned to the free space of the tablespace and can be used by any other object requiring new extents or new objects. All rows corresponding to a clustered table are deleted from the blocks of the cluster. Clustered tables are the subject of [Chapter 18](#), "[Managing Clusters](#)".
-
-

Perhaps instead of dropping a table, you want to truncate it. The `TRUNCATE` statement provides a fast, efficient method for deleting all rows from a table, but it

does not affect any structures associated with the table being truncated (column definitions, constraints, triggers, and so forth) or authorizations. The `TRUNCATE` statement is discussed in "[Truncating Tables and Clusters](#)" on page 21-9.

Managing Index-Organized Tables

This section describes aspects of managing index-organized tables, and includes the following topics:

- [What are Index-Organized Tables](#)
- [Creating Index-Organized Tables](#)
- [Maintaining Index-Organized Tables](#)
- [Analyzing Index-Organized Tables](#)
- [Using the ORDER BY Clause with Index-Organized Tables](#)
- [Converting Index-Organized Tables to Regular Tables](#)

What are Index-Organized Tables

An **index-organized table** has a storage organization that is a variant of a primary B-tree. Unlike an ordinary (heap-organized) table whose data is stored as an unordered collection (heap), data for an index-organized table is stored in a B-tree index structure in a primary key sorted manner. Besides storing the primary key column values of an index-organized table row, each index entry in the B-tree stores the nonkey column values as well.

Why use Index-Organized Tables

Index-organized tables provide fast key-based access to table data for queries involving exact match and range searches. Changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure (because there is no separate table storage area).

Also, storage requirements are reduced because key columns are not duplicated in the table and index. The remaining nonkey columns are stored in the index structure.

Index-organized tables are particularly useful when you are using applications that must retrieve data based on a primary key. Index-organized tables are also suitable for modeling application-specific index structures. For example, content-based

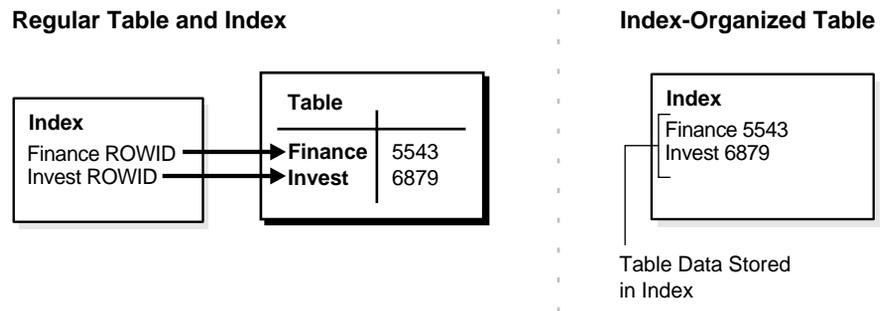
information retrieval applications containing text, image and audio data require inverted indexes that can be effectively modeled using index-organized tables.

Differences Between Index Organized and Regular Tables

As shown in [Figure 15-1](#), the index-organized table is somewhat similar to a configuration consisting of an ordinary table and an index on one or more of the table columns, but instead of maintaining two separate storage structures, one for the table and one for the B-tree index, the database system maintains only a single B-tree index. Also, rather than having a row's rowid stored in the index entry, the nonkey column values are stored. Thus, each B-tree index entry contains:

primary_key_value, non_primary_key_column_values

Figure 15-1 Structure of Regular Table versus an Index-Organized Table



Applications manipulate the index-organized table just like an ordinary table, using SQL statements. However, the database system performs all operations by manipulating the corresponding B-tree index.

See Also:

- *Oracle9i Database Concepts* for more details about index-organized tables
- *Oracle9i SQL Reference* for details of the syntax involved in creating index-organized tables

Creating Index-Organized Tables

You use the `CREATE TABLE` statement to create index-organized tables, but you must provide the following additional information:

- An `ORGANIZATION INDEX` qualifier, which indicates that this is an index-organized table
- A primary key, specified through a column constraint clause (for a single column primary key) or a table constraint clause (for a multiple-column primary key). A primary key must be specified for index-organized tables.
- An optional row overflow specification clause (`OVERFLOW`), which preserves dense clustering of the B-tree index by storing the row column values exceeding a specified threshold in a separate overflow data segment. An `INCLUDING` clause can also be specified to specify what (nonkey) columns are to be stored in the overflow data segment.
- A `PCTTHRESHOLD` value which defines the percentage of space reserved in the index block for an index-organized table. Any portion of the row that exceeds the specified threshold is stored in the overflow segment. In other words, the row is broken at a column boundary into two pieces, a head piece and tail piece. The head piece fits in the specified threshold and is stored along with the key in the index leaf block. The tail piece is stored in the overflow area as one or more row pieces. Thus, the index entry contains the key value, the nonkey column values that fit the specified threshold, and a pointer to the rest of the row.

The following example creates an index-organized table:

```
CREATE TABLE admin_docindex(
    token char(20),
    doc_id NUMBER,
    token_frequency NUMBER,
    token_offsets VARCHAR2(512),
    CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id)
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE admin_tbs2;
```

The above example shows that the `ORGANIZATION INDEX` qualifier specifies an index-organized table, where the key columns and nonkey columns reside in an index defined on columns that designate the primary key (`token`, `doc_id`) for the table.

Index-organized tables can store object types. The following example creates object type `admin_typ`, then creates an index-organized table containing a column of object type `admin_typ`:

```
CREATE OR REPLACE TYPE admin_typ AS OBJECT
    (col1 NUMBER, col2 VARCHAR2(6));
```

```
CREATE TABLE admin_iot (c1 NUMBER primary key, c2 admin_typ)
  ORGANIZATION INDEX;
```

You can also create an index-organized table of object types. For example:

```
CREATE TABLE admin_iot2 OF admin_typ (col1 PRIMARY KEY)
  ORGANIZATION INDEX;
```

See Also: ["Creating Partitioned Index-Organized Tables"](#) on page 17-19 for information about creating partitioned index-organized tables

Using the AS Subquery

You can create an index-organized table using the AS subquery. Creating an index-organized table in this manner enables you to load the table in parallel by using the PARALLEL option.

The following statement creates an index-organized table (in parallel) by selecting rows from the conventional table hr.jobs:

```
CREATE TABLE admin_iot3(i PRIMARY KEY, j, k, l)
  ORGANIZATION INDEX PARALLEL (DEGREE 2)
  AS SELECT * FROM hr.jobs;
```

Using the Overflow Clause

The overflow clause specified in the earlier example indicates that any nonkey columns of rows exceeding 20% of the block size are placed in a data segment stored in the admin_tbs2 tablespace. The key columns should fit the specified threshold.

If an update of a nonkey column causes the row to decrease in size, Oracle identifies the row piece (head or tail) to which the update is applicable and rewrites that piece.

If an update of a nonkey column causes the row to increase in size, Oracle identifies the piece (head or tail) to which the update is applicable and rewrites that row piece. If the update's target turns out to be the head piece, note that this piece can again be broken into 2 to keep the row size below the specified threshold.

The nonkey columns that fit in the index leaf block are stored as a row head-piece that contains a ROWID field linking it to the next row piece stored in the overflow data segment. The only columns that are stored in the overflow area are those that do not fit.

Choosing and Monitoring a Threshold Value You should choose a threshold value that can accommodate your key columns, as well as the first few nonkey columns (if they are frequently accessed).

After choosing a threshold value, you can monitor tables to verify that the value you specified is appropriate. You can use the `ANALYZE TABLE ... LIST CHAINED ROWS` statement to determine the number and identity of rows exceeding the threshold value.

See Also: *Oracle9i SQL Reference* for details about this use of the `ANALYZE` statement

Using the INCLUDING clause In addition to specifying `PCTTHRESHOLD`, you can use the `INCLUDING` clause to control which nonkey columns are stored with the key columns. Oracle accommodates all nonkey columns up to the column specified in the `INCLUDING` clause in the index leaf block, provided it does not exceed the specified threshold. All nonkey columns beyond the column specified in the `INCLUDING` clause are stored in the overflow area.

Note: Oracle moves all primary key columns of an indexed-organized table to the beginning of the table (in their key order), in order to provide efficient primary key based access. As an example:

```
CREATE TABLE admin_iot4(a INT, b INT, c INT, d INT,
                        primary key(c,b))
    ORGANIZATION INDEX;
```

The stored column order is: c b a d (instead of: a b c d). The last primary key column is b, based on the stored column order. The `INCLUDING` column can be the last primary key column (b in this example), or any nonkey column (that is, any column after b in the stored column order).

The example presented earlier can be modified to create an index-organized table where the `token_offsets` column value is always stored in the overflow area:

```
CREATE TABLE admin_docindex2(
    token CHAR(20),
    doc_id NUMBER,
    token_frequency NUMBER,
    token_offsets VARCHAR2(512),
    CONSTRAINT pk_admin_docindex2 PRIMARY KEY (token, doc_id))
```

```

ORGANIZATION INDEX
TABLESPACE admin_tbs
PCTTHRESHOLD 20
INCLUDING token_frequency
OVERFLOW TABLESPACE admin_tbs2;

```

Here, only nonkey columns prior to `token_offsets` (in this case a single column only) are stored with the key column values in the index leaf block.

Using Key Compression

Creating an index-organized table using key compression enables you to eliminate repeated occurrences of key column prefix values.

Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys in each index block while improving performance.

You can enable key compression using the `COMPRESS` clause while:

- creating an index-organized table
- moving an index-organized table

You can also specify the prefix length (as the number of key columns), which identifies how the key columns are broken into a prefix and suffix entry.

```

CREATE TABLE admin_iot5(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
  ORGANIZATION INDEX COMPRESS;

```

The preceding statement is equivalent to the following statement:

```

CREATE TABLE admin_iot6(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
  ORGANIZATION INDEX COMPRESS 2;

```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4) the repeated occurrences of (1,2), (1,3) are compressed away.

You can also override the default prefix length used for compression as follows:

```

CREATE TABLE admin_iot7(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
  ORGANIZATION INDEX COMPRESS 1;

```

For the list of values (1,2,3), (1,2,4), (1,2,7), (1,3,5), (1,3,4), (1,4,4), the repeated occurrences of 1 are compressed away.

You can disable compression as follows:

```
ALTER TABLE admin_iot5 MOVE NOCOMPRESS;
```

See Also: *Oracle9i Database Concepts* for more details about key compression

Maintaining Index-Organized Tables

Index-organized tables differ from regular tables only in physical organization; logically, they are manipulated in the same manner. You can use an index-organized table in place of a regular table in `INSERT`, `SELECT`, `DELETE`, and `UPDATE` statements.

Altering Index-Organized Tables

You can use the `ALTER TABLE` statement to modify physical and storage attributes for both primary key index and overflow data segments. All the attributes specified prior to the `OVERFLOW` keyword are applicable to the primary key index segment. All attributes specified after the `OVERFLOW` key word are applicable to the overflow data segment. For example, you can set the `INITTRANS` of the primary key index segment to 4 and the overflow of the data segment `INITTRANS` to 6 as follows:

```
ALTER TABLE admin_docindex INITTRANS 4 OVERFLOW INITTRANS 6;
```

You can also alter `PCTTHRESHOLD` and `INCLUDING` column values. A new setting is used to break the row into head and overflow tail pieces during subsequent operations. For example, the `PCTTHRESHOLD` and `INCLUDING` column values can be altered for the `admin_docindex` table as follows:

```
ALTER TABLE admin_docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

By setting the `INCLUDING` column to `doc_id`, all the columns that follow `token_frequency` and `token_offsets`, are stored in the overflow data segment.

For index-organized tables created without an overflow data segment, you can add an overflow data segment by using the `ADD OVERFLOW` clause. For example, you can add an overflow segment to table `admin_iot3` as follows:

```
ALTER TABLE admin_iot3 ADD OVERFLOW TABLESPACE admin_tbs2;
```

Moving (Rebuilding) Index-Organized Tables

Because index-organized tables are primarily stored in a B-tree index, you can encounter fragmentation as a consequence of incremental updates. However, you can use the `ALTER TABLE ... MOVE` statement to rebuild the index and reduce this fragmentation.

The following statement rebuilds the index-organized table `admin_docindex`:

```
ALTER TABLE admin_docindex MOVE;
```

You can rebuild index-organized tables online using the `ONLINE` keyword. The overflow data segment, if present, is rebuilt when the `OVERFLOW` keyword is specified. For example, to rebuild the `admin_docindex` table but not the overflow data segment, perform a move online as follows:

```
ALTER TABLE admin_docindex MOVE ONLINE;
```

To rebuild the `admin_docindex` table along with its overflow data segment perform the move online as shown in the following statement. This statement also illustrates moving both the table and overflow data segment to new tablespaces.

```
ALTER TABLE admin_docindex MOVE TABLESPACE admin_tbs2
OVERFLOW TABLESPACE admin_tbs3;
```

In this last example, an index organized table with a LOB column (CLOB) is built. Then the table is moved while the LOB index and data segment are rebuilt and moved to a new tablespace.

```
CREATE TABLE admin_iot_lob
  (c1 number (6) primary key,
  admin_lob CLOB)
  ORGANIZATION INDEX
  LOB (admin_lob) STORE AS (TABLESPACE admin_tbs2);
ALTER TABLE admin_iot_lob MOVE LOB (admin_lob) STORE AS (TABLESPACE admin_tbs3);
```

Updating the Key Column

A key column update is logically equivalent to deleting the row with the old key value and inserting the row with the new key value at the appropriate place to maintain the primary key order.

Logically, in the following example, the `admin_docindex` table row for `token='coins'` and `doc_id=10` is deleted and a new row for `token='medals'` and `doc_id=10` is inserted:

```
UPDATE admin_docindex
  SET token='medals'
  WHERE token='coins' and doc_id=10;
```

Analyzing Index-Organized Tables

Just like conventional tables, index-organized tables are analyzed using the `ANALYZE` statement. For example, the following statement gathers statistics for the `admin_docindex` table:

```
ANALYZE TABLE admin_docindex COMPUTE STATISTICS;
```

Note: Oracle recommends that the `DBMS_STATS` package be used for collecting optimizer statistics. See "[Analyzing Tables, Indexes, and Clusters](#)" on page 21-3 for information about collecting optimizer statistics, and for information about using the `ANALYZE` statement to collect non-optimizer statistics, validate object structure, and list chained rows.

The `ANALYZE` statement analyzes both the primary key index segment and the overflow data segment, and computes logical as well as physical statistics for the table.

- The logical statistics can be queried using `USER_TABLES`, `ALL_TABLES` or `DBA_TABLES`.
- You can query the physical statistics of the primary key index segment using `USER_INDEXES`, `ALL_INDEXES` or `DBA_INDEXES` (and using the primary key index name). For example, you can obtain the primary key index segment's physical statistics for the table `admin_docindex` as follows:

```
SELECT LAST_ANALYZED, BLEVEL, LEAF_BLOCKS, DISTINCT_KEYS
       FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_ADMIN_DOCINDEX';
```

- You can query the physical statistics for the overflow data segment using the `USER_TABLES`, `ALL_TABLES` or `DBA_TABLES`. You can identify the overflow entry by searching for `IOT_TYPE = 'IOT_OVERFLOW'`. For example, you can obtain overflow data segment physical attributes associated with the `admin_docindex` table as follows:

```
SELECT LAST_ANALYZED, NUM_ROWS, BLOCKS, EMPTY_BLOCKS
       FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
              and IOT_NAME= 'ADMIN_DOCINDEX';
```

Using the ORDER BY Clause with Index-Organized Tables

If an `ORDER BY` clause only references the primary key column or a prefix of it, then the optimizer avoids the sorting overhead as the rows are returned sorted on the primary key columns.

The following queries avoid sorting overhead because the data is already sorted on the primary key:

```
SELECT * FROM admin_docindex2 ORDER BY token, doc_id;
SELECT * FROM admin_docindex2 ORDER BY token;
```

If, however, you have an `ORDER BY` clause on a suffix of the primary key column or non-primary key columns, additional sorting is required (assuming no other secondary indexes are defined).

```
SELECT * FROM admin_docindex2 ORDER BY doc_id;
SELECT * FROM admin_docindex2 ORDER BY token_frequency
```

Converting Index-Organized Tables to Regular Tables

You can convert index-organized tables to regular tables using the Oracle `IMPORT` or `EXPORT` utilities, or the `CREATE TABLE ... AS SELECT` statement.

To convert an index-organized table to a regular table:

- Export the index-organized table data using conventional path.
- Create a regular table definition with the same definition.
- Import the index-organized table data, making sure `IGNORE=y` (ensures that object exists error is ignored).

Note: Before converting an index-organized table to a regular table, be aware that index-organized tables cannot be exported using pre-Oracle8 versions of the Export utility.

See Also: *Oracle9i Database Utilities* for more details about using the `IMPORT` and `EXPORT` utilities

Managing External Tables

Oracle allows you read-only access to data in external tables. External tables are defined as tables that do not reside in the database, and can be in any format for

which an access driver is provided. By providing Oracle with metadata describing an external table, Oracle is able to expose the data in the external table as if it were data residing in a regular database table. The external data can be queried directly and in parallel using SQL.

You can, for example, select, join, or sort external table data. You can also create views and synonyms for external tables. However, no DML operations (UPDATE, INSERT, or DELETE) are possible, and no indexes can be created, on external tables.

Note: The DBMS_STATS package can be used for gathering statistics for external tables. The ANALYZE statement is not supported for gathering statistics for external tables.

For information about using the DBMS_STATS package, see *Oracle9i Database Performance Tuning Guide and Reference*

The means of defining the metadata for external tables is through the CREATE TABLE ... ORGANIZATION EXTERNAL statement. This external table definition can be thought of as a view that allows running any SQL query against external data without requiring that the external data first be loaded into the database. An access driver is the actual mechanism used to read the external data in the table.

Oracle provides an access driver for external tables. It allows the reading of data from external files using the Oracle loader technology. The ORACLE_LOADER access driver provides data mapping capabilities which are a subset of the control file syntax of SQL*Loader utility.

Oracle's external tables feature provides a valuable means for performing basic extraction, transformation, and transportation (ETT) tasks that are common for datawarehousing.

These following sections discuss the DDL statements that are supported for external tables. Only DDL statements discussed are supported, and not all clauses of these statements are supported.

- [Creating External Tables](#)
- [Altering External Tables](#)
- [Dropping External Tables](#)
- [System and Object Privileges for External Tables](#)

See Also:

- *Oracle9i Database Utilities* contains more information about external tables and describes the access driver and its access parameters
- *Oracle9i Data Warehousing Guide* for information about using external tables in a datawarehousing environment

Creating External Tables

You create external tables using the `ORGANIZATION EXTERNAL` clause of the `CREATE TABLE` statement. You are not in fact creating a table; that is, an external table does not have any extents associated with it. Rather, you are creating metadata in the data dictionary that enables you to access external data.

The following example creates an external table, then uploads the data to a database table.

EXAMPLE: Creating an External Table and Loading Data

The file `empxt1.dat` contains the following sample data:

```
360,Jane,Janus,ST_CLERK,121,17-MAY-2001,3000,0,50,jjanus
361,Mark,Jasper,SA_REP,145,17-MAY-2001,8000,.1,80,mjasper
362,Brenda,Starr,AD_ASST,200,17-MAY-2001,5500,0,10,bstarr
363,Alex,Alda,AC_MGR,145,17-MAY-2001,9000,.15,80,aalda
```

The file `empxt2.dat` contains the following sample data:

```
401,Jesse,Cromwell,HR_REP,203,17-MAY-2001,7000,0,40,jcromwel
402,Abby,Applegate,IT_PROG,103,17-MAY-2001,9000,.2,60,aapplega
403,Carol,Cousins,AD_VP,100,17-MAY-2001,27000,.3,90,ccousins
404,John,Richardson,AC_ACCOUNT,205,17-MAY-2001,5000,0,110,jrichard
```

The following SQL statements create an external table in the `hr` schema named `admin_ext_employees` and load its data into the `hr.employees` table.

```
CONNECT / AS SYSDBA;
-- Set up directories and grant access to hr
CREATE OR REPLACE DIRECTORY admin_dat_dir
  AS '/net/dlsun301/private6/examples/submitted/ADMIN/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
  AS '/net/dlsun301/private6/examples/submitted/ADMIN/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
  AS '/net/dlsun301/private6/examples/submitted/ADMIN/flatfiles/bad';
```

```

GRANT READ ON DIRECTORY admin_dat_dir TO hr;
GRANT WRITE ON DIRECTORY admin_log_dir TO hr;
GRANT WRITE ON DIRECTORY admin_bad_dir TO hr;
-- hr connects
CONNECT hr/hr
-- create the external table
CREATE TABLE admin_ext_employees
    (employee_id      NUMBER(4),
     first_name       VARCHAR2(20),
     last_name        VARCHAR2(25),
     job_id           VARCHAR2(10),
     manager_id       NUMBER(4),
     hire_date        DATE,
     salary            NUMBER(8,2),
     commission_pct   NUMBER(2,2),
     department_id    NUMBER(4),
     email            VARCHAR2(25)
    )
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY admin_dat_dir
  ACCESS PARAMETERS
  (
    records delimited by newline
    badfile admin_bad_dir:'empxt%a_%p.bad'
    logfile admin_log_dir:'empxt%a_%p.log'
    fields terminated by ','
    missing field values are null
    ( employee_id, first_name, last_name, job_id, manager_id,
      hire_date char date_format date mask "dd-mon-yyyy",
      salary, commission_pct, department_id, email
    )
  )
  LOCATION ('empxt1.dat', 'empxt2.dat')
)
PARALLEL
REJECT LIMIT UNLIMITED;
-- enable parallel for loading (good if lots of data to load)
ALTER SESSION ENABLE PARALLEL DML;
-- load the data in hr employees table
INSERT INTO employees (employee_id, first_name, last_name, job_id, manager_id,
                      hire_date, salary, commission_pct, department_id, email)
  SELECT * FROM admin_ext_employees;

```

The following paragraphs contain descriptive information about this example.

The first few statements in this example create the directory objects for the operating system directories that contain the data sources, and for the bad record and log files specified in the access parameters. You must also grant `READ` or `WRITE` directory object privileges, as appropriate.

Note: When creating a directory object or `BFILEs`, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
 - The operating system directory path named in the Oracle `DIRECTORY` object must be an existing OS directory path.
 - The operating system directory path named in the Oracle `DIRECTORY` object should not contain any symbolic links in its components.
-
-

The `TYPE` specification is given only to illustrate its use. If not specified, `ORACLE_LOADER` is the default access driver. The access parameters, specified in the `ACCESS PARAMETERS` clause, are opaque to Oracle. These access parameters are defined by the access driver, and are provided to the access driver by Oracle when the external table is accessed. See *Oracle9i Database Utilities* for a description of the `ORACLE_LOADER` access parameters.

The `PARALLEL` clause enables parallel query on the data sources. The granule of parallelism is by default a data source, but parallel access within a data source is implemented whenever possible. For example, if `PARALLEL=3` were specified, then more than one parallel execution server could be working on a data source. But, parallel access within a data source is provided by the access driver only if all of the following conditions are met:

- The media allows random positioning within a data source
- It is possible to find a record boundary from a random position
- The data files are large enough to make it worthwhile to break up into multiple chunks

Note: Specifying a `PARALLEL` clause is of value *only* when dealing with large amounts of data. Otherwise, it is not advisable to specify a `PARALLEL` clause, and doing so can be detrimental.

The `REJECT LIMIT` clause specifies that there is no limit on the number of errors that can occur during a query of the external data. For parallel access, this limit applies to each parallel execution server independently. For example, if `REJECT LIMIT 10` is specified, each parallel query process is allowed 10 rejections. Hence, the only precisely enforced values for `REJECT LIMIT` on parallel query are 0 and `UNLIMITED`.

In this example, the `INSERT INTO TABLE` statement generates a dataflow from the external data source to the Oracle SQL engine where data is processed. As data is parsed by the access driver from the external table sources and provided to the external table interface, the external data is converted from its external representation to its Oracle internal data type.

See Also: *Oracle9i SQL Reference* provides details of the syntax of the `CREATE TABLE` statement for creating external tables and specifies restrictions on the use of clauses

Altering External Tables

You can use any of the following `ALTER TABLE` clauses to change the characteristics of an external table. No other clauses are permitted.

ALTER TABLE Clause	Description	Example
<code>REJECT LIMIT</code>	Changes the reject limit	<pre>ALTER TABLE admin_ext_employees REJECT LIMIT 100;</pre>
<code>DEFAULT DIRECTORY</code>	Changes the default directory specification	<pre>ALTER TABLE admin_ext_employees DEFAULT DIRECTORY admin_dat2_dir;</pre>
<code>ACCESS PARAMETERS</code>	Allows access parameters to be changed without dropping and re-creating the external table metadata	<pre>ALTER TABLE admin_ext_employees ACCESS PARAMETERS (FIELDS TERMINATED BY ';');</pre>
<code>LOCATION</code>	Allows data sources to be changed without dropping and re-creating the external table metadata	<pre>ALTER TABLE admin_ext_employees LOCATION ('empxt3.txt', 'empxt4.txt');</pre>
<code>PARALLEL</code>	No difference from regular tables. Allows degree of parallelism to be changed.	No new syntax
<code>ADD COLUMN</code>	No difference from regular tables. Allows a column to be added to an external table.	No new syntax

ALTER TABLE Clause	Description	Example
MODIFY COLUMN	No difference from regular tables. Allows an external table column to be modified.	No new syntax
DROP COLUMN	No difference from regular tables. Allows an external table column to be dropped.	No new syntax
RENAME TO	No difference from regular tables. Allows external table to be renamed.	No new syntax

Dropping External Tables

For an external table, the `DROP TABLE` statement removes only the table metadata in the database. It has no affect on the actual data, which resides outside of the database.

System and Object Privileges for External Tables

System and object privileges for external tables are a subset of those for regular table. Only the following system privileges are applicable to external tables:

- `CREATE ANY TABLE`
- `ALTER ANY TABLE`
- `DROP ANY TABLE`
- `SELECT ANY TABLE`

Only the following object privileges are applicable to external tables:

- `ALTER`
- `SELECT`

However, object privileges associated with a directory are:

- `READ`
- `WRITE`

For external tables, `READ` privileges are required on directory objects that contain data sources, while `WRITE` privileges are required for directory objects containing bad, log, or discard files.

Viewing Information About Tables

The following views allow you to access information about tables.

View	Description
DBA_TABLES ALL_TABLES USER_TABLES	DBA view describes all relational tables in the database. ALL view describes all tables accessible to the user. USER view is restricted to tables owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_TAB_COLUMNS ALL_TAB_COLUMNS USER_TAB_COLUMNS	These views describe the columns of tables, views, and clusters in the database. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_ALL_TABLES ALL_ALL_TABLES USER_ALL_TABLES	These views describe all relational and <i>object tables</i> in the database. Object tables are not specifically discussed in this book.
DBA_TAB_COMMENTS ALL_TAB_COMMENTS USER_TAB_COMMENTS	These views display comments for tables and views. Comments are entered using the COMMENT statement.
DBA_COL_COMMENTS ALL_COL_COMMENTS USER_COL_COMMENTS	These views display comments for table and view columns. Comments are entered using the COMMENT statement.
DBA_EXTERNAL_TABLES ALL_EXTERNAL_TABLES USER_EXTERNAL_TABLES	These views list the specific attributes of external tables in the database.
DBA_EXTERNAL_LOCATIONS ALL_EXTERNAL_LOCATIONS USER_EXTERNAL_LOCATIONS	These views list the data sources for external tables.
DBA_TAB_HISTOGRAMS ALL_TAB_HISTOGRAMS USER_TAB_HISTOGRAMS	These views describe histograms on tables and views.
DBA_TAB_COL_STATISTICS ALL_TAB_COL_STATISTICS USER_TAB_COL_STATISTICS	These views provide column statistics and histogram information extracted from the related TAB_COLUMNS views.

View	Description
DBA_TAB_MODIFICATIONS ALL_TAB_MODIFICATIONS USER_TAB_MODIFICATIONS	These views describe tables that have been modified since the last time table statistics were gathered on them. The views are populated only for tables with the <code>MONITORING</code> attribute. They are not populated immediately, but after a time lapse (usually 3 hours).
DBA_UNUSED_COL_TABS ALL_UNUSED_COL_TABS USER_UNUSED_COL_TABS	These views list tables with unused columns, as marked by the <code>ALTER TABLE ... SET UNUSED</code> statement.
DBA_PARTIAL_DROP_TABS ALL_PARTIAL_DROP_TABS USER_PARTIAL_DROP_TABS	These views list tables that have partially completed <code>DROP COLUMN</code> operations. These operations could be incomplete because the operation was interrupted by the user or a system crash.

See Also:

- ["Viewing Information About Tables"](#) on page 15-40
- *Oracle9i Database Reference* for complete descriptions of these views
- *Oracle9i Application Developer's Guide - Object-Relational Features* for information about object tables
- *Oracle9i Database Performance Tuning Guide and Reference* for information about histograms and generating statistics for tables
- ["Analyzing Tables, Indexes, and Clusters"](#) on page 21-3

Managing Indexes

This chapter discusses the management of indexes, and contains the following topics:

- [Guidelines for Managing Indexes](#)
- [Creating Indexes](#)
- [Altering Indexes](#)
- [Monitoring Space Use of Indexes](#)
- [Dropping Indexes](#)
- [Viewing Index Information](#)

See Also: [Chapter 14, "Managing Space for Schema Objects"](#) is recommended reading before attempting tasks described in this chapter.

Guidelines for Managing Indexes

Indexes are optional structures associated with tables and clusters that allow SQL statements to execute more quickly against a table. Just as the index in this manual helps you locate information faster than if there were no index, an Oracle index provides a faster access path to table data. You can use indexes without rewriting any queries. Your results are the same, but you see them more quickly.

Oracle provides several indexing schemes that provide complementary performance functionality. These are:

- B-tree indexes—the default and the most common
- B-tree cluster indexes—defined specifically for cluster
- Hash cluster indexes—defined specifically for a hash cluster
- Global and local indexes—relate to partitioned tables and indexes
- Reverse key indexes—most useful for Oracle Real Application Cluster applications
- Bitmap indexes—compact; work best for columns with a small set of values
- Function-based indexes—contain the precomputed value of a function/expression
- Domain indexes—specific to an application or cartridge.

Indexes are logically and physically independent of the data in the associated table. Being independent structures, they require storage space. You can create or drop an index without affecting the base tables, database applications, or other indexes. Oracle automatically maintains indexes when you insert, update, and delete rows of the associated table. If you drop an index, all applications continue to work. However, access to previously indexed data might be slower.

This section discusses guidelines for managing indexes and contains the following topics:

- [Create Indexes After Inserting Table Data](#)
- [Index the Correct Tables and Columns](#)
- [Order Index Columns for Performance](#)
- [Limit the Number of Indexes for Each Table](#)
- [Drop Indexes That Are No Longer Required](#)
- [Specify Index Block Space Use](#)

- Estimate Index Size and Set Storage Parameters
- Specify the Tablespace for Each Index
- Consider Parallelizing Index Creation
- Consider Creating Indexes with NOLOGGING
- Consider Costs and Benefits of Coalescing or Rebuilding Indexes
- Consider Cost Before Disabling or Dropping Constraints

See Also:

- *Oracle9i Database Concepts* for conceptual information about indexes and indexing, including descriptions of the various indexing schemes offered by Oracle
- *Oracle9i Database Performance Tuning Guide and Reference* and *Oracle9i Data Warehousing Guide* for information about bitmap indexes
- *Oracle9i Data Cartridge Developer's Guide* for information about defining domain-specific operators and indexing schemes and integrating them into the Oracle database server

Create Indexes After Inserting Table Data

Data is often inserted or loaded into a table using either the SQL*Loader or Import utility. It is more efficient to create an index for a table after inserting or loading the data. If you create one or more indexes before loading data, Oracle then must update every index as each row is inserted.

Creating an index on a table that already has data requires sort space. Some sort space comes from memory allocated for the index's creator. The amount for each user is determined by the initialization parameter `SORT_AREA_SIZE`. Oracle also swaps sort information to and from temporary segments that are only allocated during the index creation in the user's temporary tablespace.

Under certain conditions, data can be loaded into a table with SQL*Loader's direct path load and an index can be created as data is loaded.

See Also: *Oracle9i Database Utilities* for information about using SQL*Loader for direct path load

Index the Correct Tables and Columns

Use the following guidelines for determining when to create an index:

- Create an index if you frequently want to retrieve less than 15% of the rows in a large table. The percentage varies greatly according to the relative speed of a table scan and how clustered the row data is about the index key. The faster the table scan, the lower the percentage; the more clustered the row data, the higher the percentage.
- To improve performance on joins of multiple tables, index columns used for joins.

Note: Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.

- Small tables do not require indexes. If a query is taking too long, then the table might have grown from small to large.

Some columns are strong candidates for indexing. Columns with one or more of the following characteristics are candidates for indexing:

- Values are relatively unique in the column.
- There is a wide range of values (good for regular indexes).
- There is a small range of values (good for bitmap indexes).
- The column contains many nulls, but queries often select all rows having a value. In this case, use the following phrase:

```
WHERE COL_X > -9.99 * power(10,125)
```

Using the above phrase is preferable to:

```
WHERE COL_X IS NOT NULL
```

This is because the first uses an index on COL_X (assuming that COL_X is a numeric column).

Columns with the following characteristics are less suitable for indexing:

- There are many nulls in the column and you do not search on the non-null values.

LONG and LONG RAW columns cannot be indexed.

The size of a single index entry cannot exceed roughly one-half (minus some overhead) of the available space in the data block.

Order Index Columns for Performance

The order of columns in the `CREATE INDEX` statement can affect query performance. In general, specify the most frequently used columns first.

If you create a single index across columns to speed up queries that access, for example, `col1`, `col2`, and `col3`; then queries that access just `col1`, or that access just `col1` and `col2`, are also speeded up. But a query that accessed just `col2`, just `col3`, or just `col2` and `col3` does not use the index.

Limit the Number of Indexes for Each Table

A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified. Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated.

Thus, there is a trade-off between the speed of retrieving data from a table and the speed of updating the table. For example, if a table is primarily read-only, having more indexes can be useful; but if a table is heavily updated, having fewer indexes could be preferable.

Drop Indexes That Are No Longer Required

Consider dropping an index if:

- It does not speed up queries. The table could be very small, or there could be many rows in the table but very few index entries.
- The queries in your applications do not use the index.
- The index must be dropped before being rebuilt.

See Also: ["Monitoring Index Usage"](#) on page 16-21

Specify Index Block Space Use

When an index is created for a table, data blocks of the index are filled with the existing values in the table up to `PCTFREE`. The space reserved by `PCTFREE` for an index block is only used when a new row is inserted into the table and the

corresponding index entry must be placed in the correct index block (that is, between preceding and following index entries).

If no more space is available in the appropriate index block, the indexed value is placed where it belongs (based on the lexical set ordering). Therefore, if you plan on inserting many rows into an indexed table, `PCTFREE` should be high to accommodate the new index values. If the table is relatively static without many inserts, `PCTFREE` for an associated index can be low so that fewer blocks are required to hold the index data.

`PCTUSED` cannot be specified for indexes.

See Also: ["Managing Space in Data Blocks"](#) on page 14-2 for information about the `PCTFREE` parameter

Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one can facilitate better disk space planning and management. You can use the combined estimated size of indexes, along with estimates for tables, rollback segments, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.

Use the estimated size of an individual index to better manage the disk space that the index uses. When an index is created, you can set appropriate storage parameters and improve I/O performance of applications that use the index. For example, assume that you estimate the maximum size of an index before creating it. If you then set the storage parameters when you create the index, fewer extents are allocated for the table's data segment, and all of the index's data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this index.

The maximum size of a single index entry is approximately one-half the data block size.

See Also: ["Setting Storage Parameters"](#) on page 14-8 for specific information about storage parameters

Specify the Tablespace for Each Index

Indexes can be created in any tablespace. An index can be created in the same or different tablespace as the table it indexes. If you use the same tablespace for a table and its index, it can be more convenient to perform database maintenance (such as

tablespace or file backup) or to ensure application availability. All the related data is always online together.

Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace. Disk contention is reduced. But, if you use different tablespaces for a table and its index and one tablespace is offline (containing either data or index), then the statements referencing that table are not guaranteed to work.

Consider Parallelizing Index Creation

You can parallelize index creation, much the same as you can parallelize table creation. Because multiple processes work together to create the index, Oracle can create the index more quickly than if a single server process created the index sequentially.

When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an `INITIAL` value of 5M and a parallel degree of 12 consumes at least 60M of storage during index creation.

See Also:

- *Oracle9i Database Concepts* for more information about parallel execution
- *Oracle9i Data Warehousing Guide* for information about utilizing parallel execution in a datawarehousing environment

Consider Creating Indexes with NOLOGGING

You can create an index and generate minimal redo log records by specifying `NOLOGGING` in the `CREATE INDEX` statement.

Note: Because indexes created using `NOLOGGING` are not archived, perform a backup after you create the index.

Creating an index with `NOLOGGING` has the following benefits:

- Space is saved in the redo log files.
- The time it takes to create the index is decreased.
- Performance improves for parallel creation of large indexes.

In general, the relative performance improvement is greater for larger indexes created without `LOGGING` than for smaller ones. Creating small indexes without `LOGGING` has little effect on the time it takes to create an index. However, for larger indexes the performance improvement can be significant, especially when you are also parallelizing the index creation.

Consider Costs and Benefits of Coalescing or Rebuilding Indexes

Improper sizing or increased growth can produce index fragmentation. To eliminate or reduce fragmentation, you can rebuild or coalesce the index. But before you perform either task weigh the costs and benefits of each option and choose the one that works best for your situation. [Table 16–1](#) is a comparison of the costs and benefits associated with rebuilding and coalescing indexes.

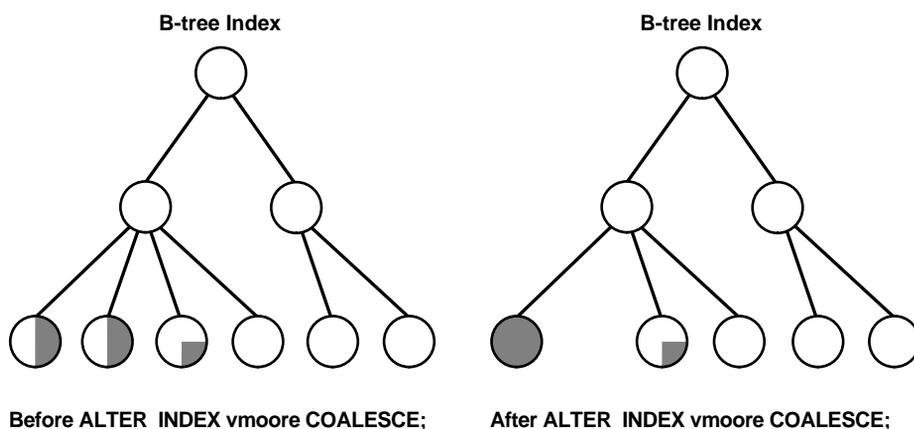
Table 16–1 To Rebuild or Coalesce ... That Is the Question

Rebuild Index	Coalesce Index
Quickly moves index to another tablespace	Cannot move index to another tablespace
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you to quickly change storage and tablespace parameters without having to drop the original index.	Quickly frees up index leaf blocks for use.

In situations where you have B-tree index leaf blocks that can be freed up for reuse, you can merge those leaf blocks using the following statement:

```
ALTER INDEX vmoore COALESCE;
```

[Figure 16–1](#) illustrates the effect of an `ALTER INDEX COALESCE` on the index `vmoore`. Before performing the operation, the first two leaf blocks are 50% full. This means you have an opportunity to reduce fragmentation and completely fill the first block, while freeing up the second. In this example, assume that `PCTFREE=0`.

Figure 16–1 Coalescing Indexes

Consider Cost Before Disabling or Dropping Constraints

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a `UNIQUE` or `PRIMARY KEY` constraint. If the associated index for a `UNIQUE` key or `PRIMARY KEY` constraint is extremely large, you can save time by leaving the constraint enabled rather than dropping and re-creating the large index. You also have the option of explicitly specifying that you want to keep or drop the index when dropping or disabling a `UNIQUE` or `PRIMARY KEY` constraint.

See Also: ["Managing Integrity Constraints"](#) on page 21-14

Creating Indexes

This section describes how to create indexes. To create an index in your own schema, *at least one* of the following conditions must be true:

- The table or cluster to be indexed is in your own schema.
- You have `INDEX` privilege on the table to be indexed.
- You have `CREATE ANY INDEX` system privilege.

To create an index in another schema, *all* of the following conditions must be true:

- You have `CREATE ANY INDEX` system privilege.

- The owner of the other schema has a quota for the tablespaces to contain the index or index partitions, or `UNLIMITED TABLESPACE` system privilege.

This section contains the following topics:

- [Creating an Index Explicitly](#)
- [Creating a Unique Index Explicitly](#)
- [Creating an Index Associated with a Constraint](#)
- [Collecting Incidental Statistics when Creating an Index](#)
- [Creating a Large Index](#)
- [Creating an Index Online](#)
- [Creating a Function-Based Index](#)
- [Creating a Key-Compressed Index](#)

See Also: *Oracle9i SQL Reference* for syntax and restrictions on the use of the `CREATE INDEX`, `ALTER INDEX`, and `DROP INDEX` statements

Creating an Index Explicitly

You can create indexes explicitly (outside of integrity constraints) using the SQL statement `CREATE INDEX`. The following statement creates an index named `emp_ename` for the `ename` column of the `emp` table:

```
CREATE INDEX emp_ename ON emp(ename)
    TABLESPACE users
    STORAGE (INITIAL 20K
    NEXT 20k
    PCTINCREASE 75)
    PCTFREE 0;
```

Notice that several storage settings and a tablespace are explicitly specified for the index. If you do not specify storage options (such as `INITIAL` and `NEXT`) for an index, the default storage options of the default or specified tablespace are automatically used.

Creating a Unique Index Explicitly

Indexes can be unique or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Nonunique indexes do not impose this restriction on the column values.

Use the `CREATE UNIQUE INDEX` statement to create a unique index. The following example creates a unique index:

```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
    TABLESPACE indx;
```

Alternatively, you can define `UNIQUE` integrity constraints on the desired columns. Oracle enforces `UNIQUE` integrity constraints by automatically defining a unique index on the unique key. This is discussed in the following section. However, it is advisable that any index that exists for query performance, including unique indexes, be created explicitly

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for more information about creating an index for performance

Creating an Index Associated with a Constraint

Oracle enforces a `UNIQUE` key or `PRIMARY KEY` integrity constraint on a table by creating a unique index on the unique key or primary key. This index is automatically created by Oracle when the constraint is enabled. No action is required by you when you issue the `CREATE TABLE` or `ALTER TABLE` statement to create the index, but you can optionally specify a `USING INDEX` clause to exercise control over its creation. This includes both when a constraint is defined and enabled, and when a defined but disabled constraint is enabled.

To enable a `UNIQUE` or `PRIMARY KEY` constraint, thus creating an associated index, the owner of the table must have a quota for the tablespace intended to contain the index, or the `UNLIMITED TABLESPACE` system privilege. A constraint's associated index always assumes the name of the constraint, unless you optionally specify otherwise.

Specifying Storage Options for an Index Associated with a Constraint

You can set the storage options for the indexes associated with `UNIQUE` and `PRIMARY KEY` constraints using the `USING INDEX` clause. The following `CREATE TABLE` statement enables a `PRIMARY KEY` constraint and specifies the associated index's storage options:

```
CREATE TABLE emp (
```

```
empno NUMBER(5) PRIMARY KEY, age INTEGER)
ENABLE PRIMARY KEY USING INDEX
TABLESPACE users
PCTFREE 0;
```

Specifying the Index Associated with a Constraint

If you require more explicit control over the indexes associated with `UNIQUE` and `PRIMARY KEY` constraints, Oracle allows you to:

- Specify an existing index that Oracle is to use to enforce the constraint
- Specify a create index statement that Oracle is to use to create the index and enforce the constraint

These options are specified using the `USING INDEX` clause. The following statements present some examples.

Example 1:

```
CREATE TABLE a (
  a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```

Example 2:

```
CREATE TABLE b(
  b1 INT,
  b2 INT,
  CONSTRAINT bu1 UNIQUE (b1, b2)
    USING INDEX (create unique index bi on b(b1, b2)),
  CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

Example 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

If a single statement creates an index with one constraint and also uses that index for another constraint, the system will attempt to rearrange the clauses to create the index before reusing it.

See Also: ["Managing Integrity Constraints"](#) on page 21-14

Collecting Incidental Statistics when Creating an Index

Oracle provides you with the opportunity to collect statistics at very little resource cost during the creation or rebuilding of an index. These statistics are stored in the data dictionary for ongoing use by the optimizer in choosing a plan for the execution of SQL statements. The following statement computes index, table, and column statistics while building index emp_ename on column ename of table emp:

```
CREATE INDEX emp_ename ON emp(ename)
      COMPUTE STATISTICS;
```

See Also:

- *Oracle9i Database Performance Tuning Guide and Reference* for information about collecting statistics and their use by the optimizer
- ["Analyzing Tables, Indexes, and Clusters"](#) on page 21-3

Creating a Large Index

When creating an extremely large index, consider allocating a larger temporary tablespace for the index creation using the following procedure:

1. Create a new temporary tablespace using the `CREATE TABLESPACE` or `CREATE TEMPORARY TABLESPACE` statement.
2. Use the `TEMPORARY TABLESPACE` option of the `ALTER USER` statement to make this your new temporary tablespace.
3. Create the index using the `CREATE INDEX` statement.
4. Drop this tablespace using the `DROP TABLESPACE` statement. Then use the `ALTER USER` statement to reset your temporary tablespace to your original temporary tablespace.

Using this procedure can avoid the problem of expanding your usual, and usually shared, temporary tablespace to an unreasonably large size that might affect future performance.

Creating an Index Online

You can create and rebuild indexes online. This enables you to update base tables at the same time you are building or rebuilding indexes on that table. You can perform DML operations while the index build is taking place, but DDL operations are not

allowed. Parallel execution is not supported when creating or rebuilding an index online.

The following statements illustrate online index build operations:

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

Note: While you can perform DML operations during an online index build, Oracle recommends that you do not perform major/large DML operations during this procedure. This is because while the DML on the base table is taking place it holds a lock on that resource. The DDL to build the index cannot proceed until the transaction acting on the base table commits or rolls back, thus releasing the lock.

For example, if you want to load rows that total up to 30% of the size of an existing table, you should perform this load before the online index build.

See Also: [Rebuilding an Existing Index](#) on page 16-20

Creating a Function-Based Index

Function-based indexes facilitate queries that qualify a value returned by a function or expression. The value of the function or expression is precomputed and stored in the index.

See Also:

- *Oracle9i Database Concepts*
- *Oracle9i Data Warehousing Guide*

These books provide additional information about function-based indexes.

Features of Function-Based Indexes

Function-based indexes allow you to:

- Create more powerful sorts

You can perform case-insensitive sorts with the `UPPER` and `LOWER` functions, descending order sorts with the `DESC` keyword, and linguistic-based sorts with the `NLSSORT` function.

- Precompute the value of a computationally intensive function and store it in the index

An index can store computationally intensive expression that you access often. When you need to access a value, it is already computed, greatly improving query execution performance.

- Increase the number of situations where the optimizer can perform a range scan instead of a full table scan

For example, consider the expression in the `WHERE` clause below:

```
CREATE INDEX idx ON Example_tab(column_a + column_b);
SELECT * FROM example_tab WHERE column_a + column_b < 10;
```

The optimizer can use a range scan for this query because the index is built on `(column_a + column_b)`. Range scans typically produce fast response times if the predicate selects less than 15% of the rows of a large table. The optimizer can estimate how many rows are selected by expressions more accurately if the expressions are materialized in a function-based index. (Expressions of function-based indexes are represented as virtual columns and analyze operation using the `DBMS_STATS` package can build histograms on such columns.)

- Enable true descending order indexes

They are treated as a special case of function-based indexes.

Note: Oracle sorts columns with the `DESC` keyword in descending order. Such indexes are treated as function-based indexes. Descending indexes cannot be bitmapped or reverse, and cannot be used in bitmapped optimizations. To get the pre-Oracle 8.1 release `DESC` behavior, remove the `DESC` keyword from the `CREATE INDEX` statement.

- Create indexes on object columns and `REF` columns

Methods that describe objects can be used as functions on which to build indexes. For example, you can use the `MAP` method to build indexes on an object type column.

See Also:

- *Oracle9i Database Globalization Support Guide* for information about the `NLSSORT` function
- *Oracle9i Database Performance Tuning Guide and Reference* for information about the optimizer
- *Oracle9i Application Developer's Guide - Object-Relational Features* for information about object and `REF` columns

How Function-Based Indexes Work

For the creation of a function-based index in your own schema, you must be granted the `QUERY REWRITE` system privileges. To create the index in another schema or on another schema's tables, you must have the `CREATE ANY INDEX` and `GLOBAL QUERY REWRITE` privileges.

You must have the following initialization parameters defined to create a function-based index:

- `QUERY_REWRITE_INTEGRITY` set to `TRUSTED`
- `QUERY_REWRITE_ENABLED` set to `TRUE`
- `COMPATIBLE` set to `8.1.0.0.0` or a greater value

Additionally, to use a function-based index:

- The table must be analyzed after the index is created.
- The query must be guaranteed not to need any `NULL` values from the indexed expression, since `NULL` values are not stored in indexes.

Note: `CREATE INDEX` stores the timestamp of the most recent function used in the function-based index. This timestamp is updated when the index is validated. When performing tablespace point-in-time recovery of a function-based index, if the timestamp on the most recent function used in the index is newer than the timestamp stored in the index, then the index is marked invalid. You must use the `ANALYZE INDEX ... VALIDATE STRUCTURE` statement to validate this index.

To illustrate a function-based index, let's consider the following statement that defines a function-based index (`area_index`) defined on the function `area(geo)`:

```
CREATE INDEX area_index ON rivers (area(geo));
```

In the following SQL statement, when `area(geo)` is referenced in the `WHERE` clause, the optimizer considers using the index `area_index`.

```
SELECT id, geo, area(geo), desc
       FROM rivers
       WHERE Area(geo) >5000;
```

Table owners should have `EXECUTE` privileges on the functions used in function-based indexes.

Because a function-based index depends upon any function it is using, it can be invalidated when a function changes. If the function is valid, you can use an `ALTER INDEX ... ENABLE` statement to enable a function-based index that has been disabled. The `ALTER INDEX ... DISABLE` statement allows you to disable the use of a function-based index. Consider doing this if you are working on the body of the function.

Examples of Function-Based Indexes

Some examples of using function-based indexes follow.

Example: Function-Based Index for Case-Insensitive Searches The following statement creates function-based index `idx` on table `emp` based on an uppercase evaluation of the `ename` column:

```
CREATE INDEX idx ON emp (UPPER(ename));
```

Now the `SELECT` statement uses the function-based index on `UPPER(ename)` to retrieve all employees with names that start with `JOH`:

```
SELECT * FROM emp WHERE UPPER(ename) LIKE 'JOH%';
```

This example also illustrates a case-insensitive search.

Example: Precomputing Arithmetic Expressions with a Function-Based Index This statement creates a function-based index on an expression:

```
CREATE INDEX idx ON t (a + b * (c - 1), a, b);
```

`SELECT` statements can use either an index range scan (in the following `SELECT` statement the expression is a prefix of the index) or index full scan (preferable when the index specifies a high degree of parallelism).

```
SELECT a FROM t WHERE a + b * (c - 1) < 100;
```

Examples: Function-Based Index for Language-Dependent Sorting You can use function-based indexes to support a linguistic sort index. `NLSSORT` is a function that returns a sort key that has been given a string. Thus, if you want to build an index on `name` using `NLSSORT`, issue the following statement:

```
CREATE INDEX nls_index ON t_table (NLSSORT(name, 'NLS_SORT = German'));
```

This statement creates index `nls_index` on table `t_table` with the collation sequence `German`.

Now, the following statement selects from `t_table` using the `NLS_SORT` index:

```
SELECT * FROM t_table ORDER BY name;
```

Rows are ordered using the collation sequence in `German`.

The following example combines a case-insensitive sort and a language sort:

```
CREATE INDEX emp_i ON emp
    UPPER ((ename), NLSSORT(ename));
```

Here, an `NLS_SORT` specification does not appear in the `NLSSORT` argument because `NLSSORT` looks at the session setting for the language of the linguistic sort key. The previous example illustrated a case where `NLS_SORT` was specified.

Creating a Key-Compressed Index

Creating an index using key compression enables you to eliminate repeated occurrences of key column prefix values.

Key compression breaks an index key into a prefix and a suffix entry. Compression is achieved by sharing the prefix entries among all the suffix entries in an index block. This sharing can lead to huge savings in space, allowing you to store more keys for each index block while improving performance.

Key compression can be useful in the following situations:

- You have a non-unique index where `ROWID` is appended to make the key unique. If you use key compression here, the duplicate key is stored as a prefix entry on the index block without the `ROWID`. The remaining rows become suffix entries consisting of only the `ROWID`.
- You have a unique multi-column index.

You enable key compression using the `COMPRESS` clause. The prefix length (as the number of key columns) can also be specified to identify how the key columns are

broken into a prefix and suffix entry. For example, the following statement compresses duplicate occurrences of a key in the index leaf block:

```
CREATE INDEX emp_ename ON emp(ename)
    TABLESPACE users
    COMPRESS 1;
```

The `COMPRESS` clause can also be specified during rebuild. For example, during rebuild you can disable compression as follows:

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

See Also: *Oracle9i Database Concepts* for a more detailed discussion of key compression

Altering Indexes

To alter an index, your schema must contain the index or you must have the `ALTER ANY INDEX` system privilege. Among the actions allowed by the `ALTER INDEX` statement are:

- Rebuild or coalesce an existing index
- Deallocate unused space or allocate a new extent
- Specify parallel execution (or not) and alter the degree of parallelism
- Alter storage parameters or physical attributes
- Specify `LOGGING` or `NOLOGGING`
- Enable or disable key compression
- Mark the index unusable
- Start or stop the monitoring of index usage

You cannot alter an index's column structure.

More detailed discussions of some of these operations are contained in the following sections:

- [Altering Storage Characteristics of an Index](#)
- [Rebuilding an Existing Index](#)
- [Monitoring Index Usage](#)

Altering Storage Characteristics of an Index

Alter the storage parameters of any index, including those created by Oracle to enforce primary and unique key integrity constraints, using the `ALTER INDEX` statement. For example, the following statement alters the `emp_ename` index:

```
ALTER INDEX emp_ename
    STORAGE (PCTINCREASE 50);
```

The storage parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the index.

For indexes that implement integrity constraints, you can choose to adjust storage parameters by issuing an `ALTER TABLE` statement that includes the `USING INDEX` subclause of the `ENABLE` clause. For example, the following statement changes the storage options of the index created on table `emp` to enforce the primary key constraint:

```
ALTER TABLE emp
    ENABLE PRIMARY KEY USING INDEX
    PCTFREE 5;
```

Rebuilding an Existing Index

Before rebuilding an existing index, compare the costs and benefits associated with rebuilding to those associated with coalescing indexes as described in [Table 16-1](#) on page 16-8.

When you rebuild an index, you use an existing index as the data source. Creating an index in this manner enables you to change storage characteristics or move to a new tablespace. Rebuilding an index based on an existing data source removes intra-block fragmentation. Compared to dropping the index and using the `CREATE INDEX` statement, re-creating an existing index offers better performance.

The following statement rebuilds the existing index `emp_name`:

```
ALTER INDEX emp_name REBUILD;
```

The `REBUILD` clause must immediately follow the index name, and precede any other options. It cannot be used in conjunction with the `DEALLOCATE UNUSED` clause.

If have the option of rebuilding the index online. The following statement rebuilds the `emp_name` index online:

```
ALTER INDEX REBUILD ONLINE;
```

If you do not have the space required to rebuild an index, you can choose instead to coalesce the index. Coalescing an index can also be done online.

See Also:

- ["Creating an Index Online"](#) on page 16-13
- ["Monitoring Space Use of Indexes"](#) on page 16-21

Monitoring Index Usage

Oracle provides a means of monitoring indexes to determine if they are being used or not used. If it is determined that an index is not being used, then it can be dropped, thus eliminating unnecessary statement overhead.

To start monitoring an index's usage, issue this statement:

```
ALTER INDEX index MONITORING USAGE;
```

Later, issue the following statement to stop the monitoring:

```
ALTER INDEX index NOMONITORING USAGE;
```

The view `V$OBJECT_USAGE` can be queried for the index being monitored to see if the index has been used. The view contains a `USED` column whose value is `YES` or `NO`, depending upon if the index has been used within the time period being monitored. The view also contains the start and stop times of the monitoring period, and a `MONITORING` column (`YES/NO`) to indicate if usage monitoring is currently active.

Each time that you specify `MONITORING USAGE`, the `V$OBJECT_USAGE` view is reset for the specified index. The previous usage information is cleared or reset, and a new start time is recorded. When you specify `NOMONITORING USAGE`, no further monitoring is performed, and the end time is recorded for the monitoring period. Until the next `ALTER INDEX ... MONITORING USAGE` statement is issued, the view information is left unchanged.

Monitoring Space Use of Indexes

If key values in an index are inserted, updated, and deleted frequently, the index can lose its acquired space efficiently over time. Monitor an index's efficiency of space usage at regular intervals by first analyzing the index's structure, using the

`ANALYZE INDEX ... VALIDATE STRUCTURE` statement, and then querying the `INDEX_STATS` view:

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

The percentage of an index's space usage varies according to how often index keys are inserted, updated, or deleted. Develop a history of an index's average efficiency of space usage by performing the following sequence of operations several times:

- Analyzing statistics
- Validating the index
- Checking `PCTUSED`
- Dropping and rebuilding (or coalescing) the index

When you find that an index's space usage drops below its average, you can condense the index's space by dropping the index and rebuilding it, or coalescing it.

See Also: ["Analyzing Tables, Indexes, and Clusters"](#) on page 21-3

Dropping Indexes

To drop an index, the index must be contained in your schema, or you must have the `DROP ANY INDEX` system privilege.

Some reasons for dropping an index include:

- The index is no longer required.
- The index is not providing anticipated performance improvements for queries issued against the associated table. For example, the table might be very small, or there might be many rows in the table but very few index entries.
- Applications do not use the index to query the data.
- The index has become invalid and must be dropped before being rebuilt.
- The index has become too fragmented and must be dropped before being rebuilt.

When you drop an index, all extents of the index's segment are returned to the containing tablespace and become available for other objects in the tablespace.

How you drop an index depends on whether you created the index explicitly with a `CREATE INDEX` statement, or implicitly by defining a key constraint on a table. If you created the index explicitly with the `CREATE INDEX` statement, then you can

drop the index with the `DROP INDEX` statement. The following statement drops the `emp_ename` index:

```
DROP INDEX emp_ename;
```

You cannot drop only the index associated with an enabled `UNIQUE` key or `PRIMARY KEY` constraint. To drop a constraint's associated index, you must disable or drop the constraint itself.

Note: If a table is dropped, all associated indexes are dropped automatically.

See Also: ["Managing Integrity Constraints"](#) on page 21-14

Viewing Index Information

The following views display information about indexes:

View	Description
DBA_INDEXES ALL_INDEXES USER_INDEXES	DBA view describes indexes on all tables in the database. ALL view describes indexes on all tables accessible to the user. USER view is restricted to indexes owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_IND_COLUMNS ALL_IND_COLUMNS USER_IND_COLUMNS	These views describe the columns of indexes on tables. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_IND_EXPRESSIONS ALL_IND_EXPRESSIONS USER_IND_EXPRESSIONS	These views describe the expressions of function-based indexes on tables.
INDEX_STATS	Stores information from the last ANALYZE INDEX ... VALIDATE STRUCTURE statement.
INDEX_HISTOGRAM	Stores information from the last ANALYZE INDEX ... VALIDATE STRUCTURE statement.
V\$OBJECT_USAGE	Contains index usage information produced by the ALTER INDEX ... MONITORING USAGE functionality.

See Also: *Oracle9i Database Reference* for a complete description of these views

Managing Partitioned Tables and Indexes

This chapter describes various aspects of managing partitioned tables and indexes, and contains the following topics:

- [What Are Partitioned Tables and Indexes?](#)
- [Partitioning Methods](#)
- [Creating Partitioned Tables](#)
- [Maintaining Partitioned Tables](#)
- [Partitioned Tables and Indexes Examples](#)
- [Viewing Information About Partitioned Tables and Indexes](#)

See Also: [Chapter 14, "Managing Space for Schema Objects"](#) is recommended reading before attempting tasks described in this chapter.

What Are Partitioned Tables and Indexes?

Today's enterprises frequently run mission critical databases containing upwards of several hundred gigabytes and, in many cases, several terabytes of data. These enterprises are challenged by the support and maintenance requirements of very large databases (VLDB), and must devise methods to meet those challenges.

One way to meet VLDB demands is to create and use **partitioned tables and indexes**. Partitioned tables allow your data to be broken down into smaller, more manageable pieces called **partitions**, or even **subpartitions**. Indexes can be partitioned in similar fashion. Each partition is stored in its own segment and can be managed individually. It can function independently of the other partitions, thus providing a structure that can be better tuned for availability and performance.

If you are using parallel execution, partitions provide another means of parallelization. Operations on partitioned tables and indexes are performed in parallel by assigning different parallel execution servers to different partitions of the table or index.

Partitions and subpartitions of a table or index all share the same logical attributes. For example, all partitions (or subpartitions) in a table share the same column and constraint definitions, and all partitions (or subpartitions) of an index share the same index options. They can, however, have different physical attributes (such as TABLESPACE).

Although you are not required to keep each table or index partition (or subpartition) in a separate tablespace, it is to your advantage to do so. Storing partitions in separate tablespaces enables you to:

- Reduce the possibility of data corruption in multiple partitions
- Back up and recover each partition independently
- Control the mapping of partitions to disk drives (important for balancing I/O load)
- Improve manageability, availability, and performance

Partitioning is transparent to existing applications and standard DML statements run against partitioned tables. However, an application can be programmed to take advantage of partitioning by using partition-extended table or index names in DML.

You can use the SQL*Loader, Import, and Export utilities to load or unload data stored in partitioned tables. These utilities are all partition and subpartition aware.

See Also:

- *Oracle9i Database Concepts* contains more information about partitioning. Before the first time you attempt to create a partitioned table or index, or perform maintenance operations on any partitioned table, it is recommended that you review the information contained in that book.
- *Oracle9i Data Warehousing Guide* and *Oracle9i Database Concepts* contain information about parallel execution
- *Oracle9i Database Utilities* describes the SQL*Loader, Import, and Export utilities.

Partitioning Methods

There are several partitioning methods offered by Oracle:

- Range partitioning
- Hash partitioning
- List partitioning
- Composite range-hash partitioning
- Composite range-list partitioning

Indexes, as well as tables, can be partitioned. A global index can only be partitioned by range, but it can be defined on any type of partitioned, or nonpartitioned, table. It can require more maintenance than a local index.

A local index is constructed so that it reflects the structure of the underlying table. It is equipartitioned with the underlying table, meaning that it is partitioned on the same columns as the underlying table, creates the same number of partitions or subpartitions, and gives them the same partition bounds as corresponding partitions of the underlying table. For local indexes, index partitioning is maintained automatically when partitions are affected by maintenance activity. This ensures that the index remains equipartitioned with the underlying table.

The following sections can help you decide on a partitioning method appropriate for your needs:

- [When to Use the Range Partitioning Method](#)
- [When to Use the Hash Partitioning Method](#)
- [When to Use the List Partitioning Method](#)

- [When to Use the Composite Range-Hash Partitioning Method](#)
- [When to Use the Composite Range-List Partitioning Method](#)

When to Use the Range Partitioning Method

Use range partitioning to map rows to partitions based on ranges of column values. This type of partitioning is useful when dealing with data that has logical ranges into which it can be distributed; for example, months of the year. Performance is best when the data evenly distributes across the range. If partitioning by range causes partitions to vary dramatically in size because of unequal distribution, you may want to consider one of the other methods of partitioning.

When creating range partitions, you must specify:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds

The example below creates a table of four partitions, one for each quarter's sales. The columns `sale_year`, `sale_month`, and `sale_day` are the **partitioning columns**, while their values constitute a specific row's **partitioning key**. The `VALUES LESS THAN` clause determines the **partition bound**: rows with partitioning key values that compare less than the ordered list of values specified by the clause are stored in the partition. Each partition is given a name (`sales_q1`, `sales_q2`, ...), and each partition is contained in a separate tablespace (`tsa`, `tsb`, ...).

```
CREATE TABLE sales
( invoice_no NUMBER,
  sale_year INT NOT NULL,
  sale_month INT NOT NULL,
  sale_day INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
( PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01)
  TABLESPACE tsa,
  PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01)
  TABLESPACE tsb,
  PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01)
  TABLESPACE tsc,
  PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01)
  TABLESPACE tsd );
```

A row with `sale_year=1999`, `sale_month=8`, and `sale_day=1` has a partitioning key of `(1999, 8, 1)` and would be stored in partition `sales_q3`.

When to Use the Hash Partitioning Method

Use hash partitioning if your data does not easily lend itself to range partitioning, but you would like to partition for performance and manageability reasons. Hash partitioning provides a method of evenly distributing data across a specified number of partitions. Rows are mapped into partitions based on a hash value of the partitioning key. Creating and using hash partitions gives you a highly tunable method of data placement, because you can influence availability and performance by spreading these evenly sized partitions across I/O devices (striping).

To create hash partitions you specify the following:

- Partitioning method: hash
- Partitioning column(s)
- Number of partitions or individual partition descriptions

The following example creates a hash-partitioned table. The partitioning column is `id`, four partitions are created and assigned system generated names, and they are placed in four named tablespaces (`gear1`, `gear2`, ...).

```
CREATE TABLE scubagear
  (id NUMBER,
   name VARCHAR2 (60))
PARTITION BY HASH (id)
PARTITIONS 4
STORE IN (gear1, gear2, gear3, gear4);
```

When to Use the List Partitioning Method

Use list partitioning when you require explicit control over how rows map to partitions. You can specify a list of discrete values for the partitioning column in the description for each partition. This is different from range partitioning, where a range of values is associated with a partition, and from hash partitioning, where the user has no control of the row to partition mapping.

The list partitioning method is specifically designed for modeling data distributions that follow discrete values. This cannot be easily done by range or hash partitioning because:

- Range partitioning assumes a natural range of values for the partitioning column. It is not possible to group together out-of-range values partitions.

- Hash partitioning allows no control over the distribution of data because the data is distributed over the various partitions using the system hash function. Again, this makes it impossible to logically group together discrete values for the partitioning columns into partitions.

Further, list partitioning allows unordered and unrelated sets of data to be grouped and organized together very naturally.

Unlike the range and hash partitioning methods, multi-column partitioning is not supported for list partitioning. If a table is partitioned by list, the partitioning key can consist only of a single column of the table. Otherwise all columns that can be partitioned by the range or hash methods can be partitioned by the list partitioning method.

When creating list partitions, you must specify:

- Partitioning method: list
- Partitioning column
- Partition descriptions, each specifying a list of literal values (a **value list**), which are the discrete values of the partitioning column that qualify a row to be included in the partition

The following example creates a list-partitioned table. It creates table `q1_sales_by_region` which is partitioned by regions consisting of groups of states.

```
CREATE TABLE q1_sales_by_region
  (deptno number,
   deptname varchar2(20),
   quarterly_sales number(10, 2),
   state varchar2(2))
PARTITION BY LIST (state)
  (PARTITION q1_northwest VALUES ('OR', 'WA'),
   PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
   PARTITION q1_northeast VALUES ('NY', 'VM', 'NJ'),
   PARTITION q1_southeast VALUES ('FL', 'GA'),
   PARTITION q1_northcentral VALUES ('SD', 'WI'),
   PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

A row is mapped to a partition by checking whether the value of the partitioning column for a row matches a value in the value list that describes the partition.

For example, some sample rows are inserted as follows:

- (10, 'accounting', 100, 'WA') maps to partition `q1_northwest`
- (20, 'R&D', 150, 'OR') maps to partition `q1_northwest`

- (30, 'sales', 100, 'FL') maps to partition `q1_southeast`
- (40, 'HR', 10, 'TX') maps to partition `q1_southwest`
- (50, 'systems engineering', 10, 'CA') does not map to any partition in the table and returns an error

One of the interesting things to note about list partitioning is that there is no apparent sense of ordering between partitions (unlike range partitioning). You can also specify a **default partition** into which rows that do not map to any other partition are mapped. If a default partition were specified in the above example, the state CA would map to that partition.

When to Use the Composite Range-Hash Partitioning Method

Range-hash partitioning partitions data using the range method, and within each partition, subpartitions it using the hash method. These composite partitions are ideal for both historical data and striping, and provide improved manageability of range partitioning and data placement, as well as the parallelism advantages of hash partitioning.

When creating range-hash partitions, you specify the following:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds
- Subpartitioning method: hash
- Subpartitioning column(s)
- Number of subpartitions for each partition or descriptions of subpartitions

The following statement creates a range-hash partitioned table. In this example, three range partitions are created, each containing eight subpartitions. Because the subpartitions are not named, system generated names are assigned, but the `STORE IN` clause distributes them across the 4 specified tablespaces (`ts1, ...,ts4`).

```
CREATE TABLE scubagear (equipno NUMBER, equipname VARCHAR(32), price NUMBER)
PARTITION BY RANGE (equipno) SUBPARTITION BY HASH(equipname)
SUBPARTITIONS 8 STORE IN (ts1, ts2, ts3, ts4)
(PARTITION p1 VALUES LESS THAN (1000),
PARTITION p2 VALUES LESS THAN (2000),
PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

The partitions of a range-hash partitioned table are logical structures only, as their data is stored in the segments of their subpartitions. As with partitions, these subpartitions share the same logical attributes. Unlike range partitions in a range-partitioned table, the subpartitions cannot have different physical attributes from the owning partition, although they are not required to reside in the same tablespace.

When to Use the Composite Range-List Partitioning Method

Like the composite range-hash partitioning method, the composite range-list partitioning method provides for partitioning based on a two level hierarchy. The first level of partitioning is based on a range of values, as for range partitioning; the second level is based on discrete values, as for list partitioning. This form of composite partitioning is well suited for historical data, but allows you to further group the rows of data based on unordered or unrelated column values.

When creating range-list partitions, you specify the following:

- Partitioning method: range
- Partitioning column(s)
- Partition descriptions identifying partition bounds
- Subpartitioning method: list
- Subpartitioning column
- Subpartition descriptions, each specifying a list of literal values (a **value list**), which are the discrete values of the subpartitioning column that qualify a row to be included in the subpartition

The following example illustrates how range-list partitioning might be used. The example tracks sales data of products by quarters and within each quarter, groups it by specified states.

```
CREATE TABLE quarterly_regional_sales
  (deptno number, item_no varchar2(20),
   txn_date date, txn_amount number, state varchar2(2))
TABLESPACE ts4
PARTITION BY RANGE (txn_date)
SUBPARTITION BY LIST (state)
(PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
 (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
  SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
  SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
  SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'))
```

```

SUBPARTITION q1_1999_northcentral VALUES ('SD', 'WI'),
SUBPARTITION q1_1999_southcentral VALUES ('OK', 'TX')
),
PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
(SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
),
PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
(SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
SUBPARTITION q3_1999_northeast VALUES ('NY', 'VM', 'NJ'),
SUBPARTITION q3_1999_southeast VALUES ('FL', 'GA'),
SUBPARTITION q3_1999_northcentral VALUES ('SD', 'WI'),
SUBPARTITION q3_1999_southcentral VALUES ('OK', 'TX')
),
PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
(SUBPARTITION q4_1999_northwest VALUES ('OR', 'WA'),
SUBPARTITION q4_1999_southwest VALUES ('AZ', 'UT', 'NM'),
SUBPARTITION q4_1999_northeast VALUES ('NY', 'VM', 'NJ'),
SUBPARTITION q4_1999_southeast VALUES ('FL', 'GA'),
SUBPARTITION q4_1999_northcentral VALUES ('SD', 'WI'),
SUBPARTITION q4_1999_southcentral VALUES ('OK', 'TX')
)
);

```

A row is mapped to a partition by checking whether the value of the partitioning column for a row falls within a specific partition range. The row is then mapped to a subpartition within that partition by identifying the subpartition whose descriptor value list contains a value matching the subpartition column value.

For example, some sample rows are inserted as follows:

- (10, 4532130, '23-Jan-1999', 8934.10, 'WA') maps to subpartition q1_1999_northwest
- (20, 5671621, '15-May-1999', 49021.21, 'OR') maps to subpartition q2_1999_northeast
- (30, 9977612, '07-Sep-1999', 30987.90, 'FL') maps to subpartition q3_1999_southeast
- (40, 9977612, '29-Nov-1999', 67891.45, 'TX') maps to subpartition q4_1999_southwest

- (40, 4532130, '5-Jan-2000', 897231.55, 'TX') does not map to any partition in the table and raises an error
- (50, 5671621, '17-Dec-1999', 76123.35, 'CA') does not map to any subpartition in the table and raises an error

The partitions of a range-list partitioned table are logical structures only, as their data is stored in the segments of their subpartitions. The list subpartitions have the same characteristics as list partitions. You can specify a default subpartition, just as you specify a default partition for list partitioning.

Creating Partitioned Tables

Creating a partitioned table or index is very similar to creating a non-partitioned table or index (as described in [Chapter 15, "Managing Tables"](#)), but you include a partitioning clause. The partitioning clause, and subclauses, that you include depend upon the type of partitioning you want to achieve.

You can partition both regular (heap organized) tables and index-organized tables, including those containing LOB columns. You can create nonpartitioned global indexes, range-partitioned global indexes, and local indexes on partitioned tables.

When you create (or alter) a partitioned table, a row movement clause, either `ENABLE ROW MOVEMENT` or `DISABLE ROW MOVEMENT` can be specified. This clause either enables or disables the migration of a row to a new partition if its key is updated. The default is `DISABLE ROW MOVEMENT`.

The following sections present details and examples of creating partitions for the various types of partitioned tables and indexes:

- [Creating Range-Partitioned Tables](#)
- [Creating Hash-Partitioned Tables](#)
- [Creating List-Partitioned Tables](#)
- [Creating Composite Range-Hash Partitioned Tables](#)
- [Creating Composite Range-List Partitioned Tables](#)
- [Using Subpartition Templates to Describe Composite Partitioned Tables](#)
- [Creating Partitioned Index-Organized Tables](#)
- [Partitioning Restrictions for Multiple Block Sizes](#)

See Also:

- *Oracle9i SQL Reference* for the exact syntax of the partitioning clauses for creating and altering partitioned tables and indexes, any restrictions on their use, and specific privileges required for creating and altering tables
- *Oracle9i Application Developer's Guide - Large Objects (LOBs)* and *Oracle9i Application Developer's Guide - Fundamentals* for information about creating partitioned tables containing columns with LOBs or other objects stored as LOBs

Creating Range-Partitioned Tables

The `PARTITION BY RANGE` clause of the `CREATE TABLE` statement specifies that the table is to be range-partitioned. The `PARTITION` clauses identify the individual partition ranges, and optional subclauses of a `PARTITION` clause can specify physical and other attributes specific to a partition's segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

In this example, more complexity is added to the example presented earlier for a range-partitioned table. Storage parameters and a `LOGGING` attribute are specified at the table level. These replace the corresponding defaults inherited from the tablespace level for the table itself, and are inherited by the range partitions. However, since there was little business in the first quarter, the storage attributes for partition `sales_q1` are made smaller. The `ENABLE ROW MOVEMENT` clause is specified to allow the migration of a row to a new partition if an update to a key value is made that would place the row in a different partition.

```
CREATE TABLE sales
  ( invoice_no NUMBER,
    sale_year  INT NOT NULL,
    sale_month INT NOT NULL,
    sale_day   INT NOT NULL )
STORAGE (INITIAL 100K NEXT 50K) LOGGING
PARTITION BY RANGE ( sale_year, sale_month, sale_day)
  ( PARTITION sales_q1 VALUES LESS THAN ( 1999, 04, 01 )
    TABLESPACE tsa STORAGE (INITIAL 20K, NEXT 10K),
    PARTITION sales_q2 VALUES LESS THAN ( 1999, 07, 01 )
    TABLESPACE tsb,
    PARTITION sales_q3 VALUES LESS THAN ( 1999, 10, 01 )
    TABLESPACE tsc,
    PARTITION sales_q4 VALUES LESS THAN ( 2000, 01, 01 )
    TABLESPACE tsd)
```

```
ENABLE ROW MOVEMENT;
```

The rules for creating range-partitioned global indexes are similar to those for creating range-partitioned tables. The following is an example of creating a range-partitioned global index on `sales_month` for the above table. Each index partition is named but is stored in the default tablespace for the index.

```
CREATE INDEX month_ix ON sales(sales_month)
  GLOBAL PARTITION BY RANGE(sales_month)
    (PARTITION pm1_ix VALUES LESS THAN (2)
    PARTITION pm2_ix VALUES LESS THAN (3)
    PARTITION pm3_ix VALUES LESS THAN (4)
    PARTITION pm4_ix VALUES LESS THAN (5)
    PARTITION pm5_ix VALUES LESS THAN (6)
    PARTITION pm6_ix VALUES LESS THAN (7)
    PARTITION pm7_ix VALUES LESS THAN (8)
    PARTITION pm8_ix VALUES LESS THAN (9)
    PARTITION pm9_ix VALUES LESS THAN (10)
    PARTITION pm10_ix VALUES LESS THAN (11)
    PARTITION pm11_ix VALUES LESS THAN (12)
    PARTITION pm12_ix VALUES LESS THAN (MAXVALUE));
```

Note: If your enterprise has or will have databases using different character sets, use caution when partitioning on character columns, because the sort sequence of characters is not identical in all character sets. For more information, see *Oracle9i Database Globalization Support Guide*.

Creating Hash-Partitioned Tables

The `PARTITION BY HASH` clause of the `CREATE TABLE` statement identifies that the table is to be hash-partitioned. The `PARTITIONS` clause can then be used to specify the number of partitions to create, and optionally, the tablespaces to store them in. Alternatively, you can use `PARTITION` clauses to name the individual partitions and their tablespaces.

The only attribute you can specify for hash partitions is `TABLESPACE`. All of the hash partitions of a table must share the same segment attributes (except `TABLESPACE`), which are inherited from the table level.

The following examples illustrate two methods of creating a hash-partitioned table named `dept`. In the first example the number of partitions is specified, but system

generated names are assigned to them and they are stored in the default tablespace of the table.

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
    PARTITION BY HASH(deptno) PARTITIONS 16;
```

In this second example, names of individual partitions, and tablespaces in which they are to reside, are specified. The initial extent size for each hash partition (segment) is also explicitly stated at the table level, and all partitions inherit this attribute.

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
    STORAGE (INITIAL 10K)
    PARTITION BY HASH(deptno)
    (PARTITION p1 TABLESPACE ts1, PARTITION p2 TABLESPACE ts2,
     PARTITION p3 TABLESPACE ts1, PARTITION p4 TABLESPACE ts3);
```

If you create a local index for the above table, Oracle constructs the index so that it is equipartitioned with the underlying table. Oracle also ensures that the index is maintained automatically when maintenance operations are performed on the underlying table. The following is an example of creating a local index on the table dept:

```
CREATE INDEX loc_dept_ix ON dept(deptno) LOCAL;
```

You can optionally name the hash partitions and tablespaces into which the local index partitions are to be stored, but if you do not do so, Oracle uses the name of the corresponding base partition as the index partition name, and stores the index partition in the same tablespace as the table partition.

Creating List-Partitioned Tables

The semantics for creating list partitions are very similar to those for creating range partitions. However, to create list partitions, you specify a `PARTITION BY LIST` clause in the `CREATE TABLE` statement, and the `PARTITION` clauses specify lists of literal values, which are the discrete values of the partitioning columns that qualify rows to be included in the partition. For list partitioning, the partitioning key can only be a single column name from the table.

Available only with list partitioning, you can use the keyword `DEFAULT` to describe the value list for a partition. This identifies a partition that will accommodate rows that do not map into any of the other partitions.

Like for range partitions, optional subclauses of a `PARTITION` clause can specify physical and other attributes specific to a partition's segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

The following example creates table `sales_by_region` and partitions it using the list method. The first two `PARTITION` clauses specify physical attributes, which override the table-level defaults. The remaining `PARTITION` clauses do not specify attributes and those partitions inherit their physical attributes from table-level defaults. A default partition is specified.

```
CREATE TABLE sales_by_region (item# INTEGER, qty INTEGER,
                             store_name VARCHAR(30), state_code VARCHAR(2),
                             sale_date DATE)
    STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5
    PARTITION BY LIST (state_code)
    (
    PARTITION region_east
        VALUES ('MA', 'NY', 'CT', 'NH', 'ME', 'MD', 'VA', 'PA', 'NJ')
        STORAGE (INITIAL 20K NEXT 40K PCTINCREASE 50)
        TABLESPACE tbs8,
    PARTITION region_west
        VALUES ('CA', 'AZ', 'NM', 'OR', 'WA', 'UT', 'NV', 'CO')
        PCTFREE 25 NOLOGGING,
    PARTITION region_south
        VALUES ('TX', 'KY', 'TN', 'LA', 'MS', 'AR', 'AL', 'GA'),
    PARTITION region_central
        VALUES ('OH', 'ND', 'SD', 'MO', 'IL', 'MI', 'IA'),
    PARTITION region_null
        VALUES (NULL),
    PARTITION region_unknown
        VALUES (DEFAULT)
    );
```

Creating Composite Range-Hash Partitioned Tables

To create a range-hash partitioned table, you start by using the `PARTITION BY RANGE` clause of a `CREATE TABLE` statement. Next, you specify a `SUBPARTITION BY HASH` clause that follows similar syntax and rules as the `PARTITION BY HASH` clause. The individual `PARTITION` and `SUBPARTITION` or `SUBPARTITIONS` clauses, and optionally a `SUBPARTITION TEMPLATE` clause, follow.

Attributes specified for a range partition apply to all subpartitions of that partition. You can specify different attributes for each range partition, and you can specify a `STORE IN` clause at the partition level if the list of tablespaces across which that

partition's subpartitions should be spread is different from those of other partitions. All of this is illustrated in the following example.

```
CREATE TABLE emp (deptno NUMBER, empname VARCHAR(32), grade NUMBER)
  PARTITION BY RANGE(deptno) SUBPARTITION BY HASH(empname)
    SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
(PARTITION p1 VALUES LESS THAN (1000) PCTFREE 40,
 PARTITION p2 VALUES LESS THAN (2000)
  STORE IN (ts2, ts4, ts6, ts8),
 PARTITION p3 VALUES LESS THAN (MAXVALUE)
  (SUBPARTITION p3_s1 TABLESPACE ts4,
   SUBPARTITION p3_s2 TABLESPACE ts5));
```

To learn how using a subpartition template can simplify the specification of a composite partitioned table, see ["Using Subpartition Templates to Describe Composite Partitioned Tables"](#) on page 17-17.

The following statement is an example of creating a local index on the `emp` table where the index segments are spread across tablespaces `ts7`, `ts8`, and `ts9`.

```
CREATE INDEX emp_ix ON emp(deptno)
  LOCAL STORE IN (ts7, ts8, ts9);
```

This local index is equipartitioned with the base table as follows:

- It consists of as many partitions as the base table.
- Each index partition consists of as many subpartitions as the corresponding base table partition.
- Index entries for rows in a given subpartition of the base table are stored in the corresponding subpartition of the index.

Creating Composite Range-List Partitioned Tables

The concept of range-list partitioning is similar to that of the other composite partitioning method, range-hash, but this time you specify that the subpartitions are to be list rather than hash. Specifically, after the `CREATE TABLE . . . PARTITION BY RANGE` clause, you include a `SUBPARTITION BY LIST` clause that follows similar syntax and rules as the `PARTITION BY LIST` clause. The individual `PARTITION` and `SUBPARTITION` clauses, and optionally a `SUBPARTITION TEMPLATE` clause, follow.

The range partitions of the composite partitioned table are described as for noncomposite range partitioned tables. This allows that optional subclauses of a `PARTITION` clause can specify physical and other attributes, including tablespace,

specific to a partition's segment. If not overridden at the partition level, partitions inherit the attributes of their underlying table.

The list subpartition's descriptions, in the `SUBPARTITION` clauses, are described as for noncomposite list partitions, except the only physical attribute that can be specified is a tablespace (optional). Subpartitions inherit all other physical attributes from the partition description.

The following example of creates a table that specifies a tablespace at the partition and subpartition levels. The number of subpartitions within each partition varies, and default subpartitions are specified.

```
CREATE TABLE sample_regional_sales
  (deptno number, item_no varchar2(20),
   txn_date date, txn_amount number, state varchar2(2))
PARTITION BY RANGE (txn_date)
SUBPARTITION BY LIST (state)
  (PARTITION q1_1999 VALUES LESS THAN (TO_DATE('1-APR-1999','DD-MON-YYYY'))
   TABLESPACE tbs_1
   (SUBPARTITION q1_1999_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q1_1999_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q1_1999_northeast VALUES ('NY', 'VM', 'NJ'),
    SUBPARTITION q1_1999_southeast VALUES ('FL', 'GA'),
    SUBPARTITION q1_others VALUES (DEFAULT) TABLESPACE tbs_4
   ),
   PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('1-JUL-1999','DD-MON-YYYY'))
   TABLESPACE tbs_2
   (SUBPARTITION q2_1999_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q2_1999_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q2_1999_northeast VALUES ('NY', 'VM', 'NJ'),
    SUBPARTITION q2_1999_southeast VALUES ('FL', 'GA'),
    SUBPARTITION q2_1999_northcentral VALUES ('SD', 'WI'),
    SUBPARTITION q2_1999_southcentral VALUES ('OK', 'TX')
   ),
   PARTITION q3_1999 VALUES LESS THAN (TO_DATE('1-OCT-1999','DD-MON-YYYY'))
   TABLESPACE tbs_3
   (SUBPARTITION q3_1999_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q3_1999_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q3_others VALUES (DEFAULT) TABLESPACE tbs_4
   ),
   PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000','DD-MON-YYYY'))
   TABLESPACE tbs_4
  );
```

This example results in the following subpartition descriptions:

- All subpartitions inherit their physical attributes, other than tablespace, from tablespace level defaults. This is because the only physical attribute that has been specified for partitions or subpartitions is tablespace. There are no table level physical attributes specified, thus tablespace level defaults are inherited at all levels.
- The first 4 subpartitions of partition `q1_1999` are all contained in `tbs_1`, except for the subpartition `q1_others`, which is stored in `tbs_4` and contains all rows that do not map to any of the other partitions.
- The 6 subpartitions of partition `q2_1999` are all stored in `tbs_2`.
- The first 2 subpartitions of partition `q3_1999` are all contained in `tbs_3`, except for the subpartition `q3_others`, which is stored in `tbs_4` and contains all rows that do not map to any of the other partitions.
- There is no subpartition description for partition `q4_1999`. This results in one default subpartition being created and stored in `tbs_4`. The subpartition's name is system generated in the form `SYS_SUBPn`.

To learn how using a subpartition template can simplify the specification of a composite partitioned table, see "[Using Subpartition Templates to Describe Composite Partitioned Tables](#)".

Using Subpartition Templates to Describe Composite Partitioned Tables

You can create subpartitions in a composite partitioned table using a subpartition template. A subpartition template simplifies the specification of subpartitions by not requiring that a subpartition descriptor be specified for every partition in the table. Instead, you describe subpartitions only once in a template, then apply that subpartition template to every partition in the table.

The subpartition template is used whenever a subpartition descriptor is not specified for a partition. If a subpartition descriptor is specified, then it is used instead of the subpartition template for that partition. If no subpartition template is specified, and no subpartition descriptor is supplied for a partition, then a single default subpartition is created.

Specifying a Subpartition Template for a Range-Hash Partitioned Table

In the case of range-hash partitioned tables, the subpartition template can describe the subpartitions in detail, or it can specify just the number of hash subpartitions.

The following example creates a range-hash partitioned table using a subpartition template:

```
CREATE TABLE emp_sub_template (deptno NUMBER, empname VARCHAR(32), grade NUMBER)
  PARTITION BY RANGE(deptno) SUBPARTITION BY HASH(empname)
  SUBPARTITION TEMPLATE
    (SUBPARTITION a TABLESPACE ts1,
     SUBPARTITION b TABLESPACE ts2,
     SUBPARTITION c TABLESPACE ts3,
     SUBPARTITION d TABLESPACE ts4
    )
  (PARTITION p1 VALUES LESS THAN (1000),
   PARTITION p2 VALUES LESS THAN (2000),
   PARTITION p3 VALUES LESS THAN (MAXVALUE)
  );
```

This example produces the following table description:

- Every partition has four subpartitions as described in the subpartition template.
- Each subpartition has a tablespace specified. It is required that if a tablespace is specified for one subpartition in a subpartition template, then one must be specified for all.
- The names of the subpartitions are generated by concatenating the partition name with the subpartition name in the form:

partition name_subpartition name

The following query displays the subpartition names and tablespaces:

```
SQL> SELECT TABLESPACE_NAME, PARTITION_NAME, SUBPARTITION_NAME
  2 FROM DBA_TAB_SUBPARTITIONS WHERE TABLE_NAME='EMP_SUB_TEMPLATE'
  3 ORDER BY TABLESPACE_NAME;
```

TABLESPACE_NAME	PARTITION_NAME	SUBPARTITION_NAME
TS1	P1	P1_A
TS1	P2	P2_A
TS1	P3	P3_A
TS2	P1	P1_B
TS2	P2	P2_B
TS2	P3	P3_B
TS3	P1	P1_C
TS3	P2	P2_C
TS3	P3	P3_C
TS4	P1	P1_D
TS4	P2	P2_D
TS4	P3	P3_D

12 rows selected.

Specifying a Subpartition Template for a Range-List Partitioned Table

The following example, for a range-list partitioned table, illustrates how using a subpartition template can help you stripe data across tablespaces. In this example a table is created where the table subpartitions are vertically striped, meaning that subpartition *n* from every partition is in the same tablespace.

```
CREATE TABLE stripe_regional_sales
    ( deptno number, item_no varchar2(20),
      txn_date date, txn_amount number, state varchar2(2))
PARTITION BY RANGE (txn_date)
SUBPARTITION BY LIST (state)
SUBPARTITION TEMPLATE
    (SUBPARTITION northwest VALUES ('OR', 'WA') TABLESPACE tbs_1,
     SUBPARTITION southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE tbs_2,
     SUBPARTITION northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE tbs_3,
     SUBPARTITION southeast VALUES ('FL', 'GA') TABLESPACE tbs_4,
     SUBPARTITION midwest VALUES ('SD', 'WI') TABLESPACE tbs_5,
     SUBPARTITION south VALUES ('AL', 'AK') TABLESPACE tbs_6,
     SUBPARTITION others VALUES (DEFAULT ) TABLESPACE tbs_7
    )
(PARTITION q1_1999 VALUES LESS THAN ( TO_DATE('01-APR-1999', 'DD-MON-YYYY')),
 PARTITION q2_1999 VALUES LESS THAN ( TO_DATE('01-JUL-1999', 'DD-MON-YYYY')),
 PARTITION q3_1999 VALUES LESS THAN ( TO_DATE('01-OCT-1999', 'DD-MON-YYYY')),
 PARTITION q4_1999 VALUES LESS THAN ( TO_DATE('1-JAN-2000', 'DD-MON-YYYY'))
);
```

If you specified the tablespaces at the partition level (for example, `tbs_1` for partition `q1_1999`, `tbs_2` for partition `q2_1999`, `tbs_3` for partition `q3_1999`, and `tbs_4` for partition `q4_1999`) and not in the subpartition template, then the table would be horizontally striped. All subpartitions would be in the tablespace of the owning partition.

Creating Partitioned Index-Organized Tables

For index-organized tables, you can use the range or hash partitioning method. However, only range partitioned index-organized tables can contain columns with LOBs. The semantics for creating range or hash-partitioned index-organized tables is similar to that for regular tables with these differences:

- When you create the table you specify the `ORGANIZATION INDEX` clause, and `INCLUDING` and `OVERFLOW` clauses as necessary.

- The `PARTITION` or `PARTITIONS` clauses can have `OVERFLOW` subclauses that allow you to specify attributes of the overflow segments at the partition level.

Specifying an `OVERFLOW` clause results in the overflow data segments themselves being equi-partitioned with the primary key index segments. Thus, for partitioned index-organized tables with overflow, each partition has an index segment and an overflow data segment.

For index-organized tables, the set of partitioning columns must be a subset of the primary key columns. Since rows of an index-organized table are stored in the primary key index for the table, the partitioning criterion has an effect on the availability. By choosing the partition key to be a subset of the primary key, an insert operation only needs to verify uniqueness of the primary key in a single partition, thereby maintaining partition independence.

Support for secondary indexes on index-organized tables is similar to the support for regular tables, however, certain maintenance operations do not mark global indexes `UNUSABLE`, as is the case for regular tables.

See Also:

- ["Managing Index-Organized Tables"](#) on page 15-24
- ["Maintaining Partitioned Tables"](#) on page 17-22
- *Oracle9i Application Developer's Guide - Fundamentals* and *Oracle9i Database Concepts* for more information about index-organized tables

Creating Range-Partitioned Index-Organized Tables

You can partition index-organized tables, and their secondary indexes, by the range method. In the following example, a range-partitioned index-organized table `sales` is created. The `INCLUDING` clause specifies all columns after `week_no` are stored in an overflow segment. There is one overflow segment for each partition, all stored in the same tablespace (`overflow_here`). Optionally, `OVERFLOW TABLESPACE` could be specified at the individual partition level, in which case some or all of the overflow segments could have separate `TABLESPACE` attributes.

```
CREATE TABLE sales(acct_no NUMBER(5),
                   acct_name CHAR(30),
                   amount_of_sale NUMBER(6),
                   week_no INTEGER,
                   sale_details VARCHAR2(1000),
                   PRIMARY KEY (acct_no, acct_name, week_no))
  ORGANIZATION INDEX
```

```

INCLUDING week_no
OVERFLOW TABLESPACE overflow_here
PARTITION BY RANGE (week_no)
(PARTITION VALUES LESS THAN (5)
TABLESPACE ts1,
PARTITION VALUES LESS THAN (9)
TABLESPACE ts2 OVERFLOW TABLESPACE overflow_ts2,
...
PARTITION VALUES LESS THAN (MAXVALUE)
TABLESPACE ts13);

```

Creating Hash-Partitioned Index-Organized Tables

The other option for partitioning index-organized tables is to use the hash method. In the following example the index-organized table, `sales`, is partitioned by the hash method.

```

CREATE TABLE sales(acct_no NUMBER(5),
acct_name CHAR(30),
amount_of_sale NUMBER(6),
week_no INTEGER,
sale_details VARCHAR2(1000),
PRIMARY KEY (acct_no, acct_name, week_no))
ORGANIZATION INDEX
INCLUDING week_no
OVERFLOW
PARTITION BY HASH (week_no)
PARTITIONS 16
STORE IN (ts1, ts2, ts3, ts4)
OVERFLOW STORE IN (ts3, ts6, ts9);

```

Note: Since a well designed hash function is supposed to distribute rows in a well balanced fashion among the partitions, updating the primary key column(s) of a row is very likely to move that row to a different partition. Therefore it is recommended that a hash-partitioned index-organized table with a changeable partitioning key be created with the `ROW MOVEMENT ENABLE` clause explicitly specified. The default is that `ROW MOVEMENT ENABLE` is disabled.

Partitioning Restrictions for Multiple Block Sizes

Use caution when creating partitioned objects in a database with tablespaces of multiple block size. The storage of partitioned objects in such tablespaces is subject to some restrictions. Specifically, all partitions of the following entities must reside in tablespaces of the same block size:

- Conventional tables
- Indexes
- Primary key index segments of index-organized tables
- Overflow segments of index-organized tables
- LOB columns stored out of line

Therefore:

- For each conventional table, all partitions of that table must be stored in tablespaces with the same block size.
- For each index-organized table, all primary key index partitions must reside in tablespaces of the same block size, and all overflow partitions of that table must reside in tablespaces of the same block size. However, index partitions and overflow partitions can reside in tablespaces of different block size.
- For each index (global or local), each partition of that index must reside in tablespaces of the same block size. However, partitions of different indexes defined on the same object can reside in tablespaces of different block sizes.
- For each LOB column, each partition of that column must be stored in tablespaces of equal block sizes. However, different LOB columns can be stored in tablespaces of different block sizes.

When you create or alter a partitioned table or index, all tablespaces you *explicitly specify* for the partitions and subpartitions of each entity must be of the same block size. If you *do not explicitly specify* tablespace storage for an entity, the tablespaces Oracle uses by default must be of the same block size. Therefore you must be aware of the default tablespaces at each level of the partitioned object.

Maintaining Partitioned Tables

This section describes how to perform partition and subpartition maintenance operations for both tables and indexes.

Table 17-1 lists maintenance operations that can be performed on table partitions (or subpartitions) and, for each type of partitioning, lists the specific clause of the ALTER TABLE statement that is used to perform that maintenance operation.

Table 17-1 ALTER TABLE Maintenance Operations for Table Partitions (Page 1 of 2)

Maintenance Operation	Range	Hash	List	Composite: Range/Hash	Composite: Range/List
Adding Partitions	ADD PARTITION	ADD PARTITION	ADD PARTITION	ADD PARTITION MODIFY PARTITION...ADD SUBPARTITION	ADD PARTITION MODIFY PARTITION...ADD SUBPARTITION
Coalescing Partitions	n/a	COALESCE PARTITION	n/a	MODIFY PARTITION... COALESCE SUBPARTITION	n/a
Dropping Partitions	DROP PARTITION	n/a	DROP PARTITION	DROP PARTITION	DROP PARTITION DROP SUBPARTITION
Exchanging Partitions	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION	EXCHANGE PARTITION EXCHANGE SUBPARTITION	EXCHANGE PARTITION EXCHANGE SUBPARTITION
Merging Partitions	MERGE PARTITIONS	n/a	MERGE PARTITIONS	MERGE PARTITIONS	MERGE PARTITIONS MERGE SUBPARTITIONS
Modifying Default Attributes	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
Modifying Real Attributes of Partitions	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION	MODIFY PARTITION MODIFY SUBPARTITION
Modifying List Partitions: Adding Values	n/a	n/a	MODIFY PARTITION... ADD VALUES	n/a	MODIFY SUBPARTITION... ADD VALUES
Modifying List Partitions: Dropping Values	n/a	n/a	MODIFY PARTITION... DROP VALUES	n/a	MODIFY SUBPARTITION... DROP VALUES
Modifying a Subpartition Template	n/a	n/a	n/a	SET SUBPARTITION TEMPLATE	SET SUBPARTITION TEMPLATE

Table 17–1 ALTER TABLE Maintenance Operations for Table Partitions (Page 2 of 2)

Maintenance Operation	Range	Hash	List	Composite: Range/Hash	Composite: Range/List
Moving Partitions	MOVE PARTITION	MOVE PARTITION	MOVE PARTITION	MOVE SUBPARTITION	MOVE SUBPARTITION
Renaming Partitions	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION	RENAME PARTITION RENAME SUBPARTITION
Splitting Partitions	SPLIT PARTITION	n/a	SPLIT PARTITION	SPLIT PARTITION	SPLIT PARTITION SPLIT SUBPARTITION
Truncating Partitions	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION TRUNCATE SUBPARTITION	TRUNCATE PARTITION TRUNCATE SUBPARTITION

Note: The first time you introduce a compressed partition into a partitioned table that has bitmap indexes and that currently contains only uncompressed partitions, you must do the following:

- Either drop all existing bitmap indexes and bitmap index partitions, or mark them UNUSABLE.
- Set the compression attribute.
- Rebuild the indexes.

These actions are independent of whether any partitions contain data and of the operation that introduces the compressed partition.

This does not apply partitioned tables with B-tree indexes.

For more information, see the *Oracle9i Data Warehousing Guide*.

Table 17–2 lists maintenance operations that can be performed on index partitions, and indicates on which type of index (global or local) they can be performed. The ALTER INDEX clause used for the maintenance operation is shown.

Global indexes do not reflect the structure of the underlying table, and if partitioned, they can only be partitioned by range. Range-partitioned indexes share some, but not all, of the partition maintenance operations that can be performed on range-partitioned tables.

Because local indexes reflect the underlying structure of the table, partitioning is maintained automatically when table partitions and subpartitions are affected by maintenance activity. Therefore, partition maintenance on local indexes is less necessary and there are fewer options.

Table 17–2 ALTER INDEX Maintenance Operations for Index Partitions

Maintenance Operation	Type of Index	Type of Index Partitioning		
		Range	Hash and List	Composite
Dropping Index Partitions	Global	DROP PARTITION	-	-
	Local	n/a	n/a	n/a
Modifying Default Attributes of Index Partitions	Global	MODIFY DEFAULT ATTRIBUTES	-	-
	Local	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
Modifying Real Attributes of Index Partitions	Global	MODIFY PARTITION	-	-
	Local	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION
Rebuilding Index Partitions	Global	REBUILD PARTITION	-	-
	Local	REBUILD PARTITION	REBUILD PARTITION	REBUILD SUBPARTITION
Renaming Index Partitions	Global	RENAME PARTITION	-	-
	Local	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION
Splitting Index Partitions	Global	SPLIT PARTITION	-	-
	Local	n/a	n/a	n/a

Note: The following sections discuss maintenance operations on partitioned tables. Where the usability of indexes or index partitions affected by the maintenance operation is discussed, consider the following:

- Only indexes and index partitions that are *not* empty are candidates for being marked `UNUSABLE`. If they are empty, the `USABLE/UNUSABLE` status is left unchanged.
 - Only indexes or index partitions with `USABLE` status are updated by subsequent DML.
-
-

Updating Global Indexes Automatically

Before discussing the individual maintenance operations for partitioned tables and indexes, it is important to discuss the effects of the `UPDATE GLOBAL INDEXES` clause that can be specified in the `ALTER TABLE` statement.

By default, many table maintenance operations on partitioned tables invalidate (mark `UNUSABLE`) global indexes. You must then rebuild the entire global index or, if partitioned, all of its partitions. Oracle enables you to override this default behavior if you specify `UPDATE GLOBAL INDEXES` in your `ALTER TABLE` statement for the maintenance operation. Specifying this clause tells Oracle to update the global index at the time it executes the maintenance operation DDL statement. This provides the following benefits:

- The global index is updated in conjunction with the base table operation. You are not required to later and independently rebuild the global index.
- There is higher availability for global indexes, since they do not get marked `UNUSABLE`. The index remains available even while the partition DDL is executing and it can be used to access other partitions in the table.
- You avoid having to look up the names of all invalid global indexes used for rebuilding them.

But also consider the following performance implications when you specify `UPDATE GLOBAL INDEXES`:

- The partition DDL statement takes longer to execute since indexes which were previously marked `UNUSABLE` are updated. However, this must be compared against the time it takes to execute DDL without updating indexes, and then rebuilding all indexes. A rule of thumb is that it is faster to update indexes if the size of the partition is less than 5% of the size of the table.

- The `DROP`, `TRUNCATE`, and `EXCHANGE` operations are no longer fast operations. Again, one must compare the time it takes to do the DDL and then rebuild all global indexes.
- Updates to the index are logged, and redo and undo records are generated. If the entire index is being rebuilt, it can optionally be done `NOLOGGING`.
- Rebuilding the entire index creates a more efficient index, since it is more compact with space better utilized. Further rebuilding the index allows you change storage options.

Note: The `UPDATE GLOBAL INDEXES` clause is not supported for partitioned index-organized tables.

The following operations support the `UPDATE GLOBAL INDEXES` clause:

- `ADD PARTITION | SUBPARTITION (hash only)`
- `COALESCE PARTITION | SUBPARTITION`
- `DROP PARTITION`
- `EXCHANGE PARTITION | SUBPARTITIO`
- `MERGE PARTITION`
- `MOVE PARTITION | SUBPARTITION`
- `SPLIT PARTITION`
- `TRUNCATE PARTITION | SUBPARTITION`

Adding Partitions

This section describes how to add new partitions to a partitioned table and explains why partitions cannot be specifically added to global partitioned or local indexes.

Adding a Partition to a Range-Partitioned Table

Use the `ALTER TABLE ... ADD PARTITION` statement to add a new partition to the "high" end (the point after the last existing partition). To add a partition at the beginning or in the middle of a table, use the `SPLIT PARTITION` clause.

For example, consider the table, `sales`, which contains data for the current month in addition to the previous 12 months. On January 1, 1999, you add a partition for January, which is stored in tablespace `tsx`.

```
ALTER TABLE sales
  ADD PARTITION jan96 VALUES LESS THAN ( '01-FEB-1999' )
  TABLESPACE tsx;
```

Local and global indexes associated with the range-partitioned table remain usable.

Adding a Partition to a Hash-Partitioned Table

When you add a partition to a hash-partitioned table, Oracle populates the new partition with rows rehashed from an existing partition (selected by Oracle) as determined by the hash function.

The following statements show two ways of adding a hash partition to table `scubagear`. Choosing the first statement adds a new hash partition whose partition name is system generated, and which is placed in the table's default tablespace. The second statement also adds a new hash partition, but that partition is explicitly named `p_named` and is created in tablespace `gear5`.

```
ALTER TABLE scubagear ADD PARTITION;
```

```
ALTER TABLE scubagear
  ADD PARTITION p_named TABLESPACE gear5;
```

Indexes may be marked `UNUSABLE` as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	<ul style="list-style-type: none"> ■ The local indexes for the new partition, and for the existing partition from which rows were redistributed, are marked <code>UNUSABLE</code> and must be rebuilt. ■ Unless you specify <code>UPDATE GLOBAL INDEXES</code>, all global indexes, or all partitions of partitioned global indexes, are marked <code>UNUSABLE</code> and must be rebuilt.
Index-organized	<ul style="list-style-type: none"> ■ For local indexes, the behavior is the same as for heap tables. ■ All global indexes remain usable.

Adding a Partition to a List-Partitioned Table

The following statement illustrates adding a new partition to a list-partitioned table. In this example physical attributes and `NOLOGGING` are specified for the partition being added.

```
ALTER TABLE q1_sales_by_region
```

```
ADD PARTITION q1_nonmainland VALUES ('HI', 'PR')
STORAGE (INITIAL 20K NEXT 20K) TABLESPACE tbs_3
NOLOGGING;
```

Any value in the set of literal values that describe the partition being added must not exist in any of the other partitions of the table.

You cannot add a partition to a list-partitioned table that has a default partition, but you can split the default partition. By doing so, you effectively create a new partition defined by the values that you specify, and a second partition that remains the default partition.

Local and global indexes associated with the list-partitioned table remain usable.

Adding Partitions to a Range-Hash Partitioned Table

Partitions can be added at both the range partition level and the hash subpartition level.

Adding a Partition to a Range-Hash Partitioned Table Adding a new range partition to a range-hash partitioned table is as described previously in ["Adding a Partition to a Range-Partitioned Table"](#). However, you can specify a `SUBPARTITIONS` clause that allows you to add a specified number of subpartitions, or a `SUBPARTITION` clause for naming specific subpartitions. If no `SUBPARTITIONS` or `SUBPARTITION` clause is specified, the partition inherits table level defaults for subpartitions.

This example adds a range partition `q1_2000` to table `sales`, which will be populated with data for the first quarter of the year 2000. There are eight subpartitions stored in tablespace `tbs5`.

```
ALTER TABLE sales ADD PARTITION q1_2000
VALUES LESS THAN (2000, 04, 01)
SUBPARTITIONS 8 STORE IN tbs5;
```

Adding a Subpartition to a Range-Hash Partitioned Table You use the `MODIFY PARTITION . . . ADD SUBPARTITION` clause of the `ALTER TABLE` statement to add a hash subpartition to a range-hash partitioned table. The newly added subpartition is populated with rows rehashed from other subpartitions of the same partition as determined by the hash function.

In the following example, a new hash subpartition `us_loc5`, stored in tablespace `us1`, is added to range partition `locations_us` in table `diving`.

```
ALTER TABLE diving MODIFY PARTITION locations_us
ADD SUBPARTITION us_locs5 TABLESPACE us1;
```

Local index subpartitions corresponding to the added and rehashed subpartitions must be rebuilt. Unless you specify `UPDATE GLOBAL INDEXES`, all global indexes, or all partitions of partitioned global indexes, are marked `UNUSABLE` and must be rebuilt.

Adding Partitions to a Range-List Partitioned Table

Partitions can be added at both the range partition level and the list subpartition level.

Adding a Partition to a Range-List Partitioned Table Adding a new range partition to a range-list partitioned table is as described previously in ["Adding a Partition to a Range-Partitioned Table"](#). However, you can specify `SUBPARTITION` clauses for naming and providing value lists for the subpartitions. If no `SUBPARTITION` clauses are specified, then the partition inherits the subpartition template. If there is no subpartition template, then a single default subpartition is created.

The following statement adds a new partition to the `quarterly_regional_sales` table that is partitioned by the range-list method. Some new physical attributes are specified for this new partition while table-level defaults are inherited for those that are not specified.

```
ALTER TABLE quarterly_regional_sales
  ADD PARTITION q1_2000 VALUES LESS THAN (TO_DATE('1-APR-2000','DD-MON-YYYY'))
  STORAGE (INITIAL 20K NEXT 20K) TABLESPACE ts3 NOLOGGING
  (
    SUBPARTITION q1_2000_northwest VALUES ('OR', 'WA'),
    SUBPARTITION q1_2000_southwest VALUES ('AZ', 'UT', 'NM'),
    SUBPARTITION q1_2000_northeast VALUES ('NY', 'VM', 'NJ'),
    SUBPARTITION q1_2000_southeast VALUES ('FL', 'GA'),
    SUBPARTITION q1_2000_northcentral VALUES ('SD', 'WI'),
    SUBPARTITION q1_2000_southcentral VALUES ('OK', 'TX')
  );
```

Adding a Subpartition to a Range-List Partitioned Table You use the `MODIFY PARTITION ... ADD SUBPARTITION` clause of the `ALTER TABLE` statement to add a list subpartition to a range-list partitioned table.

The following statement adds a new subpartition to the existing set of subpartitions in range-list partitioned table `quarterly_regional_sales`. The new subpartition is created in tablespace `ts2`.

```
ALTER TABLE quarterly_regional_sales
  MODIFY PARTITION q1_1999
```

```
ADD SUBPARTITION q1_1999_south
VALUES ('AR','MS','AL') tablespace ts2;
```

Adding Index Partitions

You cannot explicitly add a partition to a local index. Instead, a new partition is added to a local index only when you add a partition to the underlying table. Specifically, when there is a local index defined on a table and you issue the `ALTER TABLE` statement to add a partition, a matching partition is also added to the local index. Oracle assigns names and default physical storage attributes to the new index partitions, but you can rename or alter them after the `ADD PARTITION` operation is complete.

You can effectively specify a new tablespace for an index partition in an `ADD PARTITION` operation by first modifying the default attributes for the index. For example, assume that a local index, `q1_sales_by_region_locix`, was created for list partitioned table `q1_sales_by_region`. If before adding the new partition `q1_nonmainland`, as shown in ["Adding a Partition to a List-Partitioned Table"](#) on page 17-28, you had issued the following statement, then the corresponding index partition would be created in tablespace `tbs_4`.

```
ALTER INDEX q1_sales_by_region_locix
MODIFY DEFAULT ATTRIBUTES TABLESPACE tbs_4;
```

Otherwise, it would be necessary for you to use the following statement to move the index partition to `tbs_4` after adding it:

```
ALTER INDEX q1_sales_by_region_locix
REBUILD PARTITION q1_nonmainland TABLESPACE tbs_4;
```

You cannot add a partition to a global index because the highest partition always has a partition bound of `MAXVALUE`. If you want to add a new highest partition, use the `ALTER INDEX . . . SPLIT PARTITION` statement.

Coalescing Partitions

Coalescing partitions is a way of reducing the number of partitions in a hash-partitioned table, or the number of subpartitions in a range-hash partitioned table. When a hash partition is coalesced, its contents are redistributed into one or more remaining partitions determined by the hash function. The specific partition that is coalesced is selected by Oracle, and is dropped after its contents have been redistributed.

Indexes may be marked `UNUSABLE` as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	<ul style="list-style-type: none"> ■ Any local index partition corresponding to the selected partition is also dropped. Local index partitions corresponding to the one or more absorbing partitions are marked <code>UNUSABLE</code> and must be rebuilt. ■ Unless you specify <code>UPDATE GLOBAL INDEXES</code>, all global indexes, or all partitions of partitioned global indexes, are marked <code>UNUSABLE</code> and must be rebuilt.
Index-organized	<ul style="list-style-type: none"> ■ Some local indexes are marked <code>UNUSABLE</code> as noted above. ■ All global indexes remain usable.

Coalescing a Partition in a Hash-Partitioned Table

The `ALTER TABLE ... COALESCE PARTITION` statement is used to coalesce a partition in a hash-partitioned table. The following statement reduces by one the number of partitions in a table by coalescing a partition.

```
ALTER TABLE ouu1
    COALESCE PARTITION;
```

Coalescing a Subpartition in a Range-Hash Partitioned Table

The following statement distributes the contents of a subpartition of partition `us_locations` into one or more remaining subpartitions (determined by the hash function) of the same partition. Basically, this operation is the inverse of the `MODIFY PARTITION ... ADD SUBPARTITION` clause discussed in ["Adding a Subpartition to a Range-Hash Partitioned Table"](#) on page 17-29.

```
ALTER TABLE diving MODIFY PARTITION us_locations
    COALESCE SUBPARTITION;
```

Dropping Partitions

You can drop partitions from range, composite, list, or composite range-list partitioned tables. For hash-partitioned tables, or hash subpartitions of range-hash partitioned tables, you must perform a coalesce operation instead.

Dropping a Table Partition

Use one of the following statements to drop a table partition or subpartition:

- `ALTER TABLE ... DROP PARTITION` to drop a table partition

- `ALTER TABLE ... DROP SUBPARTITION` to drop a subpartition of a range-list partitioned table

If you want to preserve the data in the partition, use the `MERGE PARTITION` statement instead of the `DROP PARTITION` statement.

If there are local indexes defined for the table, this statement also drops the matching partition or subpartitions from the local index. All global indexes, or all partitions of partitioned global indexes, are marked `UNUSABLE` unless either of the following are true:

- You specify `UPDATE GLOBAL INDEXES` (cannot be specified for index-organized tables)
- The partition being dropped or its subpartitions are empty

Note: You cannot drop the only partition in a table. Instead, you must drop the table.

The following sections contain some scenarios for dropping table partitions.

Dropping a Partition from a Table that Contains Data and Global Indexes If the partition contains data and one or more global indexes are defined on the table, use one of the following methods to drop the table partition.

Method 1:

Leave the global indexes in place during the `ALTER TABLE ... DROP PARTITION` statement. Afterward, you must rebuild any global indexes (whether partitioned or not) because the index (or index partitions) will have been marked `UNUSABLE`. The following statements provide an example of dropping partition `dec98` from the `sales` table, then rebuilding its global nonpartitioned index.

```
ALTER TABLE sales DROP PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

If index `sales_area_ix` were a range-partitioned global index, then all partitions of the index would require rebuilding. Further, it is not possible to rebuild all partitions of an index in one statement. You must write a separate `REBUILD` statement for each partition in the index. The following statements rebuild the index partitions `jan99_ix`, `feb99_ix`, `mar99_ix`, ..., `dec99_ix`.

```
ALTER INDEX sales_area_ix REBUILD PARTITION jan99_ix;
```

```
ALTER INDEX sales_area_ix REBUILD PARTITION feb99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION mar99_ix;
...
ALTER INDEX sales_area_ix REBUILD PARTITION nov99_ix;
```

This method is most appropriate for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2:

Issue the `DELETE` statement to delete all rows from the partition before you issue the `ALTER TABLE ... DROP PARTITION` statement. The `DELETE` statement updates the global indexes, and also fires triggers and generates redo and undo logs.

For example, to drop the first partition, which has a partition bound of 10000, issue the following statements:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec98;
```

This method is most appropriate for small tables, or for large tables when the partition being dropped contains a small percentage of the total data in the table.

Method 3:

Specify `UPDATE GLOBAL INDEXES` in the `ALTER TABLE` statement. This causes the global index to be updated at the time the partition is dropped.

```
ALTER TABLE sales DROP PARTITION dec98
    UPDATE GLOBAL INDEXES;
```

Dropping a Partition Containing Data and Referential Integrity Constraints If a partition contains data and the table has referential integrity constraints, choose either of the following methods to drop the table partition. This table has a local index only, so it is not necessary to rebuild any indexes.

Method 1:

Disable the integrity constraints, issue the `ALTER TABLE ... DROP PARTITION` statement, then enable the integrity constraints:

```
ALTER TABLE sales
    DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales DROP PARTITION dec98;
ALTER TABLE sales
```

```
ENABLE CONSTRAINT dname_sales1;
```

This method is most appropriate for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2:

Issue the `DELETE` statement to delete all rows from the partition before you issue the `ALTER TABLE ... DROP PARTITION` statement. The `DELETE` statement enforces referential integrity constraints, and also fires triggers and generates redo and undo log.

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

This method is most appropriate for small tables or for large tables when the partition being dropped contains a small percentage of the total data in the table.

Dropping Index Partitions

You cannot explicitly drop a partition of a local index. Instead, local index partitions are dropped only when you drop a partition from the underlying table.

If a global index partition is empty, you can explicitly drop it by issuing the `ALTER INDEX ... DROP PARTITION` statement. But, if a global index partition contains data, dropping the partition causes the next highest partition to be marked `UNUSABLE`. For example, you would like to drop the index partition P1, and P2 is the next highest partition. You must issue the following statements:

```
ALTER INDEX npr DROP PARTITION P1;
ALTER INDEX npr REBUILD PARTITION P2;
```

Note: You cannot drop the highest partition in a global index.

Exchanging Partitions

You can convert a partition (or subpartition) into a nonpartitioned table, and a nonpartitioned table into a partition (or subpartition) of a partitioned table by exchanging their data segments. You can also convert a hash-partitioned table into a partition of a range-hash partitioned table, or convert the partition of the range-hash partitioned table into a hash-partitioned table. Similarly, you can convert a list-partitioned table into a partition of a range-list partitioned table, or convert the partition of the range-list partitioned table into a list-partitioned table.

Exchanging table partitions is most useful when you have an application using nonpartitioned tables that you want to convert to partitions of a partitioned table. For example, you could already have partition views that you want to migrate into partitioned tables. Exchanging partitions also facilitates high-speed data loading when used with transportable tablespaces.

When you exchange partitions, logging attributes are preserved. You can optionally specify if local indexes are also to be exchanged (`INCLUDING INDEXES` clause), and if rows are to be validated for proper mapping (`WITH VALIDATION` clause).

Note: When you specify `WITHOUT VALIDATION` for the exchange partition operation, this is normally a fast operation because it involves only data dictionary updates. However, if the table or partitioned table involved in the exchange operation has a primary key or unique constraint enabled, then the exchange operation will be performed as if `WITH VALIDATION` were specified. This is in order to maintain the integrity of the constraints.

To avoid the overhead of this validation activity, issue the following statement for each constraint before doing the exchange partition operation:

```
ALTER TABLE table_name
  DISABLE CONSTRAINT constraint_name KEEP INDEX
```

Then, enable the constraints after the exchange.

Unless you specify `UPDATE GLOBAL INDEXES` (this cannot be specified for index-organized tables), Oracle marks `UNUSABLE` the global indexes, or all global index partitions, on the table whose partition is being exchanged. Any global indexes, or global index partitions, on the table being exchanged are marked `UNUSABLE`.

See Also:

- ["Converting a Partition View into a Partitioned Table"](#) on page 17-62
- ["Using Transportable Tablespaces"](#) on page 11-46 for information about transportable tablespaces

Exchanging a Range, Hash, or List Partition

To exchange a partition of a range, hash, or list-partitioned table with a nonpartitioned table, or the reverse, use the `ALTER TABLE ... EXCHANGE PARTITION` statement. An example of converting a partition into a nonpartitioned table follows. In this example, table `stocks` can be range, hash, or list partitioned.

```
ALTER TABLE stocks
    EXCHANGE PARTITION p3 WITH stock_table_3;
```

Exchanging a Hash-Partitioned Table with a Range-Hash Partition

In this example, you are exchanging a whole hash-partitioned table, with all of its partitions, with a range-hash partitioned table's range partition and all of its hash subpartitions. This is illustrated in the following example.

First, create a hash-partitioned table:

```
CREATE TABLE t1 (i NUMBER, j NUMBER)
    PARTITION BY HASH(i)
    (PARTITION p1, PARTITION p2);
```

Populate the table, then create a range-hash partitioned table as shown:

```
CREATE TABLE t2 (i NUMBER, j NUMBER)
    PARTITION BY RANGE(j)
    SUBPARTITION BY HASH(i)
    (PARTITION p1 VALUES LESS THAN (10)
        SUBPARTITION t2_p1s1
        SUBPARTITION t2_p1s2,
        PARTITION p2 VALUES LESS THAN (20)
        SUBPARTITION t2_p2s1
        SUBPARTITION t2_p2s2));
```

It is important that the partitioning key in table `t1` is the same as the subpartitioning key in table `t2`.

To migrate the data in `t1` to `t2`, and validate the rows, use the following statement:

```
ALTER TABLE t1 EXCHANGE PARTITION p1 WITH TABLE t2
    WITH VALIDATION;
```

Exchanging a Subpartition of a Range-Hash Partitioned Table

Use the `ALTER TABLE ... EXCHANGE SUBPARTITION` statement to convert a hash subpartition of a range-hash partitioned table into a nonpartitioned table, or the reverse. The following example converts the subpartition `q3_1999_s1` of table

`sales` into the nonpartitioned table `q3_1999`. Local index partitions are exchanged with corresponding indexes on `q3_1999`.

```
ALTER TABLE sales EXCHANGE SUBPARTITION q3_1999_s1
    WITH TABLE q3_1999 INCLUDING INDEXES;
```

Exchanging a List-Partitioned Table with a Range-List Partition

The semantics of the `ALTER TABLE ... EXCHANGE PARTITION` statement are the same as described previously in ["Exchanging a Hash-Partitioned Table with a Range-Hash Partition"](#). In the example shown there, the syntax of the `CREATE TABLE` statements would only need to be modified to create a list-partitioned table and a range-list partitioned table, respectively. The actions involved remain the same.

Exchanging a Subpartition of a Range-List Partitioned Table

The semantics of the `ALTER TABLE ... EXCHANGE SUBPARTITION` are the same as described previously in ["Exchanging a Subpartition of a Range-Hash Partitioned Table"](#).

Merging Partitions

Use the `ALTER TABLE ... MERGE PARTITIONS` statement to merge the contents of two partitions into one partition. The two original partitions are dropped, as are any corresponding local indexes.

You cannot use this statement for a hash-partitioned table or for hash subpartitions of a range-hash partitioned table.

Unless the involved partitions or subpartitions are empty, indexes may be marked `UNUSABLE` as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	<ul style="list-style-type: none"> ▪ Oracle marks <code>UNUSABLE</code> all resulting corresponding local index partitions or subpartitions. ▪ Unless you specify <code>UPDATE GLOBAL INDEXES</code>, all global indexes, or all partitions of partitioned global indexes, are marked <code>UNUSABLE</code> and must be rebuilt.
Index-organized	<ul style="list-style-type: none"> ▪ Oracle marks <code>UNUSABLE</code> all resulting corresponding local index partitions or subpartitions. ▪ All global indexes remain usable.

Merging Range Partitions

You are allowed to merge the contents of two adjacent range partitions into one partition. Non adjacent range partitions cannot be merged. The resulting partition inherits the higher upper bound of the two merged partitions.

One reason for merging range partitions is to keep historical data online in larger partitions. For example, you can have daily partitions, with the oldest partition rolled up into weekly partitions, which can then be rolled up into monthly partitions, and so on.

The following scripts create an example of merging range partitions.

First, create a partitioned table and create local indexes.

```
-- Create a Table with four partitions each on its own tablespace
-- Partitioned by range on the data column.
--
CREATE TABLE four_seasons
(
    one DATE,
    two VARCHAR2(60),
    three NUMBER
)
PARTITION BY RANGE ( one )
(
PARTITION quarter_one
    VALUES LESS THAN ( TO_DATE('01-apr-1998','dd-mon-yyyy'))
    TABLESPACE quarter_one,
PARTITION quarter_two
    VALUES LESS THAN ( TO_DATE('01-jul-1998','dd-mon-yyyy'))
    TABLESPACE quarter_two,
PARTITION quarter_three
    VALUES LESS THAN ( TO_DATE('01-oct-1998','dd-mon-yyyy'))
    TABLESPACE quarter_three,
PARTITION quarter_four
    VALUES LESS THAN ( TO_DATE('01-jan-1999','dd-mon-yyyy'))
    TABLESPACE quarter_four
);
--
-- Create local PREFIXED index on Four_Seasons
-- Prefixed because the leftmost columns of the index match the
-- Partition key
--
CREATE INDEX i_four_seasons_1 ON four_seasons ( one,two )
LOCAL (
```

```

PARTITION i_quarter_one TABLESPACE i_quarter_one,
PARTITION i_quarter_two TABLESPACE i_quarter_two,
PARTITION i_quarter_three TABLESPACE i_quarter_three,
PARTITION i_quarter_four TABLESPACE i_quarter_four
);

```

Next, merge partitions.

```

--
-- Merge the first two partitions
--
ALTER TABLE four_seasons
MERGE PARTITIONS quarter_one, quarter_two INTO PARTITION quarter_two;

```

Then, rebuild the local index for the affected partition.

```

-- Rebuild index for quarter_two, which has been marked unusable
-- because it has not had all of the data from Q1 added to it.
-- Rebuilding the index will correct this.
--
ALTER TABLE four_seasons MODIFY PARTITION
quarter_two REBUILD UNUSABLE LOCAL INDEXES;

```

Merging List Partitions

When you merge list partitions, the partitions being merged can be any two partitions. They do not need to be adjacent, as for range partitions, since list partitioning does not assume any order for partitions. The resulting partition consists of all of the data from the original two partitions. If you merge a default list partition with any other partition, the resulting partition will be the default partition.

The statement below merges two partitions of a table partitioned using the list method into a partition that inherits all of its attributes from the table-level default attributes, except for PCTFREE and MAXEXTENTS, which are specified in the statement.

```

ALTER TABLE q1_sales_by_region
MERGE PARTITIONS q1_northcentral, q1_southcentral
INTO PARTITION q1_central
PCTFREE 50 STORAGE(MAXEXTENTS 20);

```

The value lists for the two original partitions were specified as:

```

PARTITION q1_northcentral VALUES ('SD','WI')
PARTITION q1_southcentral VALUES ('OK','TX')

```

The resulting `sales_west` partition's value list comprises the set that represents the union of these two partition value lists, or specifically:

- ('SD','WI','OK','TX')

Merging Range-Hash Partitions

When you merge range-hash partitions, the subpartitions are rehashed into either the number of subpartitions specified in a `SUBPARTITIONS` or `SUBPARTITION` clause, or, if no such clause is included, table-level defaults are used.

Note that the inheritance of properties is different when a range-hash partition is split (discussed in "[Splitting a Range-Hash Partition](#)" on page 17-54), versus when two range-hash partitions are merged. When a partition is split, the new partitions can inherit properties from the original partition since there is only one parent. However, when partitions are merged, properties must be inherited from *table level* defaults because there are two parents and the new partition cannot inherit from either at the expense of the other.

The following example merges two range-hash partitions:

```
ALTER TABLE all_seasons
  MERGE PARTITIONS quarter_1, quarter_2 INTO PARTITION quarter_2
  SUBPARTITIONS 8;
```

Merging Range-List Partitions

Partitions can be merged at the range partition level and subpartitions can be merged at the list subpartition level.

Merging Partitions in a Range-List Partitioned Table Merging range partitions in a range-list partitioned table is as described previously in "[Merging Range Partitions](#)" on page 17-39. However, when you merge two range-list partitions, the resulting new partition inherits the subpartition descriptions from the subpartition template, if one exists. If no subpartition template exists, then a single default subpartition is created for the new partition.

This following statement merges two partitions in the range-list partitioned `stripe_regional_sales` table. A subpartition template exists for the table.

```
ALTER TABLE stripe_regional_sales
  MERGE PARTITIONS q1_1999, q2_1999 INTO PARTITION q1_q2_1999
  PCTFREE 50 STORAGE(MAXEXTENTS 20);
```

Some new physical attributes are specified for this new partition while table-level defaults are inherited for those that are not specified. The new resulting partition `q1_q2_1999` inherits the high-value bound of the partition `q2_1999` and the subpartition value-list descriptions from the subpartition template description of the table.

The data in the resulting partitions consists of data from both the partitions. However, there may be cases where Oracle returns an error. This can occur because data may map out of the new partition when both of the following conditions exist:

- Some of the merged subpartitions' literal values were not included in the subpartition template
- The subpartition template does not contain a default partition definition.

This error condition can be eliminated by always specifying a default partition in the default subpartition template.

Merging Subpartitions in a Range-List Partitioned Table You can merge the contents of any two arbitrary list subpartitions belonging to the *same* range partition. The resulting subpartition's value-list descriptor includes all of the literal values in the value lists for the partitions being merged.

The following statement merges two subpartitions of a table partitioned using range-list method into a new subpartition located in tablespace `ts4`:

```
ALTER TABLE quarterly_regional_sales
MERGE SUBPARTITIONS q1_1999_northwest, q1_1999_southwest
INTO SUBPARTITION q1_1999_west
TABLESPACE ts4;
```

The value lists for the original two partitions were:

- Subpartition `q1_1999_northwest` was described as ('WA' , 'OR')
- Subpartition `q1_1999_southwest` was described as ('AZ' , 'NM' , 'UT')

The resulting subpartition's value list comprises the set that represents the union of these two subpartition value lists:

- Subpartition `q1_1999_west` has a value list described as ('WA' , 'OR' , 'AZ' , 'NM' , 'UT')

The tablespace in which the resulting subpartition is located and the subpartition's attributes are determined by the partition-level default attributes, except for those specified explicitly. If any of the existing subpartition names are being reused, then

the new subpartition inherits the subpartition attributes of the subpartition whose name is being reused.

Modifying Default Attributes

You can modify the default attributes of a table, or for a partition of a composite partitioned table. When you modify default attributes, the new attributes affect only future partitions, or subpartitions, that are created. The default values can still be specifically overridden when creating a new partition or subpartition.

Modifying Default Attributes of a Table

You modify the default attributes that will be inherited for range, list, or hash partitions using the `MODIFY DEFAULT ATTRIBUTES` clause of `ALTER TABLE`. The following example changes the default value of `PCTFREE` in table `emp` for any new partitions that are created.

```
ALTER TABLE emp
    MODIFY DEFAULT ATTRIBUTES PCTFREE 25;
```

For hash-partitioned tables, only the `TABLESPACE` attribute can be modified.

Modifying Default Attributes of a Partition

To modify the default attributes inherited when creating subpartitions, use the `ALTER TABLE ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION`. The following statement modifies the `TABLESPACE` in which future subpartitions of partition `p1` in range-hash partitioned table `emp` will reside.

```
ALTER TABLE emp
    MODIFY DEFAULT ATTRIBUTES FOR PARTITION p1 TABLESPACE ts1;
```

Since all subpartitions of a range-hash partitioned table must share the same attributes, except `TABLESPACE`, it is the only attribute that can be changed.

Modifying Default Attributes of Index Partitions

In similar fashion to table partitions, you can alter the default attributes that will be inherited by partitions of a range-partitioned global index, or local index partitions of partitioned tables. For this you use the `ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES` statement. Use the `ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION` statement if you are altering default attributes to be inherited by subpartitions of a composite partitioned table.

Modifying Real Attributes of Partitions

It is possible to modify attributes of an existing partition of a table or index.

You cannot change the `TABLESPACE` attribute. Use `ALTER TABLESPACE . . . MOVE PARTITION/SUBPARTITION` to move a partition or subpartition to a new tablespace.

Modifying Real Attributes for a Range or List Partition

Use the `ALTER TABLE . . . MODIFY PARTITION` statement to modify existing attributes of a range partition or list partition. You can modify segment attributes (except `TABLESPACE`), or you can allocate and deallocate extents, mark local index partitions `UNUSABLE`, or rebuild local indexes that have been marked `UNUSABLE`.

If this is a range partition of a range-hash partitioned table, note the following:

- If you allocate or deallocate an extent, this action is performed for every subpartition of the specified partition.
- Likewise, changing any other attributes results in corresponding changes to those attributes of all the subpartitions for that partition. The partition level default attributes are changed as well. To avoid changing attributes of existing subpartitions, use the `FOR PARTITION` clause of the `MODIFY DEFAULT ATTRIBUTES` statement.

The following are some examples of modifying the real attributes of a partition.

This example modifies the `MAXEXTENTS` storage attribute for the range partition `sales_q1` of table `sales`:

```
ALTER TABLE sales MODIFY PARTITION sales_q1
    STORAGE (MAXEXTENTS 10);
```

All of the local index subpartitions of partition `ts1` in range-hash partitioned table `scubagear` are marked `UNUSABLE` in the following example:

```
ALTER TABLE scubagear MPDIFY PARTITION ts1 UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes for a Hash Partition

You also use the `ALTER TABLE . . . MODIFY PARTITION` statement to modify attributes of a hash partition. However, since the physical attributes of individual hash partitions must all be the same (except for `TABLESPACE`), you are restricted to:

- Allocating a new extent
- Deallocating an unused extent

- Marking a local index subpartition `UNUSABLE`
- Rebuilding local index subpartitions that are marked `UNUSABLE`

The following example rebuilds any unusable local index partitions associated with hash partition `P1` of table `dept`:

```
ALTER TABLE dept MODIFY PARTITION p1
    REBUILD UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes of a Subpartition

With the `MODIFY SUBPARTITION` clause of `ALTER TABLE` you can perform the same actions as listed previously for partitions, but at the specific composite partitioned table subpartition level. For example:

```
ALTER TABLE emp MODIFY SUBPARTITION p3_s1
    REBUILD UNUSABLE LOCAL INDEXES;
```

Modifying Real Attributes of Index Partitions

The `MODIFY PARTITION` clause of `ALTER INDEX` allows you to modify the real attributes of an index partition or its subpartitions. The rules are very similar to those for table partitions, but unlike the `MODIFY PARTITION` clause for `ALTER TABLE`, there is no subclause to rebuild an unusable index partition, but there is a subclause to coalesce an index partition or its subpartitions. In this context, coalesce means to merge index blocks where possible to free them for reuse.

You can also allocate or deallocate storage for a subpartition of a local index, or mark it `UNUSABLE`, using the `MODIFY SUBPARTITION` clause.

Modifying List Partitions: Adding Values

List partitioning allows you the option of adding literal values from the defining value list.

Adding Values for a List Partition

Use the `MODIFY PARTITION ... ADD VALUES` clause of the `ALTER TABLE` statement to extend the value list of an existing partition. Literal values being added must not have been included in any other partition's value list. The partition value list for any corresponding local index partition is correspondingly extended, and any global index, or global or local index partitions, remain usable.

The following statement adds a new set of state codes ('OK', 'KS') to an existing partition list.

```
ALTER TABLE sales_by_region
  MODIFY PARTITION region_south
    ADD VALUES ('OK', 'KS');
```

The existence of a default partition can have a performance impact when adding values to other partitions. This is because in order to add values to a list partition, Oracle must check that the values being added do not already exist in the default partition. If any of the values do exist in the default partition, an error is raised.

Note: Because a query is executed to check for the existence of rows in the default partition that correspond to the literal values being added, it is advisable to create a local prefixed index on the table. This speeds up the execution of the query and the overall operation.

You cannot add values to a default list partition.

Adding Values for a List Subpartition

This operation is essentially the same as described for ["Modifying List Partitions: Adding Values"](#), however, you use a `MODIFY SUBPARTITION` clause instead of the `MODIFY PARTITION` clause. For example, to extend the range of literal values in the value list for subpartition `q1_1999_southeast` use the following statement:

```
ALTER TABLE quarterly_regional_sales
  MODIFY SUBPARTITION q1_1999_southeast
    ADD VALUES ('KS');
```

Literal values being added must not have been included in any other subpartition's value list within the owning partition. However, they can be duplicates of literal values in the subpartition value lists of other partitions within the table.

Modifying List Partitions: Dropping Values

List partitioning allows you the option of dropping literal values from the defining value list.

Dropping Values from a List Partition

Use the `MODIFY PARTITION . . . DROP VALUES` clause of the `ALTER TABLE` statement to remove literal values from the value list of an existing partition. The statement is always executed with validation, meaning that it checks to see if any

rows exist in the partition that correspond to the set of values being dropped. If any such rows are found then Oracle returns an error message and the operation fails. When necessary, use a `DELETE` statement to delete corresponding rows from the table before attempting to drop values.

Note: You cannot drop all literal values from the value list describing the partition. You must use the `ALTER TABLE ... DROP PARTITION` statement instead.

The partition value list for any corresponding local index partition reflects the new value list, and any global index, or global or local index partitions, remain usable.

The statement below drops a set of state codes ('OK' and 'KS') from an existing partition value list.

```
ALTER TABLE sales_by_region
  MODIFY PARTITION region_south
    DROP VALUES ('OK', 'KS');
```

Note: Because a query is executed to check for the existence of rows in the partition that correspond to the literal values being dropped, it is advisable to create a local prefixed index on the table. This speeds up the execution of the query and the overall operation.

You cannot drop values from a default list partition.

Dropping Values from a List Subpartition

This operation is essentially the same as described for "[Modifying List Partitions: Dropping Values](#)", however, you use a `MODIFY SUBPARTITION` clause instead of the `MODIFY PARTITION` clause. For example, to remove a set of literal values in the value list for subpartition `q1_1999_southeast` use the following statement:

```
ALTER TABLE quarterly_regional_sales
  MODIFY SUBPARTITION q1_1999_southeast
    DROP VALUES ('KS');
```

Modifying a Subpartition Template

You can modify a subpartition template of a composite partitioned table by replacing it with a new subpartition template. Any subsequent operations that use the subpartition template (such as `ADD PARTITION` or `MERGE PARTITIONS`) will now use the new subpartition template. Existing subpartitions remain unchanged.

Use the `ALTER TABLE ... SET SUBPARTITION TEMPLATE` statement to specify a new subpartition template. For example:

```
ALTER TABLE emp_sub_template
  SET SUBPARTITION TEMPLATE
    (SUBPARTITION e, TABLESPACE ts1,
     SUBPARTITION f, TABLESPACE ts2,
     SUBPARTITION g, TABLESPACE ts3,
     SUBPARTITION h, TABLESPACE ts4
    );
```

You can drop a subpartition template by specifying an empty list:

```
ALTER TABLE emp_sub_template
  SET SUBPARTITION TEMPLATE ( );
```

Moving Partitions

Use the `MOVE PARTITION` clause of the `ALTER TABLE` statement to:

- Re-cluster data and reduce fragmentation
- Move a partition to another tablespace
- Modify create-time attributes

Typically, you can change the physical storage attributes of a partition in a single step using an `ALTER TABLE/INDEX ... MODIFY PARTITION` statement. However, there are some physical attributes, such as `TABLESPACE`, that you cannot modify using `MODIFY PARTITION`. In these cases, use the `MOVE PARTITION` clause.

Unless the partition being moved does not contain any data, indexes may be marked `UNUSABLE` according to the following table:

Table Type	Index Behavior
Regular (Heap)	<ul style="list-style-type: none"> ■ The matching partition in each local index is marked <code>UNUSABLE</code>. You must rebuild these index partitions after issuing <code>MOVE PARTITION</code>. ■ Unless you specify <code>UPDATE GLOBAL INDEXES</code>, any global indexes, or all partitions of partitioned global indexes, are marked <code>UNUSABLE</code>.
Index-organized	Any local or global indexes defined for the partition being moved remain usable because they are primary-key based logical rowids. However, the guess information for these rowids becomes incorrect.

Moving Table Partitions

Use the `MOVE PARTITION` clause to move a partition. For example, to move the most active partition to a tablespace that resides on its own disk (in order to balance I/O) and to not log the action, issue the following statement:

```
ALTER TABLE parts MOVE PARTITION depot2
    TABLESPACE ts094 NOLOGGING;
```

This statement always drops the partition's old segment and creates a new segment, even if you do not specify a new tablespace.

Moving Subpartitions

The following statement shows how to move data in a subpartition of a table. In this example, a `PARALLEL` clause has also been specified.

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
    TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Moving Index Partitions

The `ALTER TABLE ... MOVE PARTITION` statement for regular tables, marks all partitions of a global index `UNUSABLE`. You can rebuild the entire index by rebuilding each partition individually using the `ALTER INDEX ... REBUILD PARTITION` statement. You can perform these rebuilds concurrently.

You can also simply drop the index and re-create it.

Rebuilding Index Partitions

Some reasons for rebuilding index partitions include:

- To recover space and improve performance
- To repair a damaged index partition caused by media failure
- To rebuild a local index partition after loading the underlying table partition with Import or SQL*Loader
- To rebuild index partitions that have been marked `UNUSABLE`

The following sections discuss your options for rebuilding index partitions and subpartitions.

Rebuilding Global Index Partitions

You can rebuild global index partitions in two ways:

1. Rebuild each partition by issuing the `ALTER INDEX ... REBUILD PARTITION` statement (you can run the rebuilds concurrently).
2. Drop the entire global index and re-create it.

Note: This second method is more efficient because the table is scanned only once.

For most maintenance operations on partitioned tables with global indexes, you can optionally avoid the need to rebuild the global index by specifying `UPDATE GLOBAL INDEXES` on your DDL statement.

Rebuilding Local Index Partitions

Rebuild local indexes using either `ALTER INDEX` or `ALTER TABLE` as follows:

- `ALTER INDEX ... REBUILD PARTITION/SUBPARTITION`

This statement rebuilds an index partition or subpartition unconditionally.

- `ALTER TABLE ... MODIFY PARTITION/SUBPARTITION ... REBUILD UNUSABLE LOCAL INDEXES`

This statement finds all of the unusable indexes for the given table partition or subpartition and rebuilds them. It only rebuilds an index partition if it has been marked `UNUSABLE`.

Using Alter Index to Rebuild a Partition The `ALTER INDEX ... REBUILD PARTITION` statement rebuilds one partition of an index. It cannot be used on a range-hash partitioned table. When you re-create the index, you can also choose to move the partition to a new tablespace or change attributes.

For range-hash partitioned tables, use `ALTER INDEX ... REBUILD SUBPARTITION` to rebuild a subpartition of an index. You can move the subpartition to another tablespace or specify a parallel clause. The following statement rebuilds a subpartition of a local index on a table and moves the index subpartition to another tablespace.

```
ALTER INDEX scuba
    REBUILD SUBPARTITION bcd_types
    TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

Using Alter Table to Rebuild an Index Partition The `REBUILD UNUSABLE LOCAL INDEXES` clause of `ALTER TABLE ... MODIFY PARTITION` does not allow you to specify any new attributes for the rebuilt index partition. The following example finds and rebuilds any unusable local index partitions for table `scubagear`, partition `p1`.

```
ALTER TABLE scubagear
    MODIFY PARTITION p1 REBUILD UNUSABLE LOCAL INDEXES;
```

There is a corresponding `ALTER TABLE ... MODIFY SUBPARTITION` clause for rebuilding unusable local index subpartitions.

Renaming Partitions

It is possible to rename partitions and subpartitions of both tables and indexes. One reason for renaming a partition might be to assign a meaningful name, as opposed to a default system name that was assigned to the partition in another maintenance operation.

Renaming a Table Partition

Rename a range, hash, or list partition, using the `ALTER TABLE ... RENAME PARTITION` statement. For example:

```
ALTER TABLE scubagear RENAME PARTITION sys_p636 TO tanks;
```

Renaming a Table Subpartition

Likewise, you can assign new names to subpartitions of a table. In this case you would use the `ALTER TABLE ... RENAME SUBPARTITION` syntax.

Renaming Index Partitions

Index partitions and subpartitions can be renamed in similar fashion, but the `ALTER INDEX` syntax is used.

Renaming an Index Partition Use the `ALTER INDEX ... RENAME PARTITION` statement to rename an index partition.

Renaming an Index Subpartition This next statement simply shows how to rename a subpartition that has a system generated name that was a consequence of adding a partition to an underlying table:

```
ALTER INDEX scuba RENAME SUBPARTITION sys_subp3254 TO bcd_types;
```

Splitting Partitions

The `SPLIT PARTITION` clause of the `ALTER TABLE` or `ALTER INDEX` statement is used to redistribute the contents of a partition into two new partitions. Consider doing this when a partition becomes too large and causes backup, recovery, or maintenance operations to take a long time to complete. You can also use the `SPLIT PARTITION` clause to redistribute the I/O load.

This clause cannot be used for hash partitions or subpartitions.

Unless the partition you are splitting does not contain any data, indexes may be marked `UNUSABLE` as explained in the following table:

Table Type	Index Behavior
Regular (Heap)	<ul style="list-style-type: none"> ▪ Oracle marks <code>UNUSABLE</code> the new partitions (there are two) in each local index. ▪ Unless you specify <code>UPDATE GLOBAL INDEXES</code>, any global indexes, or all partitions of partitioned global indexes, are marked <code>UNUSABLE</code> and must be rebuilt.
Index-organized	<ul style="list-style-type: none"> ▪ Oracle marks <code>UNUSABLE</code> the new partitions (there are two) in each local index. ▪ All global indexes remain usable.

Splitting a Partition of a Range-Partitioned Table

You split a range partition using the `ALTER TABLE ... SPLIT PARTITION` statement. You specify a value of the partitioning key column within the range of the partition at which to split the partition. The first of the resulting two new partitions includes all rows in the original partition whose partitioning key column

values map lower than the specified value. The second partition contains all rows whose partitioning key column values map greater than or equal to the specified value.

You can optionally specify new attributes for the two partitions resulting from the split. If there are local indexes defined on the table, this statement also splits the matching partition in each local index.

In the following example `fee_katy` is a partition in the table `vet_cats`, which has a local index, `jaf1`. There is also a global index, `vet` on the table. `vet` contains two partitions, `vet_parta`, and `vet_partb`.

To split the partition `fee_katy`, and rebuild the index partitions, issue the following statements:

```
ALTER TABLE vet_cats SPLIT PARTITION
    fee_katy at (100) INTO ( PARTITION
    fee_katy1 ..., PARTITION fee_katy2 ...);
ALTER INDEX JAF1 REBUILD PARTITION fee_katy1;
ALTER INDEX JAF1 REBUILD PARTITION fee_katy2;
ALTER INDEX VET REBUILD PARTITION vet_parta;
ALTER INDEX VET REBUILD PARTITION vet_partb;
```

Note: If you do not specify new partition names, Oracle assigns names of the form `SYS_Pn`. You can examine the data dictionary to locate the names assigned to the new local index partitions. You may want to rename them. Any attributes you do not specify are inherited from the original partition.

Splitting a Partition of a List-Partitioned Table

You split a list partition using the `ALTER TABLE ... SPLIT PARTITION` statement. The `SPLIT PARTITION` clause allows you to specify a value list of literal values that define a partition into which rows with corresponding partitioning key values are inserted. The remaining rows of the original partition are inserted into a second partition whose value list is the remaining values from the original partition.

You can optionally specify new attributes for the two partitions resulting from the split.

The following statement splits the partition `region_east` into 2 partitions:

```
ALTER TABLE sales_by_region
    SPLIT PARTITION region_east VALUES ('CT', 'VA', 'MD')
```

```

INTO
  ( PARTITION region_east_1
    PCTFREE 25 TABLESPACE tbs2,
    PARTITION region_east_2
    STORAGE (NEXT 2M PCTINCREASE 25))
PARALLEL 5;

```

The literal-value list for the original `region_east` partition was specified as:

```

PARTITION region_east VALUES ('MA', 'NY', 'CT', 'NH', 'ME', 'MD', 'VA', 'PA', 'NJ')

```

The two new partition's are:

- `region_east_1` with a literal-value list of ('CT', 'VA', 'MD')
- `region_east_2` inheriting the remaining literal-value list of ('NY', 'NH', 'ME', 'VA', 'PA', 'NJ')

The individual partitions have new physical attributes specified at the partition level. The operation is executed with parallelism of degree 5.

You can split a default list partition just like you split any other list partition. This is also the only means of adding a partition to list-partitioned table that contains a default partition. When you split the default partition, you create a new partition defined by the values that you specify, and a second partition that remains the default partition.

The following example splits the default partition of `sales_by_region`, thereby creating a new partition.

```

ALTER TABLE sales_by_region
  SPLIT PARTITION region_unknown VALUES ('MT', 'WY', 'ID')
  INTO
  ( PARTITION region_wildwest,
    PARTITION region_unknown);

```

Splitting a Range-Hash Partition

This is the opposite of merging range-hash partitions. When you split range-hash partitions, the new subpartitions are rehashed into either the number of subpartitions specified in a `SUBPARTITIONS` or `SUBPARTITION` clause. Or, if no such clause is included, the new partitions inherit the number of subpartitions (and tablespaces) from the partition being split.

Note that the inheritance of properties is different when a range-hash partition is split, versus when two range-hash partitions are merged. When a partition is split, the new partitions can inherit properties from the original partition since there is

only one parent. However, when partitions are merged, properties must be inherited from *table level* defaults because there are two parents and the new partition cannot inherit from either at the expense of the other.

The following example splits a range-hash partition:

```
ALTER TABLE all_seasons SPLIT PARTITION quarter_1
  AT (TO_DATE('16-dec-1997','dd-mon-yyyy'))
  INTO (PARTITION q1_1997_1 SUBPARTITIONS 4 STORE IN (ts1,ts3),
        PARTITION q1_1997_2);
```

Splitting Partitions in a Range-List Partitioned Table

Partitions can be split at both the range partition level and at the list subpartition level.

Splitting a Range-List Partition Splitting a range partition of a range-list partitioned table is similar to what is described in ["Splitting a Partition of a Range-Partitioned Table"](#) on page 17-52. No subpartition literal value list can be specified for either of the new partitions. The new partitions inherit the subpartition descriptions from the original partition being split.

The following example splits the `q1_1999` partition of the `quarterly_regional_sales` table:

```
ALTER TABLE quarterly_regional_sales SPLIT PARTITION q1_1999
  AT (to_date('15-Feb-1999','dd-mon-yyyy'))
  INTO ( PARTITION q1_1999_jan_feb
        PCTFREE 25 TABLESPACE ts1,
        PARTITION q1_1999_feb_mar
        STORAGE (NEXT 2M PCTINCREASE 25) TABLESPACE ts2)
  PARALLEL 5;
```

This operation splits the partition `q1_1999` into two resulting partitions: `q1_1999_jan_feb` and `q1_1999_feb_mar`. Both partitions inherit their subpartition descriptions from the original partition. The individual partitions have new physical attributes, including tablespaces, specified at the partition level. These new attributes become the default attributes of the new partitions. This operation is run with parallelism of degree 5.

The `ALTER TABLE ... SPLIT PARTITION` statement provides no means of specifically naming subpartitions resulting from the split of a partition in a composite partitioned table. However, for those subpartitions in the parent partition with names of the form "*partition name_subpartition name*", Oracle generates corresponding names in the newly created subpartitions using the new partition

names. All other subpartitions are assigned system generated names of the form `SYS_SUBPn`. System generated names are also assigned for the subpartitions of any partition resulting from the split for which a name is not specified. Unnamed partitions are assigned a system generated partition name of the form `SYS_Pn`.

The following query displays the subpartition names resulting from the previous split partition operation on table `quarterly_regional_sales`. It also reflects the results of other operations performed on this table in preceding sections of this chapter since it's creation in ["When to Use the Composite Range-List Partitioning Method"](#) on page 17-8.

```
SQL> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLESPACE_NAME
       2     FROM DBA_TAB_SUBPARTITIONS WHERE TABLE_NAME='QUARTERLY_REGIONAL_SALES'
       3     ORDER BY PARTITION_NAME;
```

PARTITION_NAME	SUBPARTITION_NAME	TABLESPACE_NAME
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_WEST	TS2
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_NORTHEAST	TS2
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_SOUTHEAST	TS2
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_NORTHCENTRAL	TS2
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_SOUTHCENTRAL	TS2
Q1_1999_FEB_MAR	Q1_1999_FEB_MAR_SOUTH	TS2
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_WEST	TS1
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_NORTHEAST	TS1
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_SOUTHEAST	TS1
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_NORTHCENTRAL	TS1
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_SOUTHCENTRAL	TS1
Q1_1999_JAN_FEB	Q1_1999_JAN_FEB_SOUTH	TS1
Q1_2000	Q1_2000_NORTHWEST	TS3
Q1_2000	Q1_2000_SOUTHWEST	TS3
Q1_2000	Q1_2000_NORTHEAST	TS3
Q1_2000	Q1_2000_SOUTHEAST	TS3
Q1_2000	Q1_2000_NORTHCENTRAL	TS3
Q1_2000	Q1_2000_SOUTHCENTRAL	TS3
Q2_1999	Q2_1999_NORTHWEST	TS4
Q2_1999	Q2_1999_SOUTHWEST	TS4
Q2_1999	Q2_1999_NORTHEAST	TS4
Q2_1999	Q2_1999_SOUTHEAST	TS4
Q2_1999	Q2_1999_NORTHCENTRAL	TS4
Q2_1999	Q2_1999_SOUTHCENTRAL	TS4
Q3_1999	Q3_1999_NORTHWEST	TS4
Q3_1999	Q3_1999_SOUTHWEST	TS4
Q3_1999	Q3_1999_NORTHEAST	TS4
Q3_1999	Q3_1999_SOUTHEAST	TS4

Q3_1999	Q3_1999_NORTHCENTRAL	TS4
Q3_1999	Q3_1999_SOUTHCENTRAL	TS4
Q4_1999	Q4_1999_NORTHWEST	TS4
Q4_1999	Q4_1999_SOUTHWEST	TS4
Q4_1999	Q4_1999_NORTHEAST	TS4
Q4_1999	Q4_1999_SOUTHEAST	TS4
Q4_1999	Q4_1999_NORTHCENTRAL	TS4
Q4_1999	Q4_1999_SOUTHCENTRAL	TS4

36 rows selected.

Splitting a Range-List Subpartition Splitting a list subpartition of a range-list partitioned table is similar to what is described in ["Splitting a Partition of a List-Partitioned Table"](#) on page 17-53, but the syntax is that of `SUBPARTITION` rather than `PARTITION`. For example, the following statement splits a subpartition of the `quarterly_regional_sales` table:

```
ALTER TABLE quarterly_regional_sales SPLIT SUBPARTITION q2_1999_southwest
VALUES ('UT') INTO
  ( SUBPARTITION q2_1999_utah
    TABLESPACE ts2,
    SUBPARTITION q2_1999_southwest
    TABLESPACE ts3
  )
PARALLEL;
```

This operation splits the subpartition `q2_1999_southwest` into two subpartitions:

- `q2_1999_utah` with literal-value list of ('UT')
- `q2_1999_southwest` which inherits the remaining literal-value list of ('AZ' , 'NM')

The individual subpartitions have new physical attributes that are inherited from the subpartition being split.

Splitting Index Partitions

You cannot explicitly split a partition in a local index. A local index partition is split only when you split a partition in the underlying table. However, you can split a global index partition as is done in the following example:

```
ALTER INDEX quon1 SPLIT
PARTITION canada AT ( 100 ) INTO
PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
```

```
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

The index being split can contain index data, and the resulting partitions do not require rebuilding, unless the original partition was previously marked `UNUSABLE`.

Optimizing `SPLIT PARTITION` and `SPLIT SUBPARTITION` Operations

Oracle implements a `SPLIT PARTITION` operation by creating two new partitions and redistributing the rows from the partition being split into the two new partitions. This is an expensive operation because it is necessary to scan all the rows of the partition being split and then insert them one-by-one into the new partitions. Local index partitions corresponding to the new partitions also need to be rebuilt. Further if you do not use the `UPDATE GLOBAL INDEXES` clause, global indexes also require rebuilding.

Sometimes after a split operation, one of the new partitions contains all of the rows from the partition being split, while the other partition contains no rows. This is often the case when splitting the first partition of a table. Oracle can detect such situations and optimize the split operation. This optimization results in a fast split operation that behaves like an add partition operation.

Specifically, Oracle can optimize and speed up `SPLIT PARTITION` operations if two conditions are met:

- One of the two resulting partitions must be empty.
- The non-empty resulting partition must have storage characteristics identical to those of the partition being split. Specifically:
 - If the partition being split is composite, then the storage characteristics of each subpartition in the new non-empty resulting partition must be identical to those of the subpartitions of the partition being split.
 - If the partition being split contains a `LOB` column, then the storage characteristics of each `LOB` (sub)partition in the new non-empty resulting partition must be identical to those of the `LOB` (sub)partitions of the partition being split.

If both of these conditions are met after the split, then all global indexes remain usable, even if you did not specify the `UPDATE GLOBAL INDEXES` clause. Local index (sub)partitions associated with both resulting partitions remain usable if they were usable before the split. Local index (sub)partition(s) corresponding to the non-empty resulting partition will be identical to the local index (sub)partition(s) of the partition that was split.

The same optimization holds for `SPLIT SUBPARTITION` operations.

Truncating Partitions

Use the `ALTER TABLE ... TRUNCATE PARTITION` statement to remove all rows from a table partition. Truncating a partition is similar to dropping a partition, except that the partition is emptied of its data, but not physically dropped.

You cannot truncate an index partition. However, if there are local indexes defined for the table, the `ALTER TABLE TRUNCATE PARTITION` statement truncates the matching partition in each local index. Unless you specify `UPDATE GLOBAL INDEXES` (cannot be specified for index-organized tables), any global indexes, or all partitions of partitioned global indexes, are marked `UNUSABLE` and must be rebuilt.

Truncating a Table Partition

Use the `ALTER TABLE ... TRUNCATE PARTITION` statement to remove all rows from a table partition, with or without reclaiming space.

Truncating Table Partitions Containing Data and Global Indexes If the partition contains data and global indexes, use one of the following methods to truncate the table partition.

Method 1:

Leave the global indexes in place during the `ALTER TABLE TRUNCATE PARTITION` statement. In this example, table `sales` has a global index `sales_area_ix`, which is rebuilt.

```
ALTER TABLE sales TRUNCATE PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

This method is most appropriate for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2:

Issue the `DELETE` statement to delete all rows from the partition before you issue the `ALTER TABLE ... TRUNCATE PARTITION` statement. The `DELETE` statement updates the global indexes, and also fires triggers and generates redo and undo logs.

For example, to truncate the first partition, which has a partition bound of 10000, issue the following statements:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales TRUNCATE PARTITION dec98;
```

This method is most appropriate for small tables, or for large tables when the partition being truncated contains a small percentage of the total data in the table.

Method 3:

Specify `UPDATE GLOBAL INDEXES` in the `ALTER TABLE` statement. This causes the global index to be truncated at the time the partition is truncated.

```
ALTER TABLE sales TRUNCATE PARTITION dec98
    UPDATE GLOBAL INDEXES;
```

Truncating a Partition Containing Data and Referential Integrity Constraints If a partition contains data and has referential integrity constraints, choose either of the following methods to truncate the table partition.

Method 1:

Disable the integrity constraints, issue the `ALTER TABLE ... TRUNCATE PARTITION` statement, then re-enable the integrity constraints:

```
ALTER TABLE sales
    DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales TRUNCATE PARTITION dec94;
ALTER TABLE sales
    ENABLE CONSTRAINT dname_sales1;
```

This method is most appropriate for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2:

Issue the `DELETE` statement to delete all rows from the partition before you issue the `ALTER TABLE ... TRUNCATE PARTITION` statement. The `DELETE` statement enforces referential integrity constraints, and also fires triggers and generates redo and undo log.

Note: You can substantially reduce the amount of logging by setting the `NOLOGGING` attribute (using `ALTER TABLE ... MODIFY PARTITION ... NOLOGGING`) for the partition before deleting all of its rows.

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

This method is most appropriate for small tables, or for large tables when the partition being truncated contains a small percentage of the total data in the table.

Truncating a Subpartition

You use the `ALTER TABLE ... TRUNCATE SUBPARTITION` statement to remove all rows from a subpartition of a composite partitioned table. Corresponding local index subpartitions are also truncated.

The following statement shows how to truncate data in a subpartition of a table. In this example, the space occupied by the deleted rows is made available for use by other schema objects in the tablespace.

```
ALTER TABLE diving
  TRUNCATE SUBPARTITION us_locations
  DROP STORAGE;
```

Partitioned Tables and Indexes Examples

This section presents some examples for working with partitioned tables and indexes.

Moving the Time Window in a Historical Table

A **historical** table describes the business transactions of an enterprise over intervals of time. Historical tables can be **base** tables, which contain base information; for example, sales, checks, and orders. Historical tables can also be **rollup** tables, which contain summary information derived from the base information using operations such as `GROUP BY`, `AVERAGE`, or `COUNT`.

The time interval in a historical table is often a rolling window. DBAs periodically delete sets of rows that describe the oldest transactions, and in turn allocate space for sets of rows that describe the most recent transactions. For example, at the close of business on April 30, 1995, the DBA deletes the rows (and supporting index entries) that describe transactions from April 1994, and allocates space for the April 1995 transactions.

Now consider a specific example. You have a table, `order`, which contains 13 months of transactions: a year of historical data in addition to orders for the current month. There is one partition for each month. These monthly partitions are named `order_yymm`, as are the tablespaces in which they reside.

The `order` table contains two local indexes, `order_ix_onum`, which is a local, prefixed, unique index on the order number, and `order_ix_supp`, which is a local, non-prefixed index on the supplier number. The local index partitions are named with suffixes that match the underlying table. There is also a global unique index, `order_ix_cust`, for the customer name. `order_ix_cust` contains three partitions, one for each third of the alphabet. So on October 31, 1994, change the time window on `order` as follows:

1. Back up the data for the oldest time interval.

```
ALTER TABLESPACE order_9310 BEGIN BACKUP;
...
ALTER TABLESPACE order_9310 END BACKUP;
```

2. Drop the partition for the oldest time interval.

```
ALTER TABLE order DROP PARTITION order_9310;
```

3. Add the partition to the most recent time interval.

```
ALTER TABLE order ADD PARTITION order_9411;
```

4. Recreate the global index partitions.

```
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_AH;
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_IP;
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_QZ;
```

Ordinarily, Oracle acquires sufficient locks to ensure that no operation (DML, DDL, or utility) interferes with an individual DDL statement, such as `ALTER TABLE ... DROP PARTITION`. However, if the partition maintenance operation requires several steps, it is the DBA's responsibility to ensure that applications (or other maintenance operations) do not interfere with the multi-step operation in progress. Some methods for doing this are:

- Bring down all user-level applications during a well-defined batch window.
- Ensure that no one is able to access table `order` by revoking access privileges from a role that is used in all applications.

Converting a Partition View into a Partitioned Table

This scenario describes how to convert a partition view (also called "manual partition") into a partitioned table. The partition view is defined as follows:

```
CREATE VIEW accounts AS
```

```

SELECT * FROM accounts_jan98
UNION ALL
SELECT * FROM accounts_feb98
UNION ALL
...
SELECT * FROM accounts_dec98;

```

To incrementally migrate the partition view to a partitioned table, follow these steps:

1. Initially, only the two most recent partitions, `accounts_nov98` and `accounts_dec98`, will be migrated from the view to the table by creating the partitioned table. Each partition gets a segment of two blocks (as a placeholder).

```

CREATE TABLE accounts_new (...)
  TABLESPACE ts_temp STORAGE (INITIAL 2)
  PARTITION BY RANGE (opening_date)
    (PARTITION jan98 VALUES LESS THAN ('01-FEB-1998'),
     ...
     PARTITION dec98 VALUES LESS THAN ('01-JAN-1999'));

```

2. Use the `EXCHANGE PARTITION` statement to migrate the tables to the corresponding partitions.

```

ALTER TABLE accounts_new
  EXCHANGE PARTITION nov98 WITH TABLE
  accounts_nov98 WITH VALIDATION;

ALTER TABLE accounts_new
  EXCHANGE PARTITION dec98 WITH TABLE
  accounts_dec98 WITH VALIDATION;

```

So now the placeholder data segments associated with the `nov98` and `dec98` partitions have been exchanged with the data segments associated with the `accounts_nov98` and `accounts_dec98` tables.

3. Redefine the `accounts` view.

```

CREATE OR REPLACE VIEW accounts AS
  SELECT * FROM accounts_jan98
  UNION ALL
  SELECT * FROM accounts_feb_98
  UNION ALL
  ...
  UNION ALL
  SELECT * FROM accounts_new PARTITION (nov98)

```

```
UNION ALL
SELECT * FROM accounts_new PARTITION (dec98);
```

4. Drop the `accounts_nov98` and `accounts_dec98` tables, which own the placeholder segments that were originally attached to the `nov98` and `dec98` partitions.
5. After all the tables in the `UNION ALL` view are converted into partitions, drop the view and rename the partitioned to the name of the view being dropped.

```
DROP VIEW accounts;
RENAME accounts_new TO accounts;
```

Viewing Information About Partitioned Tables and Indexes

The following views display information specific to partitioned tables and indexes:

View	Description
DBA_PART_TABLES ALL_PART_TABLES USER_PART_TABLES	DBA view displays partitioning information for all partitioned tables in the database. ALL view displays partitioning information for all partitioned tables accessible to the user. USER view is restricted to partitioning information for partitioned tables owned by the user.
DBA_TAB_PARTITIONS ALL_TAB_PARTITIONS USER_TAB_PARTITIONS	Display partition-level partitioning information, partition storage parameters, and partition statistics generated by the <code>DBMS_STATS</code> package or the <code>ANALYZE</code> statement.
DBA_TAB_SUBPARTITIONS ALL_TAB_SUBPARTITIONS USER_TAB_SUBPARTITIONS	Display subpartition-level partitioning information, subpartition storage parameters, and subpartition statistics generated by the <code>DBMS_STATS</code> package or the <code>ANALYZE</code> statement.
DBA_PART_KEY_COLUMNS ALL_PART_KEY_COLUMNS USER_PART_KEY_COLUMNS	Display the partitioning key columns for partitioned tables.
DBA_SUBPART_KEY_COLUMNS ALL_SUBPART_KEY_COLUMNS USER_SUBPART_KEY_COLUMNS	Display the subpartitioning key columns for composite-partitioned tables (and local indexes on composite-partitioned tables).
DBA_PART_COL_STATISTICS ALL_PART_COL_STATISTICS USER_PART_COL_STATISTICS	Display column statistics and histogram information for the partitions of tables.

View	Description
DBA_SUBPART_COL_STATISTICS ALL_SUBPART_COL_STATISTICS USER_SUBPART_COL_STATISTICS	Display column statistics and histogram information for subpartitions of tables.
DBA_PART_HISTOGRAMS ALL_PART_HISTOGRAMS USER_PART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table partitions.
DBA_SUBPART_HISTOGRAMS ALL_SUBPART_HISTOGRAMS USER_SUBPART_HISTOGRAMS	Display the histogram data (end-points for each histogram) for histograms on table subpartitions.
DBA_PART_INDEXES ALL_PART_INDEXES USER_PART_INDEXES	Display partitioning information for partitioned indexes.
DBA_IND_PARTITIONS ALL_IND_PARTITIONS USER_IND_PARTITIONS	Display the following for index partitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.
DBA_IND_SUBPARTITIONS ALL_IND_SUBPARTITIONS USER_IND_SUBPARTITIONS	Display the following for index subpartitions: partition-level partitioning information, storage parameters for the partition, statistics collected by the DBMS_STATS package or the ANALYZE statement.

See Also:

- *Oracle9i Database Reference* for complete descriptions of these views
- *Oracle9i Database Performance Planning and Oracle9i Database Performance Tuning Guide and Reference* for information about histograms and generating statistics for tables
- "[Analyzing Tables, Indexes, and Clusters](#)" on page 21-3

Managing Clusters

This chapter describes aspects of managing clusters. It contains the following topics relating to the management of indexed clusters, clustered tables, and cluster indexes:

- [Guidelines for Managing Clusters](#)
- [Creating Clusters](#)
- [Altering Clusters](#)
- [Dropping Clusters](#)
- [Viewing Information About Clusters](#)

See Also:

- [Chapter 19, "Managing Hash Clusters"](#) for a description of another type of cluster: a hash cluster
- [Chapter 14, "Managing Space for Schema Objects"](#) is recommended reading before attempting tasks described in this chapter

Guidelines for Managing Clusters

A **cluster** provides an optional method of storing table data. A cluster is made up of a group of tables that share the same data blocks. The tables are grouped together because they share common columns and are often used together. For example, the `emp` and `dept` table share the `deptno` column. When you cluster the `emp` and `dept` tables (see [Figure 18-1](#)), Oracle physically stores all rows for each department from both the `emp` and `dept` tables in the same data blocks.

Because clusters store related rows of different tables together in the same data blocks, properly used clusters offer two primary benefits:

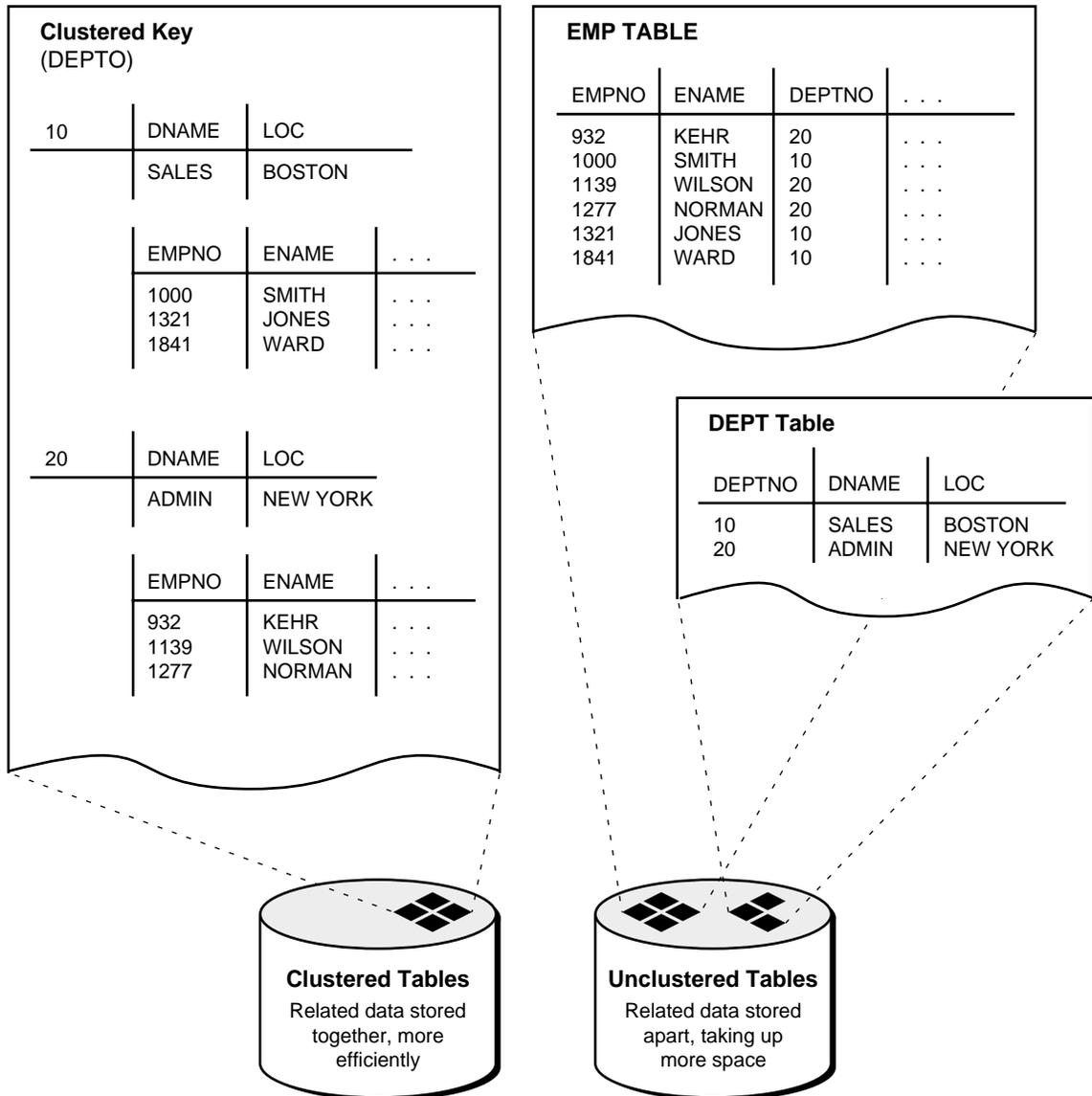
- Disk I/O is reduced and access time improves for joins of clustered tables.
- The **cluster key** is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating every table added to the cluster. Each cluster key value is stored only once each in the cluster and the cluster index, no matter how many rows of different tables contain the value.

Therefore, less storage might be required to store related table and index data in a cluster than is necessary in non-clustered table format. For example, in [Figure 18-1](#), notice how each cluster key (each `deptno`) is stored just once for many rows that contain the same value in both the `emp` and `dept` tables.

After creating a cluster, you can create tables in the cluster. However, before any rows can be inserted into the clustered tables, a cluster index must be created. Using clusters does not affect the creation of additional indexes on the clustered tables; they can be created and dropped as usual.

You should not use clusters for tables that are frequently accessed individually.

Figure 18–1 Clustered Table Data



The following sections describe guidelines to consider when managing clusters, and contains the following topics:

- [Choose Appropriate Tables for the Cluster](#)
- [Choose Appropriate Columns for the Cluster Key](#)
- [Specify Data Block Space Use](#)
- [Specify the Space Required by an Average Cluster Key and Its Associated Rows](#)
- [Specify the Location of Each Cluster and Cluster Index Rows](#)
- [Estimate Cluster Size and Set Storage Parameters](#)

See Also:

- *Oracle9i Database Concepts* for more information about clusters
- *Oracle9i Database Performance Tuning Guide and Reference* for guidelines on when to use clusters

Choose Appropriate Tables for the Cluster

Use clusters for tables for which the following conditions are true:

- The tables are primarily queried--that is, tables that are *not* predominantly inserted into or updated.
- Records from the tables are frequently queried together or joined.

Choose Appropriate Columns for the Cluster Key

Choose cluster key columns carefully. If multiple columns are used in queries that join the tables, make the cluster key a composite key. In general, the characteristics that indicate a good cluster index are the same as those for any index. For information about characteristics of a good index, see "[Guidelines for Managing Indexes](#)" on page 16-2.

A good cluster key has enough unique values so that the group of rows corresponding to each key value fills approximately one data block. Having too few rows for each cluster key value can waste space and result in negligible performance gains. Cluster keys that are so specific that only a few rows share a common value can cause wasted space in blocks, unless a small `SIZE` was specified at cluster creation time (see "[Specify the Space Required by an Average Cluster Key and Its Associated Rows](#)" on page 18-5).

Too many rows for each cluster key value can cause extra searching to find rows for that key. Cluster keys on values that are too general (for example, `male` and `female`) result in excessive searching and can result in worse performance than with no clustering.

A cluster index cannot be unique or include a column defined as `long`.

Specify Data Block Space Use

By specifying the `PCTFREE` and `PCTUSED` parameters during the creation of a cluster, you can affect the space utilization and amount of space reserved for updates to the current rows in the data blocks of a cluster's data segment. `PCTFREE` and `PCTUSED` parameters specified for tables created in a cluster are ignored; clustered tables automatically use the settings specified for the cluster.

See Also: ["Managing Space in Data Blocks"](#) on page 14-2 for information about setting the `PCTFREE` and `PCTUSED` parameters

Specify the Space Required by an Average Cluster Key and Its Associated Rows

The `CREATE CLUSTER` statement has an optional argument, `SIZE`, which is the estimated number of bytes required by an average cluster key and its associated rows. Oracle uses the `SIZE` parameter when performing the following tasks:

- Estimating the number of cluster keys (and associated rows) that can fit in a clustered data block
- Limiting the number of cluster keys placed in a clustered data block. This maximizes the storage efficiency of keys within a cluster.

`SIZE` does not limit the space that can be used by a given cluster key. For example, if `SIZE` is set such that two cluster keys can fit in one data block, any amount of the available data block space can still be used by either of the cluster keys.

By default, Oracle stores only one cluster key and its associated rows in each data block of the cluster's data segment. Although block size can vary from one operating system to the next, the rule of one key for each block is maintained as clustered tables are imported to other databases on other machines.

If all the rows for a given cluster key value cannot fit in one block, the blocks are chained together to speed access to all the values with the given key. The cluster index points to the beginning of the chain of blocks, each of which contains the cluster key value and associated rows. If the cluster `SIZE` is such that more than one key fits in a block, blocks can belong to more than one chain.

Specify the Location of Each Cluster and Cluster Index Rows

If you have the proper privileges and tablespace quota, you can create a new cluster and the associated cluster index in any tablespace that is currently online. Always specify the `TABLESPACE` option in a `CREATE CLUSTER/INDEX` statement to identify the tablespace to store the new cluster or index.

The cluster and its cluster index can be created in different tablespaces. In fact, creating a cluster and its index in different tablespaces that are stored on different storage devices allows table data and index data to be retrieved simultaneously with minimal disk contention.

Estimate Cluster Size and Set Storage Parameters

The following are benefits of estimating a cluster's size before creating it:

- You can use the combined estimated size of clusters, along with estimates for indexes, rollback segments, and redo log files, to determine the amount of disk space that is required to hold an intended database. From these estimates, you can make correct hardware purchases and other decisions.
- You can use the estimated size of an individual cluster to better manage the disk space that the cluster will use. When a cluster is created, you can set appropriate storage parameters and improve I/O performance of applications that use the cluster.

Whether or not you estimate table size before creation, you can explicitly set storage parameters when creating each non-clustered table. Any storage parameter that you do not explicitly set when creating or subsequently altering a table automatically uses the corresponding default storage parameter set for the tablespace in which the table resides. Clustered tables also automatically use the storage parameters of the cluster.

Creating Clusters

To create a cluster in your schema, you must have the `CREATE CLUSTER` system privilege and a quota for the tablespace intended to contain the cluster or the `UNLIMITED TABLESPACE` system privilege.

To create a cluster in another user's schema you must have the `CREATE ANY CLUSTER` system privilege, and the owner must have a quota for the tablespace intended to contain the cluster or the `UNLIMITED TABLESPACE` system privilege.

You create a cluster using the `CREATE CLUSTER` statement. The following statement creates a cluster named `emp_dept`, which stores the `emp` and `dept` tables, clustered by the `deptno` column:

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
  PCTUSED 80
  PCTIFREE 5
  SIZE 600
  TABLESPACE users
  STORAGE (INITIAL 200K
    NEXT 300K
    MINEXTENTS 2
    MAXEXTENTS 20
    PCTINCREASE 33);
```

If no `INDEX` keyword is specified, as is true in this example, an index cluster is created by default. You can also create a `HASH` cluster, when hash parameters (`HASHKEYS`, `HASH IS`, or `SINGLE TABLE HASHKEYS`) are specified. Hash clusters are described in [Chapter 19, "Managing Hash Clusters"](#).

See Also: *Oracle9i SQL Reference* for a more complete description of syntax, restrictions, and authorizations required for the SQL statements presented in this chapter

Creating Clustered Tables

To create a table in a cluster, you must have either the `CREATE TABLE` or `CREATE ANY TABLE` system privilege. You do not need a tablespace quota or the `UNLIMITED TABLESPACE` system privilege to create a table in a cluster.

You create a table in a cluster using the `CREATE TABLE` statement with the `CLUSTER` option. The `emp` and `dept` tables can be created in the `emp_dept` cluster using the following statements:

```
CREATE TABLE emp (
  empno NUMBER(5) PRIMARY KEY,
  ename VARCHAR2(15) NOT NULL,
  . . .
  deptno NUMBER(3) REFERENCES dept)
  CLUSTER emp_dept (deptno);

CREATE TABLE dept (
  deptno NUMBER(3) PRIMARY KEY, . . . )
  CLUSTER emp_dept (deptno);
```

Note: You can specify the schema for a clustered table in the `CREATE TABLE` statement. A clustered table can be in a different schema than the schema containing the cluster. Also, the names of the columns are not required to match, but their structure must match.

Creating Cluster Indexes

To create a cluster index, one of the following conditions must be true:

- Your schema contains the cluster.
- You have the `CREATE ANY INDEX` system privilege.

In either case, you must also have either a quota for the tablespace intended to contain the cluster index, or the `UNLIMITED TABLESPACE` system privilege.

A cluster index must be created before any rows can be inserted into any clustered table. The following statement creates a cluster index for the `emp_dept` cluster:

```
CREATE INDEX emp_dept_index
ON CLUSTER emp_dept
INITRANS 2
MAXTRANS 5
TABLESPACE users
STORAGE (INITIAL 50K
NEXT 50K
MINEXTENTS 2
MAXEXTENTS 10
PCTINCREASE 33)
PCTFREE 5;
```

The cluster index clause (`ON CLUSTER`) identifies the cluster, `emp_dept`, for which the cluster index is being created. The statement also explicitly specifies several storage settings for the cluster and cluster index.

Altering Clusters

To alter a cluster, your schema must contain the cluster or you must have the `ALTER ANY CLUSTER` system privilege. You can alter an existing cluster to change the following settings:

- Physical attributes (`PCTFREE`, `PCTUSED`, `INITRANS`, `MAXTRANS`, and storage characteristics)

- The average amount of space required to store all the rows for a cluster key value (`SIZE`)
- The default degree of parallelism

Additionally, you can explicitly allocate a new extent for the cluster, or deallocate any unused extents at the end of the cluster. Oracle dynamically allocates additional extents for the data segment of a cluster as required. In some circumstances, however, you might want to explicitly allocate an additional extent for a cluster. For example, when using Oracle9i Real Application Clusters, you can allocate an extent of a cluster explicitly for a specific instance. You allocate a new extent for a cluster using the `ALTER CLUSTER` statement with the `ALLOCATE EXTENT` clause.

When you alter data block space usage parameters (`PCTFREE` and `PCTUSED`) or the cluster size parameter (`SIZE`) of a cluster, the new settings apply to all data blocks used by the cluster, including blocks already allocated and blocks subsequently allocated for the cluster. Blocks already allocated for the table are reorganized when necessary (not immediately).

When you alter the transaction entry settings (`INITTRANS` and `MAXTRANS`) of a cluster, a new setting for `INITTRANS` applies only to data blocks subsequently allocated for the cluster, while a new setting for `MAXTRANS` applies to all blocks (already and subsequently allocated blocks) of a cluster.

The storage parameters `INITIAL` and `MINEXTENTS` cannot be altered. All new settings for the other storage parameters affect only extents subsequently allocated for the cluster.

To alter a cluster, use the `ALTER CLUSTER` statement. The following statement alters the `emp_dept` cluster:

```
ALTER CLUSTER emp_dept
  PCTFREE 30
  PCTUSED 60;
```

See Also: *Oracle9i Real Application Clusters Administration* for specific uses of the `ALTER CLUSTER` statement in an Oracle Real Application Clusters environment

Altering Clustered Tables

You can alter clustered tables using the `ALTER TABLE` statement. However, any data block space parameters, transaction entry parameters, or storage parameters you set in an `ALTER TABLE` statement for a clustered table generate an error message (ORA-01771, illegal option for a clustered table). Oracle

uses the parameters of the cluster for all clustered tables. Therefore, you can use the `ALTER TABLE` statement only to add or modify columns, drop non-cluster key columns, or add, drop, enable, or disable integrity constraints or triggers for a clustered table. For information about altering tables, see "[Altering Tables](#)" on page 15-10.

Altering Cluster Indexes

You alter cluster indexes exactly as you do other indexes. See "[Altering Indexes](#)" on page 16-19.

Note: When estimating the size of cluster indexes, remember that the index is on each cluster key, not the actual rows. Therefore, each key appears only once in the index.

Dropping Clusters

A cluster can be dropped if the tables within the cluster are no longer necessary. When a cluster is dropped, so are the tables within the cluster and the corresponding cluster index. All extents belonging to both the cluster's data segment and the index segment of the cluster index are returned to the containing tablespace and become available for other segments within the tablespace.

To drop a cluster that contains no tables, and its cluster index, use the `DROP CLUSTER` statement. For example, the following statement drops the empty cluster named `emp_dept`:

```
DROP CLUSTER emp_dept;
```

If the cluster contains one or more clustered tables and you intend to drop the tables as well, add the `INCLUDING TABLES` option of the `DROP CLUSTER` statement, as follows:

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

If the `INCLUDING TABLES` option is not included and the cluster contains tables, an error is returned.

If one or more tables in a cluster contain primary or unique keys that are referenced by `FOREIGN KEY` constraints of tables outside the cluster, the cluster cannot be dropped unless the dependent `FOREIGN KEY` constraints are also dropped. This

can be easily done using the `CASCADE CONSTRAINTS` option of the `DROP CLUSTER` statement, as shown in the following example:

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

Oracle returns an error if you do not use the `CASCADE CONSTRAINTS` option and constraints exist.

Dropping Clustered Tables

To drop a cluster, your schema must contain the cluster or you must have the `DROP ANY CLUSTER` system privilege. You do not need additional privileges to drop a cluster that contains tables, even if the clustered tables are not owned by the owner of the cluster.

Clustered tables can be dropped individually without affecting the table's cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a non-clustered table is dropped—with the `DROP TABLE` statement. See "[Dropping Tables](#)" on page 15-23.

Note: When you drop a single table from a cluster, Oracle deletes each row of the table individually. To maximize efficiency when you intend to drop an entire cluster, drop the cluster including all tables by using the `DROP CLUSTER` statement with the `INCLUDING TABLES` option. Drop an individual table from a cluster (using the `DROP TABLE` statement) only if you want the rest of the cluster to remain.

Dropping Cluster Indexes

A cluster index can be dropped without affecting the cluster or its clustered tables. However, clustered tables cannot be used if there is no cluster index; you must re-create the cluster index to allow access to the cluster. Cluster indexes are sometimes dropped as part of the procedure to rebuild a fragmented cluster index. For information about dropping an index, see "[Dropping Indexes](#)" on page 16-22.

Viewing Information About Clusters

The following views display information about clusters:

View	Description
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA view describes all clusters in the database. ALL view describes all clusters accessible to the user. USER view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_CLU_COLUMNS USER_CLU_COLUMNS	These views map table columns to cluster columns

See Also: *Oracle9i Database Reference* for complete descriptions of these views

Managing Hash Clusters

This chapter describes how to manage hash clusters, and contains the following topics:

- [When to Use Hash Clusters](#)
- [Creating Hash Clusters](#)
- [Altering Hash Clusters](#)
- [Dropping Hash Clusters](#)
- [Viewing Information About Hash Clusters](#)

See Also: [Chapter 14, "Managing Space for Schema Objects"](#) is recommended reading before attempting tasks described in this chapter.

When to Use Hash Clusters

Storing a table in a hash cluster is an optional way to improve the performance of data retrieval. A hash cluster provides an alternative to a nonclustered table with an index or an index cluster. With an indexed table or index cluster, Oracle locates the rows in a table using key values that Oracle stores in a separate index. To use hashing, you create a hash cluster and load tables into it. Oracle physically stores the rows of a table in a hash cluster and retrieves them according to the results of a **hash function**.

Oracle uses a hash function to generate a distribution of numeric values, called **hash values**, that are based on specific cluster key values. The key of a hash cluster, like the key of an index cluster, can be a single column or composite key (multiple column key). To find or store a row in a hash cluster, Oracle applies the hash function to the row's cluster key value. The resulting hash value corresponds to a data block in the cluster, which Oracle then reads or writes on behalf of the issued statement.

To find or store a row in an indexed table or cluster, a minimum of two (there are usually more) I/Os must be performed:

- One or more I/Os to find or store the key value in the index
- Another I/O to read or write the row in the table or cluster

In contrast, Oracle uses a hash function to locate a row in a hash cluster; no I/O is required. As a result, a minimum of one I/O operation is necessary to read or write a row in a hash cluster.

This section helps you decide when to use hash clusters by contrasting situations where hashing is most useful against situations where there is no advantage. If you find your decision is to use indexing rather than hashing, then you should consider whether to store a table individually or as part of a cluster.

Note: Even if you decide to use hashing, a table can still have separate indexes on any columns, including the cluster key.

See Also:

- *Oracle9i Database Concepts* for more information about hash clusters
- *Oracle9i Application Developer's Guide - Fundamentals* for additional recommendations on the use of hash clusters

Situations Where Hashing Is Useful

Hashing is useful when you have the following conditions:

- Most queries are equality queries on the cluster key:

```
SELECT ... WHERE cluster_key = ...;
```

In such cases, the cluster key in the equality condition is hashed, and the corresponding hash key is usually found with a single read. In comparison, for an indexed table the key value must first be found in the index (usually several reads), and then the row is read from the table (another read).

- The tables in the hash cluster are primarily static in size so that you can determine the number of rows and amount of space required for the tables in the cluster. If tables in a hash cluster require more space than the initial allocation for the cluster, performance degradation can be substantial because overflow blocks are required.

Situations Where Hashing Is Not Advantageous

Hashing is not advantageous in the following situations:

- Most queries on the table retrieve rows over a range of cluster key values. For example, in full table scans or queries such as the following, a hash function cannot be used to determine the location of specific hash keys. Instead, the equivalent of a full table scan must be done to fetch the rows for the query.

```
SELECT . . . WHERE cluster_key < . . . ;
```

With an index, key values are ordered in the index, so cluster key values that satisfy the `WHERE` clause of a query can be found with relatively few I/Os.

- The table is not static, but instead is continually growing. If a table grows without limit, the space required over the life of the table (its cluster) cannot be predetermined.
- Applications frequently perform full-table scans on the table and the table is sparsely populated. A full-table scan in this situation takes longer under hashing.
- You cannot afford to preallocate the space that the hash cluster will eventually need.

Creating Hash Clusters

A hash cluster is created using a `CREATE CLUSTER` statement, but you specify a `HASHKEYS` clause. The following example contains a statement to create a cluster named `trial_cluster` that stores the `trial` table, clustered by the `trialno` column (the cluster key); and another statement creating a table in the cluster.

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
  PCTUSED 80
  PCTFREE 5
  TABLESPACE users
  STORAGE (INITIAL 250K      NEXT 50K
           MINEXTENTS 1     MAXEXTENTS 3
           PCTINCREASE 0)
  HASH IS trialno HASHKEYS 150;

CREATE TABLE trial (
  trialno NUMBER(5,0) PRIMARY KEY,
  ...)
  CLUSTER trial_cluster (trialno);
```

As with index clusters, the key of a hash cluster can be a single column or a composite key (multiple column key). In this example, it is a single column.

The `HASHKEYS` value, in this case 150, specifies and limits the number of unique hash values that can be generated by the hash function used by the cluster. Oracle rounds the number specified to the nearest prime number.

If no `HASH IS` clause is specified, Oracle uses an internal hash function. If the cluster key is already a unique identifier that is uniformly distributed over its range, you can bypass the internal hash function and specify the cluster key as the hash value, as is the case in the above example. You can also use the `HASH IS` clause to specify a user-defined hash function.

You cannot create a cluster index on a hash cluster, and you need not create an index on a hash cluster key.

For additional information about creating tables in a cluster, guidelines for setting parameters of the `CREATE CLUSTER` statement common to index and hash clusters, and the privileges required to create any cluster, see [Chapter 18, "Managing Clusters"](#). The following sections explain and provide guidelines for setting the parameters of the `CREATE CLUSTER` statement specific to hash clusters:

- [Creating Single-Table Hash Clusters](#)
- [Controlling Space Use Within a Hash Cluster](#)

- [Estimating Size Required by Hash Clusters](#)

See Also:

- *Oracle9i Database Concepts* for a discussion of hash functions and specifying user-defined hash functions
- *Oracle9i SQL Reference* for a more complete description of syntax, restrictions, and authorizations required for the SQL statements `CREATE CLUSTER` and `CREATE TABLE`

Creating Single-Table Hash Clusters

You can also create a **single-table hash cluster**, which provides fast access to rows in a table. However, this table must be the only table in the hash cluster. Essentially, there must be a one-to-one mapping between hash keys and data rows. The following statement creates a single-table hash cluster named `peanut` with the cluster key `variety`:

```
CREATE CLUSTER peanut (variety NUMBER)
  SIZE 512 SINGLE TABLE HASHKEYS 500;
```

Oracle rounds the `HASHKEY` value up to the nearest prime number, so this cluster has a maximum of 503 hash key values, each of size 512 bytes.

Note: The `SINGLE TABLE` option is valid only for hash clusters. `HASHKEYS` must also be specified.

Controlling Space Use Within a Hash Cluster

When creating a hash cluster, it is important to choose the cluster key correctly and set the `HASH IS`, `SIZE`, and `HASHKEYS` parameters so that performance and space use are optimal. The following guidelines describe how to set these parameters.

Choosing the Key

Choosing the correct cluster key is dependent on the most common types of queries issued against the clustered tables. For example, consider the `emp` table in a hash cluster. If queries often select rows by employee number, the `empno` column should be the cluster key. If queries often select rows by department number, the `deptno` column should be the cluster key. For hash clusters that contain a single table, the cluster key is typically the entire primary key of the contained table.

The key of a hash cluster, like that of an index cluster, can be a single column or a composite key (multiple column key). A hash cluster with a composite key must use Oracle's internal hash function.

Setting HASH IS

Specify the `HASH IS` parameter only if the cluster key is a single column of the `NUMBER` datatype, and contains uniformly distributed integers. If the above conditions apply, you can distribute rows in the cluster so that each unique cluster key value hashes, with no collisions (two cluster key values having the same hash value), to a unique hash value. If these conditions do not apply, omit this option so that you use the internal hash function.

Setting SIZE

`SIZE` should be set to the average amount of space required to hold all rows for any given hash key. Therefore, to properly determine `SIZE`, you must be aware of the characteristics of your data:

- If the hash cluster is to contain only a single table and the hash key values of the rows in that table are unique (one row for each value), `SIZE` can be set to the average row size in the cluster.
- If the hash cluster is to contain multiple tables, `SIZE` can be set to the average amount of space required to hold all rows associated with a representative hash value.

Further, once you have determined a (preliminary) value for `SIZE`, consider the following. If the `SIZE` value is small (more than four hash keys can be assigned for each data block) you can use this value for `SIZE` in the `CREATE CLUSTER` statement. However, if the value of `SIZE` is large (four or fewer hash keys can be assigned for each data block), then you should also consider the expected frequency of collisions and whether performance of data retrieval or efficiency of space usage is more important to you.

- If the hash cluster does not use the internal hash function (if you specified `HASH IS`) and you expect few or no collisions, you can use your preliminary value of `SIZE`. No collisions occur and space is used as efficiently as possible.
- If you expect frequent collisions on inserts, the likelihood of overflow blocks being allocated to store rows is high. To reduce the possibility of overflow blocks and maximize performance when collisions are frequent, you should adjust `SIZE` as shown in the following chart.

Available Space for each Block/Calculated SIZE	Setting for SIZE
1	SIZE
2	SIZE + 15%
3	SIZE + 12%
4	SIZE + 8%
>4	SIZE

Overestimating the value of `SIZE` increases the amount of unused space in the cluster. If space efficiency is more important than the performance of data retrieval, disregard the above adjustments and use the original value for `SIZE`.

Setting HASHKEYS

For maximum distribution of rows in a hash cluster, Oracle rounds the `HASHKEYS` value up to the nearest prime number.

Controlling Space in Hash Clusters: Examples

The following examples show how to correctly choose the cluster key and set the `HASH IS`, `SIZE`, and `HASHKEYS` parameters. For all examples, assume that the data block size is 2K and that on average, 1950 bytes of each block is available data space (block size minus overhead).

Example 1 You decide to load the `emp` table into a hash cluster. Most queries retrieve employee records by their employee number. You estimate that the maximum number of rows in the `emp` table at any given time is 10000 and that the average row size is 55 bytes.

In this case, `empno` should be the cluster key. Since this column contains integers that are unique, the internal hash function can be bypassed. `SIZE` can be set to the average row size, 55 bytes. Note that 34 hash keys are assigned for each data block. `HASHKEYS` can be set to the number of rows in the table, 10000. Oracle rounds this value up to the next highest prime number: 10007.

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```

Example 2 Conditions similar to the previous example exist. In this case, however, rows are usually retrieved by department number. At most, there are 1000 departments with an average of 10 employees for each department. Department numbers increment by 10 (0, 10, 20, 30, . . .).

In this case, deptno should be the cluster key. Since this column contains integers that are uniformly distributed, the internal hash function can be bypassed. A preliminary value of SIZE (the average amount of space required to hold all rows for each department) is 55 bytes * 10, or 550 bytes. Using this value for SIZE, only three hash keys can be assigned for each data block. If you expect some collisions and want maximum performance of data retrieval, slightly alter your estimated SIZE to prevent collisions from requiring overflow blocks. By adjusting SIZE by 12%, to 620 bytes (refer to "Setting SIZE" on page 19-6), there is more space for rows from expected collisions.

HASHKEYS can be set to the number of unique department numbers, 1000. Oracle rounds this value up to the next highest prime number: 1009.

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
HASH IS deptno HASHKEYS 1000;
```

Estimating Size Required by Hash Clusters

As with index clusters, it is important to estimate the storage required for the data in a hash cluster.

Oracle guarantees that the initial allocation of space is sufficient to store the hash table according to the settings SIZE and HASHKEYS. If settings for the storage parameters INITIAL, NEXT, and MINEXTENTS do not account for the hash table size, incremental (additional) extents are allocated until at least SIZE*HASHKEYS is reached. For example, assume that the data block size is 2K, the available data space for each block is approximately 1900 bytes (data block size minus overhead), and that the STORAGE and HASH parameters are specified in the CREATE CLUSTER statement as follows:

```
STORAGE (INITIAL 100K
NEXT 150K
MINEXTENTS 1
PCTINCREASE 0)
SIZE 1500
HASHKEYS 100
```

In this example, only one hash key can be assigned for each data block. Therefore, the initial space required for the hash cluster is at least 100*2K or 200K. The settings for the storage parameters do not account for this requirement. Therefore, an initial extent of 100K and a second extent of 150K are allocated to the hash cluster.

Alternatively, assume the `HASH` parameters are specified as follows:

```
SIZE 500 HASHKEYS 100
```

In this case, three hash keys are assigned to each data block. Therefore, the initial space required for the hash cluster is at least 34*2K or 68K. The initial settings for the storage parameters are sufficient for this requirement (an initial extent of 100K is allocated to the hash cluster).

Altering Hash Clusters

You can alter a hash cluster with the `ALTER CLUSTER` statement:

```
ALTER CLUSTER emp_dept . . . ;
```

The implications for altering a hash cluster are identical to those for altering an index cluster, described in "[Altering Clusters](#)" on page 18-8. However, the `SIZE`, `HASHKEYS`, and `HASH IS` parameters cannot be specified in an `ALTER CLUSTER` statement. To change these parameters, you must re-create the cluster, then copy the data from the original cluster.

Dropping Hash Clusters

You can drop a hash cluster using the `DROP CLUSTER` statement:

```
DROP CLUSTER emp_dept ;
```

A table in a hash cluster is dropped using the `DROP TABLE` statement. The implications of dropping hash clusters and tables in hash clusters are the same as those for dropping index clusters.

See Also: "[Dropping Clusters](#)" on page 18-10

Viewing Information About Hash Clusters

The following views display information about hash clusters:

View	Description
DBA_CLUSTERS ALL_CLUSTERS USER_CLUSTERS	DBA view describes all clusters (including hash clusters) in the database. ALL view describes all clusters accessible to the user. USER view is restricted to clusters owned by the user. Some columns in these views contain statistics that are generated by the DBMS_STATS package or ANALYZE statement.
DBA_CLU_COLUMNS USER_CLU_COLUMNS	These views map table columns to cluster columns.
DBA_CLUSTER_HASH_EXPRESSIONS ALL_CLUSTER_HASH_EXPRESSIONS USER_CLUSTER_HASH_EXPRESSIONS	These views list hash functions for hash clusters.

See Also: *Oracle9i Database Reference* for complete descriptions of these views

Managing Views, Sequences, and Synonyms

This chapter describes the management of views, sequences, and synonyms and contains the following topics:

- [Managing Views](#)
- [Managing Sequences](#)
- [Managing Synonyms](#)
- [Viewing Information About Views, Synonyms, and Sequences](#)

Managing Views

A view is a tailored presentation of the data contained in one or more tables (or other views), and takes the output of a query and treats it as a table. You can think of a view as a "stored query" or a "virtual table." You can use views in most places where a table can be used.

This section describes aspects of managing views, and contains the following topics:

- [Creating Views](#)
- [Updating a Join View](#)
- [Altering Views](#)
- [Dropping Views](#)
- [Replacing Views](#)

Creating Views

To create a view, you must meet the following requirements:

- To create a view in your schema, you must have the `CREATE VIEW` privilege. To create a view in another user's schema, you must have the `CREATE ANY VIEW` system privilege. You can acquire these privileges explicitly or through a role.
- The owner of the view (whether it is you or another user) must have been explicitly granted privileges to access all objects referenced in the view definition. The owner *cannot* have obtained these privileges through roles. Also, the functionality of the view is dependent on the privileges of the view's owner. For example, if the owner of the view has only the `INSERT` privilege for Scott's `emp` table, the view can only be used to insert new rows into the `emp` table, not to `SELECT`, `UPDATE`, or `DELETE` rows.
- If the owner of the view intends to grant access to the view to other users, the owner must have received the object privileges to the base objects with the `GRANT OPTION` or the system privileges with the `ADMIN OPTION`.

You can create views using the `CREATE VIEW` statement. Each view is defined by a query that references tables, materialized views, or other views. As with all subqueries, the query that defines a view cannot contain the `FOR UPDATE` clause.

The following statement creates a view on a subset of data in the `emp` table:

```
CREATE VIEW sales_staff AS
    SELECT empno, ename, deptno
```

```
FROM emp
WHERE deptno = 10
WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

The query that defines the `sales_staff` view references only rows in department 10. Furthermore, the `CHECK OPTION` creates the view with the constraint (named `sales_staff_cnst`) that `INSERT` and `UPDATE` statements issued against the view cannot result in rows that the view cannot select. For example, the following `INSERT` statement successfully inserts a row into the `emp` table by means of the `sales_staff` view, which contains all rows with department number 10:

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```

However, the following `INSERT` statement returns an error because it attempts to insert a row for department number 30, which cannot be selected using the `sales_staff` view:

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

The view could optionally have been constructed specifying the `WITH READ ONLY` clause, which prevents any updates, inserts, or deletes from being done to the base table through the view. If no `WITH` clause is specified, the view, with some restrictions, is inherently updatable.

See Also: *Oracle9i SQL Reference* for detailed syntax, restriction, and authorization information relating to creating and maintaining views

Join Views

You can also create views that specify more than one base table or view in the `FROM` clause. These are called **join views**. The following statement creates the `division1_staff` view that joins data from the `emp` and `dept` tables:

```
CREATE VIEW division1_staff AS
SELECT ename, empno, job, dname
FROM emp, dept
WHERE emp.deptno IN (10, 30)
AND emp.deptno = dept.deptno;
```

An **updatable join view** is a join view where `UPDATE`, `INSERT`, and `DELETE` operations are allowed. See ["Updating a Join View"](#) on page 20-5 for further discussion.

Expansion of Defining Queries at View Creation Time

When a view is created, Oracle expands any wildcard (*) in a top-level view query into a column list. The resulting query is stored in the data dictionary; any subqueries are left intact. The column names in an expanded column list are enclosed in quote marks to account for the possibility that the columns of the base object were originally entered with quotes and require them for the query to be syntactically correct.

As an example, assume that the `dept` view is created as follows:

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

Oracle stores the defining query of the `dept` view as:

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept;
```

Views created with errors do not have wildcards expanded. However, if the view is eventually compiled without errors, wildcards in the defining query are expanded.

Creating Views with Errors

If there are no syntax errors in a `CREATE VIEW` statement, Oracle can create the view even if the defining query of the view cannot be executed. In this case, the view is considered "created with errors." For example, when a view is created that refers to a nonexistent table or an invalid column of an existing table, or when the view owner does not have the required privileges, the view can be created anyway and entered into the data dictionary. However, the view is not yet usable.

To create a view with errors, you must include the `FORCE` option of the `CREATE VIEW` statement.

```
CREATE FORCE VIEW AS ...;
```

By default, views with errors are not created as `VALID`. When you try to create such a view, Oracle returns a message indicating the view was created with errors. The status of a view created with errors is `INVALID`. If conditions later change so that the query of an invalid view can be executed, the view can be recompiled and be made valid (usable). For information changing conditions and their impact on views, see "[Managing Object Dependencies](#)" on page 21-23.

Updating a Join View

An updatable join view (also referred to as a **modifiable join view**) is a view that contains more than one table in the top-level `FROM` clause of the `SELECT` statement, and is not restricted by the `WITH READ ONLY` clause.

Note: There are some restrictions and conditions which can affect whether a join view is updatable. Specifics are listed in the description of the `CREATE VIEW` statement in the *Oracle9i SQL Reference*.

Additionally, if a view is a join on other nested views, then the other nested views must be mergeable into the top level view. For a discussion of mergeable and unmergeable views, and more generally, how the optimizer optimizes statements referencing views, see *Oracle9i Database Concepts* and *Oracle9i Database Performance Tuning Guide and Reference*.

There are data dictionary views that indicate whether the columns in a join view are updatable. See "[Using the UPDATABLE_COLUMNS Views](#)" on page 20-9 for descriptions of these views.

The rules for updatable join views are as follows:

Rule	Description
General Rule	Any <code>INSERT</code> , <code>UPDATE</code> , or <code>DELETE</code> operation on a join view can modify only one underlying base table at a time.
<code>UPDATE</code> Rule	All updatable columns of a join view must map to columns of a key-preserved table . See " Key-Preserved Tables " on page 20-6 for a discussion of key-preserved tables. If the view is defined with the <code>WITH CHECK OPTION</code> clause, then all join columns and all columns of repeated tables are non-updatable.
<code>DELETE</code> Rule	Rows from a join view can be deleted as long as there is exactly one key-preserved table in the join. If the view is defined with the <code>WITH CHECK OPTION</code> clause and the key preserved table is repeated, then the rows cannot be deleted from the view.
<code>INSERT</code> Rule	An <code>INSERT</code> statement must not explicitly or implicitly refer to the columns of a nonkey preserved table . If the join view is defined with the <code>WITH CHECK OPTION</code> clause, <code>INSERT</code> statements are not permitted.

Examples illustrating these rules, and a discussion of key-preserved tables, are presented in succeeding sections.

The examples given work only if you explicitly define the primary and foreign keys in the tables, or define unique indexes. Following are the appropriately constrained table definitions for `emp` and `dept`.

```
CREATE TABLE dept (  
    deptno      NUMBER(4) PRIMARY KEY,  
    dname       VARCHAR2(14),  
    loc         VARCHAR2(13));  
  
CREATE TABLE emp (  
    empno       NUMBER(4) PRIMARY KEY,  
    ename       VARCHAR2(10),  
    job        VARCHAR2(9),  
    mgr        NUMBER(4),  
    sal        NUMBER(7,2),  
    comm       NUMBER(7,2),  
    deptno     NUMBER(2),  
    FOREIGN KEY (DEPTNO) REFERENCES DEPT(DEPTNO));
```

You could also omit the primary and foreign key constraints listed above, and create a `UNIQUE INDEX` on `dept (deptno)` to make the following examples work.

The following statement created the `emp_dept` join view which is referenced in the examples:

```
CREATE VIEW emp_dept AS  
    SELECT emp.empno, emp.ename, emp.deptno, emp.sal, dept.dname, dept.loc  
    FROM emp, dept  
    WHERE emp.deptno = dept.deptno  
           AND dept.loc IN ('DALLAS', 'NEW YORK', 'BOSTON');
```

Key-Preserved Tables

The concept of a **key-preserved table** is fundamental to understanding the restrictions on modifying join views. A table is key preserved if every key of the table can also be a key of the result of the join. So, a key-preserved table has its keys preserved through a join.

Note: It is not necessary that the key or keys of a table be selected for it to be key preserved. It is sufficient that if the key or keys were selected, then they would also be key(s) of the result of the join.

The key-preserving property of a table does not depend on the actual data in the table. It is, rather, a property of its schema. For example, if in the `emp` table there was at most one employee in each department, then `deptno` would be unique in the result of a join of `emp` and `dept`, but `dept` would still not be a key-preserved table.

If you `SELECT` all rows from `emp_dept`, the results are:

EMPNO	ENAME	DEPTNO	DNAME	LOC
7782	CLARK	10	ACCOUNTING	NEW YORK
7839	KING	10	ACCOUNTING	NEW YORK
7934	MILLER	10	ACCOUNTING	NEW YORK
7369	SMITH	20	RESEARCH	DALLAS
7876	ADAMS	20	RESEARCH	DALLAS
7902	FORD	20	RESEARCH	DALLAS
7788	SCOTT	20	RESEARCH	DALLAS
7566	JONES	20	RESEARCH	DALLAS

8 rows selected.

In this view, `emp` is a key-preserved table, because `empno` is a key of the `emp` table, and also a key of the result of the join. `dept` is *not* a key-preserved table, because although `deptno` is a key of the `dept` table, it is not a key of the join.

DML Statements and Join Views

The general rule is that any `UPDATE`, `DELETE`, or `INSERT` statement on a join view can modify only one underlying base table. The following examples illustrate rules specific to `UPDATE`, `DELETE`, and `INSERT` statements.

UPDATE Statements The following example shows an `UPDATE` statement that successfully modifies the `emp_dept` view:

```
UPDATE emp_dept
   SET sal = sal * 1.10
   WHERE deptno = 10;
```

The following `UPDATE` statement would be disallowed on the `emp_dept` view:

```
UPDATE emp_dept
   SET loc = 'BOSTON'
   WHERE ename = 'SMITH';
```

This statement fails with an error (ORA-01779 cannot modify a column which maps to a non key-preserved table), because it attempts to

modify the base `dept` table, and the `dept` table is not key preserved in the `emp_dept` view.

In general, all updatable columns of a join view must map to columns of a key-preserved table. If the view is defined using the `WITH CHECK OPTION` clause, then all join columns and all columns taken from tables that are referenced more than once in the view are not modifiable.

So, for example, if the `emp_dept` view were defined using `WITH CHECK OPTION`, the following `UPDATE` statement would fail:

```
UPDATE emp_dept
   SET deptno = 10
   WHERE ename = 'SMITH';
```

The statement fails because it is trying to update a join column.

DELETE Statements You can delete from a join view provided there is *one and only one* key-preserved table in the join.

The following `DELETE` statement works on the `emp_dept` view:

```
DELETE FROM emp_dept
   WHERE ename = 'SMITH';
```

This `DELETE` statement on the `emp_dept` view is legal because it can be translated to a `DELETE` operation on the base `emp` table, and because the `emp` table is the only key-preserved table in the join.

If you were to create the following view, a `DELETE` operation could not be performed on the view because both `e1` and `e2` are key-preserved tables:

```
CREATE VIEW emp_emp AS
   SELECT e1.ename, e2.empno, deptno
   FROM emp e1, emp e2
   WHERE e1.empno = e2.empno;
```

If a view is defined using the `WITH CHECK OPTION` clause and the key-preserved table is repeated, then rows cannot be deleted from such a view.

```
CREATE VIEW emp_mgr AS
   SELECT e1.ename, e2.ename mname
   FROM emp e1, emp e2
   WHERE e1.mgr = e2.empno
   WITH CHECK OPTION;
```

No deletion can be performed on this view because the view involves a self-join of the table that is key preserved.

INSERT Statements The following `INSERT` statement on the `emp_dept` view succeeds:

```
INSERT INTO emp_dept (ename, empno, deptno)
VALUES ('KURODA', 9010, 40);
```

This statement works because only one key-preserved base table is being modified (`emp`), and `40` is a valid `deptno` in the `dept` table (thus satisfying the `FOREIGN KEY` integrity constraint on the `emp` table).

An `INSERT` statement, such as the following, would fail for the same reason that such an `UPDATE` on the base `emp` table would fail: the `FOREIGN KEY` integrity constraint on the `emp` table is violated (because there is no `deptno 77`).

```
INSERT INTO emp_dept (ename, empno, deptno)
VALUES ('KURODA', 9010, 77);
```

The following `INSERT` statement would fail with an error (ORA-01776 cannot modify more than one base table through a view):

```
INSERT INTO emp_dept (empno, ename, loc)
VALUES (9010, 'KURODA', 'BOSTON');
```

An `INSERT` cannot implicitly or explicitly refer to columns of a nonkey-preserved table. If the join view is defined using the `WITH CHECK OPTION` clause, then you cannot perform an `INSERT` to it.

Using the `UPDATABLE_ COLUMNS` Views

The views described in the following table can assist you when modifying join views.

View	Description
<code>DBA_UPDATABLE_ COLUMNS</code>	Shows all columns in all tables and views that are modifiable.
<code>ALL_UPDATABLE_ COLUMNS</code>	Shows all columns in all tables and views accessible to the user that are modifiable.
<code>USER_UPDATABLE_ COLUMNS</code>	Shows all columns in all tables and views in the user's schema that are modifiable.

The updatable columns in view `emp_dept` are shown below.

```
SELECT COLUMN_NAME, UPDATABLE
       FROM USER_UPDATABLE_COLUMNS
       WHERE TABLE_NAME = 'EMP_DEPT';
```

COLUMN_NAME	UPD
-----	----
EMPNO	YES
ENAME	YES
DEPTNO	YES
SAL	YES
DNAME	NO
LOC	NO

6 rows selected.

Altering Views

You use the `ALTER VIEW` statement only to explicitly recompile a view that is invalid. If you want to change the definition of a view, see ["Replacing Views"](#) on page 20-10.

The `ALTER VIEW` statement allows you to locate recompilation errors before run time. To ensure that the alteration does not affect the view or other objects that depend on it, you can explicitly recompile a view after altering one of its base tables.

To use the `ALTER VIEW` statement, the view must be in your schema, or you must have the `ALTER ANY TABLE` system privilege.

Dropping Views

You can drop any view contained in your schema. To drop a view in another user's schema, you must have the `DROP ANY VIEW` system privilege. Drop a view using the `DROP VIEW` statement. For example, the following statement drops the `emp_dept` view:

```
DROP VIEW emp_dept;
```

Replacing Views

To replace a view, you must have all the privileges required to drop and create a view. If the definition of a view must change, the view must be replaced; you cannot change the definition of a view. You can replace views in the following ways:

- You can drop and re-create the view.

Caution: When a view is dropped, all grants of corresponding object privileges are revoked from roles and users. After the view is re-created, privileges must be re-granted.

- You can redefine the view with a `CREATE VIEW` statement that contains the `OR REPLACE` option. The `OR REPLACE` option replaces the current definition of a view and preserves the current security authorizations. For example, assume that you created the `sales_staff` view as shown earlier, and, in addition, you granted several object privileges to roles and other users. However, now you need to redefine the `sales_staff` view to change the department number specified in the `WHERE` clause. You can replace the current version of the `sales_staff` view with the following statement:

```
CREATE OR REPLACE VIEW sales_staff AS
  SELECT empno, ename, deptno
  FROM emp
  WHERE deptno = 30
  WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

Before replacing a view, consider the following effects:

- Replacing a view replaces the view's definition in the data dictionary. All underlying objects referenced by the view are not affected.
- If a constraint in the `CHECK OPTION` was previously defined but not included in the new view definition, the constraint is dropped.
- All views and PL/SQL program units dependent on a replaced view become invalid (not usable). See "[Managing Object Dependencies](#)" on page 21-23 for more information on how Oracle manages such dependencies.

Managing Sequences

Sequences are database objects from which multiple users can generate unique integers. You can use sequences to automatically generate primary key values. This section describes various aspects of managing sequences, and contains the following topics:

- [Creating Sequences](#)
- [Altering Sequences](#)

- [Dropping Sequences](#)

See Also:

- *Oracle9i Database Concepts* for more information about sequences
- *Oracle9i SQL Reference* for statement syntax and information about accessing sequence numbers using `CURRVAL` and `NEXTVAL` pseudocolumns
- *Oracle9i Application Developer's Guide - Fundamentals* for information about using sequences in applications

Creating Sequences

To create a sequence in your schema, you must have the `CREATE SEQUENCE` system privilege. To create a sequence in another user's schema, you must have the `CREATE ANY SEQUENCE` privilege.

Create a sequence using the `CREATE SEQUENCE` statement. For example, the following statement creates a sequence used to generate employee numbers for the `empno` column of the `emp` table:

```
CREATE SEQUENCE emp_sequence
    INCREMENT BY 1
    START WITH 1
    NOMAXVALUE
    NOCYCLE
    CACHE 10;
```

The `CACHE` option pre-allocates a set of sequence numbers and keeps them in memory so that sequence numbers can be accessed faster. When the last of the sequence numbers in the cache has been used, Oracle reads another set of numbers into the cache.

Oracle might skip sequence numbers if you choose to cache a set of sequence numbers. For example, when an instance abnormally shuts down (for example, when an instance failure occurs or a `SHUTDOWN ABORT` statement is issued), sequence numbers that have been cached but not used are lost. Also, sequence numbers that have been used but not saved are lost as well. Oracle might also skip cached sequence numbers after an export and import. See *Oracle9i Database Utilities* for details.

See Also:

- *Oracle9i Real Application Clusters Deployment and Performance* for information about how caching sequence numbers improves performance in an Oracle Real Application Clusters environment
- *Oracle9i Application Developer's Guide - Fundamentals* for information about using sequences in your applications, and performance considerations when caching sequence numbers

Altering Sequences

To alter a sequence, your schema must contain the sequence, or you must have the `ALTER ANY SEQUENCE` system privilege. You can alter a sequence to change any of the parameters that define how it generates sequence numbers except the sequence's starting number. To change the starting point of a sequence, drop the sequence and then re-create it.

Alter a sequence using the `ALTER SEQUENCE` statement. For example, the following statement alters the `emp_sequence`:

```
ALTER SEQUENCE emp_sequence
  INCREMENT BY 10
  MAXVALUE 10000
  CYCLE
  CACHE 20;
```

Dropping Sequences

You can drop any sequence in your schema. To drop a sequence in another schema, you must have the `DROP ANY SEQUENCE` system privilege. If a sequence is no longer required, you can drop the sequence using the `DROP SEQUENCE` statement. For example, the following statement drops the `order_seq` sequence:

```
DROP SEQUENCE order_seq;
```

When a sequence is dropped, its definition is removed from the data dictionary. Any synonyms for the sequence remain, but return an error when referenced.

Managing Synonyms

A synonym is an alias for a schema object. Synonyms can provide a level of security by masking the name and owner of an object and by providing location

transparency for remote objects of a distributed database. Also, they are convenient to use and reduce the complexity of SQL statements for database users.

Synonyms allow underlying objects to be renamed or moved, where only the synonym needs to be redefined and applications based on the synonym continue to function without modification.

You can create both public and private synonyms. A **public** synonym is owned by the special user group named `PUBLIC` and is accessible to every user in a database. A **private** synonym is contained in the schema of a specific user and available only to the user and the user's grantees.

This section contains the following synonym management information:

- [Creating Synonyms](#)
- [Dropping Synonyms](#)

See Also:

- *Oracle9i Database Concepts* for more information about synonyms
- *Oracle9i SQL Reference* for statement syntax

Creating Synonyms

To create a private synonym in your own schema, you must have the `CREATE SYNONYM` privilege. To create a private synonym in another user's schema, you must have the `CREATE ANY SYNONYM` privilege. To create a public synonym, you must have the `CREATE PUBLIC SYNONYM` system privilege.

Create a synonym using the `CREATE SYNONYM` statement. The underlying schema object need not exist, nor do you need privileges to access the object. The following statement creates a public synonym named `public_emp` on the `emp` table contained in the schema of `jward`:

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp;
```

Dropping Synonyms

You can drop any private synonym in your own schema. To drop a private synonym in another user's schema, you must have the `DROP ANY SYNONYM` system privilege. To drop a public synonym, you must have the `DROP PUBLIC SYNONYM` system privilege.

Drop a synonym that is no longer required using `DROP SYNONYM` statement. To drop a private synonym, omit the `PUBLIC` keyword. To drop a public synonym, include the `PUBLIC` keyword.

For example, the following statement drops the private synonym named `emp`:

```
DROP SYNONYM emp;
```

The following statement drops the public synonym named `public_emp`:

```
DROP PUBLIC SYNONYM public_emp;
```

When you drop a synonym, its definition is removed from the data dictionary. All objects that reference a dropped synonym remain. However, they become invalid (not usable). For more information about how dropping synonyms can affect other schema objects, see "[Managing Object Dependencies](#)".

Viewing Information About Views, Synonyms, and Sequences

The following views display information about views, synonyms, and sequences:

View	Description
DBA_VIEWS ALL_VIEWS USER_VIEWS	DBA view describes all views in the database. ALL view is restricted to views accessible to the current user. USER view is restricted to views owned by the current user.
DBA_SYNONYMS ALL_SYNONYMS USER_SYNONYMS	These views describe synonyms.
DBA_SEQUENCES ALL_SEQUENCES USER_SEQUENCES	These views describe sequences.
DBA_UPDATABLE_COLUMNS ALL_UPDATABLE_COLUMNS USER_UPDATABLE_COLUMNS	These views describe all columns in join views that are updatable.

See Also: *Oracle9i Database Reference* for complete descriptions of these views

General Management of Schema Objects

This chapter describes schema object management issues that are common across multiple types of schema objects. The following topics are presented:

- [Creating Multiple Tables and Views in a Single Operation](#)
- [Renaming Schema Objects](#)
- [Analyzing Tables, Indexes, and Clusters](#)
- [Truncating Tables and Clusters](#)
- [Enabling and Disabling Triggers](#)
- [Managing Integrity Constraints](#)
- [Managing Object Dependencies](#)
- [Managing Object Name Resolution](#)
- [Changing Storage Parameters for the Data Dictionary](#)
- [Displaying Information About Schema Objects](#)

See Also: *Oracle9i SQL Reference* for more information about syntax, authorizations, and restrictions for the SQL statements discussed in this chapter

Creating Multiple Tables and Views in a Single Operation

You can create several tables and views and grant privileges in one operation using the `CREATE SCHEMA` statement. The `CREATE SCHEMA` statement is useful if you want to guarantee the creation of several tables, views, and grants in one operation. If an individual table, view or grant fails, the entire statement is rolled back. None of the objects are created, nor are the privileges granted.

Specifically, the `CREATE SCHEMA` statement can include *only* `CREATE TABLE`, `CREATE VIEW`, and `GRANT` statements. You must have the privileges necessary to issue the included statements. You are not actually creating a schema, that is done when the user is created with a `CREATE USER` statement. Rather, you are populating the schema.

The following statement creates two tables and a view that joins data from the two tables:

```
CREATE SCHEMA AUTHORIZATION scott
  CREATE TABLE dept (
    deptno NUMBER(3,0) PRIMARY KEY,
    dname VARCHAR2(15),
    loc VARCHAR2(25)
  )
  CREATE TABLE emp (
    empno NUMBER(5,0) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
    job VARCHAR2(10),
    mgr NUMBER(5,0),
    hiredate DATE DEFAULT (sysdate),
    sal NUMBER(7,2),
    comm NUMBER(7,2),
    deptno NUMBER(3,0) NOT NULL
    CONSTRAINT dept_fkey REFERENCES dept)
  CREATE VIEW sales_staff AS
    SELECT empno, ename, sal, comm
    FROM emp
    WHERE deptno = 30
    WITH CHECK OPTION CONSTRAINT sales_staff_cnst
  GRANT SELECT ON sales_staff TO human_resources;
```

The `CREATE SCHEMA` statement does not support Oracle extensions to the ANSI `CREATE TABLE` and `CREATE VIEW` statements, including the `STORAGE` clause.

Renaming Schema Objects

To rename an object, it must be in your schema. You can rename schema objects in either of the following ways:

- Drop and re-create the object
- Rename the object using the `RENAME` statement

If you drop and re-create an object, all privileges granted for that object are lost. Privileges must be regranted when the object is re-created.

Alternatively, a table, view, sequence, or a private synonym of a table, view, or sequence can be renamed using the `RENAME` statement. When using the `RENAME` statement, integrity constraints, indexes, and grants made for the object are carried forward for the new name. For example, the following statement renames the `sales_staff` view:

```
RENAME sales_staff TO dept_30;
```

Note: You cannot use `RENAME` for stored PL/SQL program unit, public synonym, index, or cluster. To rename such an object, you must drop and re-create it.

Before renaming a schema object, consider the following effects:

- All views and PL/SQL program units dependent on a renamed object become invalid, and must be recompiled before next use.
- All synonyms for a renamed object return an error when used.

See Also: ["Managing Object Dependencies"](#) on page 21-23 for more information about how Oracle manages object dependencies

Analyzing Tables, Indexes, and Clusters

You analyze a schema object (table, index, or cluster) to:

- Collect and manage statistics for it
- Verify the validity of its storage format
- Identify migrated and chained rows of a table or cluster

Note: Oracle Corporation strongly recommends that you use the `DBMS_STATS` package rather than `ANALYZE` to collect optimizer statistics. That package lets you collect statistics in parallel, collect global statistics for partitioned objects, and fine tune your statistics collection in other ways. Further, the cost-based optimizer, which depends upon statistics, will eventually use only statistics that have been collected by `DBMS_STATS`. See *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information on this package.

However, you must use the `ANALYZE` statement (rather than `DBMS_STATS`) for statistics collection not related to the cost-based optimizer, such as:

- To use the `VALIDATE` or `LIST CHAINED ROWS` clauses
 - To collect information on freelist blocks
-
-

The following topics are discussed in this section:

- [Collecting Statistics for Tables, Indexes, and Clusters](#)
- [Validating Tables, Indexes, Clusters, and Materialized Views](#)
- [Listing Chained Rows of Tables and Clusters](#)

See Also: ["Analyzing Index-Organized Tables"](#) on page 15-32 for information specific to analyzing index-organized tables

Collecting Statistics for Tables, Indexes, and Clusters

You can use the `DBMS_STATS` package or the `ANALYZE` statement to gather statistics about the physical storage characteristics of a table, index, or cluster. These statistics are stored in the data dictionary and can be used by the optimizer to choose the most efficient execution plan for SQL statements accessing analyzed objects.

Oracle recommends using the more versatile `DBMS_STATS` package for gathering optimizer statistics, but you must use the `ANALYZE` statement to collect non-optimizer statistics, such as empty blocks, average space, and so forth.

Computing Statistics Using the `DBMS_STATS` Package

The `DBMS_STATS` package allows both the gathering of statistics, including utilizing parallel execution, and the external manipulation of statistics. Statistics can

be stored in tables outside of the data dictionary, where they can be manipulated without affecting the optimizer. Statistics can be copied between databases or backup copies can be made.

The following `DBMS_STATS` procedures enable the gathering of optimizer statistics:

- `GATHER_INDEX_STATS`
- `GATHER_TABLE_STATS`
- `GATHER_SCHEMA_STATS`
- `GATHER_DATABASE_STATS`

See Also:

- *Oracle9i Database Performance Tuning Guide and Reference* for information about using `DBMS_STATS` to gather statistics for the optimizer
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of the `DBMS_STATS` package

Computing Statistics Using the `ANALYZE` Statement

The following statement computes statistics for the `emp` table:

```
ANALYZE TABLE emp COMPUTE STATISTICS;
```

The following query estimates statistics on the `emp` table, using the default statistical sample of 1064 rows:

```
ANALYZE TABLE emp ESTIMATE STATISTICS;
```

To specify the statistical sample that Oracle should use, include the `SAMPLE` option with the `ESTIMATE STATISTICS` option. You can specify an integer that indicates either a number of rows or index values, or a percentage of the rows or index values in the table. The following statements show examples of each option:

```
ANALYZE TABLE emp
  ESTIMATE STATISTICS
  SAMPLE 2000 ROWS;
```

```
ANALYZE TABLE emp
  ESTIMATE STATISTICS
  SAMPLE 33 PERCENT;
```

In either case, if you specify a percentage greater than 50, or a number of rows or index values that is greater than 50% of those in the object, Oracle computes the exact statistics, rather than estimating.

If the data dictionary currently contains statistics for the specified object when an `ANALYZE` statement is issued, the new statistics replace the old statistics in the data dictionary.

Some Optional Means of Computing Statistics

There are some PL/SQL packaged procedures that effectively allow you to execute an `ANALYZE` statement. These are:

- `DBMS_UTILITY.ANALYZE_SCHEMA`
- `DBMS_UTILITY.ANALYZE_DATABASE`
- `DBMS_DDL.ANALYZE_OBJECT`

It is recommended that these packaged procedures only be used for collecting non-optimizer statistics.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for information on the `DBMS_UTILITY` package
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for information on the `DBMS_DDL` package

Validating Tables, Indexes, Clusters, and Materialized Views

To verify the integrity of the structure of a table, index, cluster, or materialized view, use the `ANALYZE` statement with the `VALIDATE STRUCTURE` option. If the structure is valid, no error is returned. However, if the structure is corrupt, you receive an error message.

For example, in rare cases such as hardware or other system failures, an index can become corrupted and not perform correctly. When validating the index, you can confirm that every entry in the index points to the correct row of the associated table. If the index is corrupt, you can drop and re-create it.

If a table, index, or cluster is corrupt, you should drop it and re-create it. If a materialized view is corrupt, perform a complete refresh and ensure that you have remedied the problem. If the problem is not corrected, drop and re-create the materialized view.

The following statement analyzes the `emp` table:

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

You can validate an object and all related objects (for example, indexes) by including the `CASCADE` option. The following statement validates the `emp` table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

You can specify that you want to perform structure validation online while DML is occurring against the object being validated. There can be a slight performance impact when validating with ongoing DML affecting the object, but this is offset by the flexibility of being able to perform `ANALYZE` online. The following statement validates the `emp` table and all associated indexes online:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE ONLINE;
```

Listing Chained Rows of Tables and Clusters

You can look at the chained and migrated rows of a table or cluster using the `ANALYZE` statement with the `LIST CHAINED ROWS` clause. The results of this statement are stored in a specified table created explicitly to accept the information returned by the `LIST CHAINED ROWS` clause. These results are useful in determining whether you have enough room for updates to rows. For example, this information can show whether `PCTFREE` is set appropriately for the table or cluster.

Creating a `CHAINED_ROWS` Table

To create the table to accept data returned by an `ANALYZE . . . LIST CHAINED ROWS` statement, execute the `UTLCHAIN.SQL` or `UTLCHN1.SQL` script. These scripts are provided by Oracle. They create a table named `CHAINED_ROWS` in the schema of the user submitting the script.

Note: Your choice of script to execute for creating the `CHAINED_ROWS` table is dependent upon the compatibility level of your database and the type of table you are analyzing. See the *Oracle9i SQL Reference* for more information.

After a `CHAINED_ROWS` table is created, you specify it in the `INTO` clause of the `ANALYZE` statement. For example, the following statement inserts rows containing

information about the chained rows in the emp_dept cluster into the CHAINED_ROWS table:

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO CHAINED_ROWS;
```

See Also: *Oracle9i Database Reference* for a description of the CHAINED_ROWS table

Eliminating Migrated or Chained Rows in a Table

You can use the information in the CHAINED_ROWS table to reduce or eliminate migrated and chained rows in an existing table. Use the following procedure.

1. Use the ANALYZE statement to collect information about migrated and chained rows.

```
ANALYZE TABLE order_hist LIST CHAINED ROWS;
```

2. Query the output table:

```
SELECT *
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'ORDER_HIST';
```

OWNER_NAME	TABLE_NAME	CLUST...	HEAD_ROWID	TIMESTAMP
SCOTT	ORDER_HIST	...	AAAA1uAAHAAAAA1AAA	04-MAR-96
SCOTT	ORDER_HIST	...	AAAA1uAAHAAAAA1AAB	04-MAR-96
SCOTT	ORDER_HIST	...	AAAA1uAAHAAAAA1AAC	04-MAR-96

The output lists all rows that are either migrated or chained.

3. If the output table shows that you have many migrated or chained rows, then you can eliminate migrated rows by continuing through the following steps:
4. Create an intermediate table with the same columns as the existing table to hold the migrated and chained rows:

```
CREATE TABLE int_order_hist
AS SELECT *
FROM order_hist
WHERE ROWID IN
(SELECT HEAD_ROWID
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'ORDER_HIST');
```

5. Delete the migrated and chained rows from the existing table:

```
DELETE FROM order_hist
WHERE ROWID IN
  (SELECT HEAD_ROWID
   FROM CHAINED_ROWS
   WHERE TABLE_NAME = 'ORDER_HIST');
```

6. Insert the rows of the intermediate table into the existing table:

```
INSERT INTO order_hist
SELECT *
FROM int_order_hist;
```

7. Drop the intermediate table:

```
DROP TABLE int_order_history;
```

8. Delete the information collected in step 1 from the output table:

```
DELETE FROM CHAINED_ROWS
WHERE TABLE_NAME = 'ORDER_HIST';
```

9. Use the ANALYZE statement again, and query the output table.
10. Any rows that appear in the output table are chained. You can eliminate chained rows only by increasing your data block size. It might not be possible to avoid chaining in all situations. Chaining is often unavoidable with tables that have a LONG column or long CHAR or VARCHAR2 columns.

Truncating Tables and Clusters

You can delete all rows of a table or all rows in a group of clustered tables so that the table (or cluster) still exists, but is completely empty. For example, consider a table that contains monthly data, and at the end of each month, you need to empty it (delete all rows) after archiving its data.

To delete all rows from a table, you have the following options:

- Use the DELETE statement.
- Use the DROP and CREATE statements.
- Use the TRUNCATE statement.

Each of these options are discussed in the following sections

Using DELETE

You can delete the rows of a table using the `DELETE` statement. For example, the following statement deletes all rows from the `emp` table:

```
DELETE FROM emp;
```

If there are many rows present in a table or cluster when using the `DELETE` statement, significant system resources are consumed as the rows are deleted. For example, CPU time, redo log space, and rollback segment space from the table and any associated indexes require resources. Also, as each row is deleted, triggers can be fired. The space previously allocated to the resulting empty table or cluster remains associated with that object. With `DELETE` you can choose which rows to delete, whereas `TRUNCATE` and `DROP` affect the entire object.

Using DROP and CREATE

You can drop a table and then re-create the table. For example, the following statements drop and then re-create the `emp` table:

```
DROP TABLE emp;  
CREATE TABLE emp ( ... );
```

When dropping and re-creating a table or cluster, all associated indexes, integrity constraints, and triggers are also dropped, and all objects that depend on the dropped table or clustered table are invalidated. Also, all grants for the dropped table or clustered table are dropped.

Using TRUNCATE

You can delete all rows of the table using the `TRUNCATE` statement. For example, the following statement truncates the `emp` table:

```
TRUNCATE TABLE emp;
```

Using the `TRUNCATE` statement provides a fast, efficient method for deleting all rows from a table or cluster. A `TRUNCATE` statement does not generate any rollback information and it commits immediately. It is a DDL statement and cannot be rolled back. A `TRUNCATE` statement does not affect any structures associated with the table being truncated (constraints and triggers) or authorizations. A `TRUNCATE` statement also specifies whether space currently allocated for the table is returned to the containing tablespace after truncation.

You can truncate any table or cluster in your own schema. Any user who has the `DROP ANY TABLE` system privilege can truncate a table or cluster in any schema.

Before truncating a table or clustered table containing a parent key, all referencing foreign keys in different tables must be disabled. A self-referential constraint does not have to be disabled.

As a `TRUNCATE` statement deletes rows from a table, triggers associated with the table are not fired. Also, a `TRUNCATE` statement does not generate any audit information corresponding to `DELETE` statements if auditing is enabled. Instead, a single audit record is generated for the `TRUNCATE` statement being issued. See [Chapter 26, "Auditing Database Use"](#) for information about auditing.

A hash cluster cannot be truncated, nor can tables within a hash or index cluster be individually truncated. Truncation of an index cluster deletes all rows from all tables in the cluster. If all the rows must be deleted from an individual clustered table, use the `DELETE` statement or drop and re-create the table.

The `REUSE STORAGE` or `DROP STORAGE` options of the `TRUNCATE` statement control whether space currently allocated for a table or cluster is returned to the containing tablespace after truncation. The default option, `DROP STORAGE`, reduces the number of extents allocated to the resulting table to the original setting for `MINEXTENTS`. Freed extents are then returned to the system and can be used by other objects.

Alternatively, the `REUSE STORAGE` option specifies that all space currently allocated for the table or cluster remains allocated to it. For example, the following statement truncates the `emp_dept` cluster, leaving all extents previously allocated for the cluster available for subsequent inserts and deletes:

```
TRUNCATE CLUSTER emp_dept REUSE STORAGE;
```

The `REUSE` or `DROP STORAGE` option also applies to any associated indexes. When a table or cluster is truncated, all associated indexes are also truncated. The storage parameters for a truncated table, cluster, or associated indexes are not changed as a result of the truncation.

Enabling and Disabling Triggers

Database triggers are procedures that are stored in the database and activated ("fired") when specific conditions occur, such as adding a row to a table. You can use triggers to supplement the standard capabilities of Oracle to provide a highly customized database management system. For example, you can create a trigger to

restrict DML operations against a table, allowing only statements issued during regular business hours.

Database triggers can be associated with a table, schema, or database. They are implicitly fired when:

- DML statements are executed (INSERT, UPDATE, DELETE) against an associated table
- Certain DDL statements are executed (for example: ALTER, CREATE, DROP) on objects within a database or schema
- A specified database event occurs (for example: STARTUP, SHUTDOWN, SERVERERROR)

This is not a complete list. See the *Oracle9i SQL Reference* for a full list of statements and database events that cause triggers to fire

Create triggers with the `CREATE TRIGGER` statement. They can be defined as firing BEFORE or AFTER the triggering event, or INSTEAD OF it. The following statement creates a trigger `scott.emp_permit_changes` on table `scott.emp`. The trigger fires before any of the specified statements are executed.

```
CREATE TRIGGER scott.emp_permit_changes
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON scott.emp
  .
pl/sql block
  .
```

You can later remove a trigger from the database by issuing the `DROP TRIGGER` statement.

A trigger can be in either of two distinct modes:

- Enabled
An enabled trigger executes its trigger body if a triggering statement is issued and the trigger restriction, if any, evaluates to true. By default, triggers are enabled when first created.
- Disabled
A disabled trigger does not execute its trigger body, even if a triggering statement is issued and the trigger restriction (if any) evaluates to true.

To enable or disable triggers using the `ALTER TABLE` statement, you must own the table, have the `ALTER` object privilege for the table, or have the `ALTER ANY TABLE` system privilege. To enable or disable an individual trigger using the `ALTER TRIGGER` statement, you must own the trigger or have the `ALTER ANY TRIGGER` system privilege.

See Also:

- *Oracle9i Database Concepts* for a more detailed description of triggers
- *Oracle9i SQL Reference* for syntax, restrictions, and specific authorization requirements for the SQL statements used to create and manage triggers
- *Oracle9i Application Developer's Guide - Fundamentals* for information about creating and using triggers

Enabling Triggers

You enable a disabled trigger using the `ALTER TRIGGER` statement with the `ENABLE` option. To enable the disabled trigger named `reorder` on the `inventory` table, enter the following statement:

```
ALTER TRIGGER reorder ENABLE;
```

To enable all triggers defined for a specific table, use the `ALTER TABLE` statement with the `ENABLE ALL TRIGGERS` option. To enable all triggers defined for the `INVENTORY` table, enter the following statement:

```
ALTER TABLE inventory  
    ENABLE ALL TRIGGERS;
```

Disabling Triggers

Consider temporarily disabling a trigger if one of the following conditions is true:

- An object that the trigger references is not available.
- You must perform a large data load and want it to proceed quickly without firing triggers.
- You are loading data into the table to which the trigger applies.

You disable a trigger using the `ALTER TRIGGER` statement with the `DISABLE` option. To disable the trigger `reorder` on the `inventory` table, enter the following statement:

```
ALTER TRIGGER reorder DISABLE;
```

You can disable all triggers associated with a table at the same time using the `ALTER TABLE` statement with the `DISABLE ALL TRIGGERS` option. For example, to disable all triggers defined for the `inventory` table, enter the following statement:

```
ALTER TABLE inventory
  DISABLE ALL TRIGGERS;
```

Managing Integrity Constraints

Integrity constraints are rules that restrict the values for one or more columns in a table. Constraint clauses can appear in either `CREATE TABLE` or `ALTER TABLE` statements, and identify the column or columns affected by the constraint and identify the conditions of the constraint.

This section discusses the concepts of constraints and identifies the SQL statements used to define and manage integrity constraints. The following topics are contained in this section:

- [Integrity Constraint States](#)
- [Setting Integrity Constraints Upon Definition](#)
- [Modifying, Renaming, or Dropping Existing Integrity Constraints](#)
- [Deferring Constraint Checks](#)
- [Reporting Constraint Exceptions](#)
- [Viewing Constraint Information](#)

See Also:

- *Oracle9i Database Concepts* for a more thorough discussion of integrity constraints
- *Oracle9i Application Developer's Guide - Fundamentals* for detailed information and examples of using integrity constraints in applications

Integrity Constraint States

You can specify that a constraint is enabled (`ENABLE`) or disabled (`DISABLE`). If a constraint is enabled, data is checked as it is entered or updated in the database, and data that does not conform to the constraint's rule is prevented from being entered. If a constraint is disabled, then data that does not conform can be allowed to enter the database.

Additionally, you can specify that existing data in the table must conform to the constraint (`VALIDATE`). Conversely, if you specify `NOVALIDATE`, you are not ensured that existing data conforms.

An integrity constraint defined on a table can be in one of the following states:

- `ENABLE, VALIDATE`
- `ENABLE, NOVALIDATE`
- `DISABLE, VALIDATE`
- `DISABLE, NOVALIDATE`

For details about the meaning of these states and an understanding of their consequences, see the *Oracle9i SQL Reference*. Some of these consequences are discussed here.

Disabling Constraints

To enforce the rules defined by integrity constraints, the constraints should always be enabled. However, consider temporarily disabling the integrity constraints of a table for the following performance reasons:

- When loading large amounts of data into a table
- When performing batch operations that make massive changes to a table (for example, changing every employee's number by adding 1000 to the existing number)
- When importing or exporting one table at a time

In all three cases, temporarily disabling integrity constraints can improve the performance of the operation, especially in data warehouse configurations.

It is possible to enter data that violates a constraint while that constraint is disabled. Thus, you should always enable the constraint after completing any of the operations listed in the bullets above.

Enabling Constraints

While a constraint is enabled, no row violating the constraint can be inserted into the table. However, while the constraint is disabled such a row can be inserted. This row is known as an exception to the constraint. If the constraint is in the enable novalidated state, violations resulting from data entered while the constraint was disabled remain. The rows that violate the constraint must be either updated or deleted in order for the constraint to be put in the validated state.

You can identify exceptions to a specific integrity constraint while attempting to enable the constraint. See "[Reporting Constraint Exceptions](#)" on page 21-21. All rows violating constraints are noted in an EXCEPTIONS table, which you can examine.

Enable Novalidate Constraint State

When a constraint is in the enable novalidate state, all subsequent statements are checked for conformity to the constraint. However, any existing data in the table is not checked. A table with enable novalidated constraints can contain invalid data, but it is not possible to add new invalid data to it. Enabling constraints in the novalidated state is most useful in data warehouse configurations that are uploading valid OLTP data.

Enabling a constraint does not require validation. Enabling a constraint novalidate is much faster than enabling and validating a constraint. Also, validating a constraint that is already enabled does not require any DML locks during validation (unlike validating a previously disabled constraint). Enforcement guarantees that no violations are introduced during the validation. Hence, enabling without validating enables you to reduce the downtime typically associated with enabling a constraint.

Integrity Constraint States: Procedures and Benefits

Using integrity constraint states in the following order can ensure the best benefits:

1. Disable state.
2. Perform the operation (load, export, import).
3. Enable novalidate state.
4. Enable state.

Some benefits of using constraints in this order are:

- No locks are held.

- All constraints can go to enable state concurrently.
- Constraint enabling is done in parallel.
- Concurrent activity on table is permitted.

Setting Integrity Constraints Upon Definition

When an integrity constraint is defined in a `CREATE TABLE` or `ALTER TABLE` statement, it can be enabled, disabled, or validated or not validated as determined by your specification of the `ENABLE/DISABLE` clause. If the `ENABLE/DISABLE` clause is not specified in a constraint's definition, Oracle automatically enables and validates the constraint.

Disabling Constraints Upon Definition

The following `CREATE TABLE` and `ALTER TABLE` statements both define and disable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY DISABLE, . . . ;
```

```
ALTER TABLE emp
    ADD PRIMARY KEY (empno) DISABLE;
```

An `ALTER TABLE` statement that defines and disables an integrity constraint never fails because of rows in the table that violate the integrity constraint. The definition of the constraint is allowed because its rule is not enforced.

Enabling Constraints Upon Definition

The following `CREATE TABLE` and `ALTER TABLE` statements both define and enable integrity constraints:

```
CREATE TABLE emp (
    empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY, . . . ;
```

```
ALTER TABLE emp
    ADD CONSTRAINT emp.pk PRIMARY KEY (empno);
```

An `ALTER TABLE` statement that defines and attempts to enable an integrity constraint can fail because rows of the table violate the integrity constraint. If this case, the statement is rolled back and the constraint definition is not stored and not enabled.

When you enable a `UNIQUE` or `PRIMARY KEY` constraint an associated index is created.

See Also: ["Creating an Index Associated with a Constraint"](#) on page 16-11

Modifying, Renaming, or Dropping Existing Integrity Constraints

You can use the `ALTER TABLE` statement to enable, disable, modify, or drop a constraint. When Oracle is using a `UNIQUE` or `PRIMARY KEY` index to enforce a constraint, and constraints associated with that index are dropped or disabled, the index is dropped, unless you specify otherwise.

While enabled foreign keys reference a `PRIMARY` or `UNIQUE` key, you cannot disable or drop the `PRIMARY` or `UNIQUE` key constraint or the index.

Disabling Enabled Constraints

The following statements disable integrity constraints. The second statement specifies that the associated indexes are to be kept.

```
ALTER TABLE dept
  DISABLE CONSTRAINT dname_ukey;
```

```
ALTER TABLE dept
  DISABLE PRIMARY KEY KEEP INDEX,
  DISABLE UNIQUE (dname, loc) KEEP INDEX;
```

The following statements enable novalidate disabled integrity constraints:

```
ALTER TABLE dept
  ENABLE NOVALIDATE CONSTRAINT dname_ukey;
```

```
ALTER TABLE dept
  ENABLE NOVALIDATE PRIMARY KEY,
  ENABLE NOVALIDATE UNIQUE (dname, loc);
```

The following statements enable or validate disabled integrity constraints:

```
ALTER TABLE dept
  MODIFY CONSTRAINT dname_key VALIDATE;
ALTER TABLE dept
  MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

The following statements enable disabled integrity constraints:

```
ALTER TABLE dept
    ENABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
    ENABLE PRIMARY KEY,
    ENABLE UNIQUE (dname, loc);
```

To disable or drop a `UNIQUE` key or `PRIMARY KEY` constraint and all dependent `FOREIGN KEY` constraints in a single step, use the `CASCADE` option of the `DISABLE` or `DROP` clauses. For example, the following statement disables a `PRIMARY KEY` constraint and any `FOREIGN KEY` constraints that depend on it:

```
ALTER TABLE dept
    DISABLE PRIMARY KEY CASCADE;
```

Renaming Constraints

The `ALTER TABLE ... RENAME CONSTRAINT` statement enables you to rename any currently existing constraint for a table. The new constraint name must not conflict with any existing constraint names for a user.

The following statement renames the `dname_ukey` constraint for table `dept`:

```
ALTER TABLE dept
    RENAME CONSTRAINT dname_ukey TO dname_unikey;
```

When you rename a constraint, all dependencies on the base table remain valid.

The `RENAME CONSTRAINT` clause provides a means of renaming system generated constraint names.

Dropping Constraints

You can drop an integrity constraint if the rule that it enforces is no longer true, or if the constraint is no longer needed. You can drop the constraint using the `ALTER TABLE` statement with one of the following clauses:

- `DROP PRIMARY KEY`
- `DROP UNIQUE`
- `DROP CONSTRAINT`

The following two statements drop integrity constraints. The second statement keeps the index associated with the `PRIMARY KEY` constraint:

```
ALTER TABLE dept
    DROP UNIQUE (dname, loc);
```

```
ALTER TABLE emp
  DROP PRIMARY KEY KEEP INDEX,
  DROP CONSTRAINT dept_fkey;
```

If FOREIGN KEYS reference a UNIQUE or PRIMARY KEY, you must include the CASCADE CONSTRAINTS clause in the DROP statement, or you cannot drop the constraint.

Deferring Constraint Checks

When Oracle checks a constraint, it signals an error if the constraint is not satisfied. You can defer checking the validity of constraints until the end of a transaction.

When you issue the SET CONSTRAINTS statement, the SET CONSTRAINTS mode lasts for the duration of the transaction, or until another SET CONSTRAINTS statement resets the mode.

Notes:

- You cannot issue a SET CONSTRAINT statement inside a trigger.
 - Deferrable unique and primary keys must use nonunique indexes.
-
-

Set All Constraints Deferred

Within the application being used to manipulate the data, you must set all constraints deferred before you actually begin processing any data. Use the following DML statement to set all deferrable constraints deferred:

```
SET CONSTRAINTS ALL DEFERRED;
```

Note: The SET CONSTRAINTS statement applies only to the current transaction. The defaults specified when you create a constraint remain as long as the constraint exists. The ALTER SESSION SET CONSTRAINTS statement applies for the current session only.

Check the Commit (Optional)

You can check for constraint violations before committing by issuing the SET CONSTRAINTS ALL IMMEDIATE statement just before issuing the COMMIT. If there

are any problems with a constraint, this statement fails and the constraint causing the error is identified. If you commit while constraints are violated, the transaction is rolled back and you receive an error message.

Reporting Constraint Exceptions

If exceptions exist when a constraint is validated, an error is returned and the integrity constraint remains novalidated. When a statement is not successfully executed because integrity constraint exceptions exist, the statement is rolled back. If exceptions exist, you cannot validate the constraint until all exceptions to the constraint are either updated or deleted.

You cannot use the `CREATE TABLE` statement to determine which rows are in violation. To determine which rows violate the integrity constraint, issue the `ALTER TABLE` statement with the `EXCEPTIONS` option in the `ENABLE` clause. The `EXCEPTIONS` option places the `ROWID`, table owner, table name, and constraint name of all exception rows into a specified table.

You must create an appropriate exceptions report table to accept information from the `EXCEPTIONS` option of the `ENABLE` clause before enabling the constraint. You can create an exception table by executing the `UTLEXCPT.SQL` script or the `UTLEXPT1.SQL` script.

Note: Your choice of script to execute for creating the `EXCEPTIONS` table is dependent upon the compatibility level of your database and the type of table you are analyzing. See the *Oracle9i SQL Reference* for more information.

Both of these scripts create a table named `EXCEPTIONS`. You can create additional exceptions tables with different names by modifying and resubmitting the script.

The following statement attempts to validate the `PRIMARY KEY` of the `dept` table, and if exceptions exist, information is inserted into a table named `EXCEPTIONS`:

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO EXCEPTIONS;
```

If duplicate primary key values exist in the `dept` table and the name of the `PRIMARY KEY` constraint on `dept` is `sys_c00610`, the following rows might be placed in the table `EXCEPTIONS` by the previous statement:

```
SELECT * FROM EXCEPTIONS;
```

ROWID	OWNER	TABLE_NAME	CONSTRAINT
-------	-------	------------	------------

```

-----
AAAAZ9AABAAABvqAAB SCOTT      DEPT      SYS_C00610
AAAAZ9AABAAABvqAAG SCOTT      DEPT      SYS_C00610

```

A more informative query would be to join the rows in an exception report table and the master table to list the actual rows that violate a specific constraint, as shown in the following example:

```

SELECT deptno, dname, loc FROM dept, EXCEPTIONS
       WHERE EXCEPTIONS.constraint = 'SYS_C00610'
       AND dept.rowid = EXCEPTIONS.row_id;

```

```

DEPTNO      DNAME          LOC
-----
10          ACCOUNTING     NEW YORK
10          RESEARCH       DALLAS

```

All rows that violate a constraint must be either updated or deleted from the table containing the constraint. When updating exceptions, you must change the value violating the constraint to a value consistent with the constraint or to a null. After the row in the master table is updated or deleted, the corresponding rows for the exception in the exception report table should be deleted to avoid confusion with later exception reports. The statements that update the master table and the exception report table should be in the same transaction to ensure transaction consistency.

To correct the exceptions in the previous examples, you might issue the following transaction:

```

UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM EXCEPTIONS WHERE constraint = 'SYS_C00610';
COMMIT;

```

When managing exceptions, the goal is to eliminate all exceptions in your exception report table.

Note: While you are correcting current exceptions for a table with the constraint disabled, it is possible for other users to issue statements creating new exceptions. You can avoid this by enable novalidating the constraint before you start eliminating exceptions.

See Also: *Oracle9i Database Reference* for a description of the EXCEPTIONS table

Viewing Constraint Information

Oracle provides the following views that enable you to see constraint definitions on tables and to identify columns that are specified in constraints:

View	Description
DBA_CONSTRAINTS ALL_CONSTRAINTS USER_CONSTRAINTS	DBA view describes all constraint definitions in the database. ALL view describes constraint definitions accessible to current user. USER view describes constraint definitions owned by the current user.
DBA_CONS_COLUMNS ALL_CONS_COLUMNS USER_CONS_COLUMNS	DBA view describes all columns in the database that are specified in constraints. ALL view describes only those columns accessible to current user that are specified in constraints. USER view describes only those columns owned by the current user that are specified in constraints.

See Also: *Oracle9i Database Reference* contains descriptions of the columns in these views

Managing Object Dependencies

This section describes the various object dependencies, and contains the following topics:

- [Manually Recompiling Views](#)
- [Manually Recompiling Procedures and Functions](#)
- [Manually Recompiling Packages](#)

First, review [Table 21–1](#), which shows how objects are affected by changes to other objects on which they depend.

Table 21–1 Operations that Affect Object Status

Operation	Resulting Status of Object	Resulting Status of Dependent Objects
CREATE [TABLE SEQUENCE SYNONYM]	VALID if there are no errors	No change ¹
ALTER TABLE (ADD, RENAME, MODIFY columns) RENAME [TABLE SEQUENCE SYNONYM VIEW]	VALID if there no errors	INVALID

Table 21–1 Operations that Affect Object Status (Cont.)

Operation	Resulting Status of Object	Resulting Status of Dependent Objects
DROP [TABLE SEQUENCE SYNONYM VIEW PROCEDURE FUNCTION PACKAGE]	None. The object is dropped.	INVALID
CREATE [VIEW PROCEDURE] ²	VALID if there are no errors; INVALID if there are syntax or authorization errors	No change ¹
CREATE OR REPLACE [VIEW PROCEDURE] ²	VALID if there are no errors; INVALID if there are syntax or authorization errors	INVALID
REVOKE <i>object privilege</i> ³ ON <i>object</i> TO FROM <i>user</i>	No change	All objects of user that depend on object are INVALID ³
REVOKE <i>object privilege</i> ³ ON <i>object</i> TO FROM PUBLIC	No change	All objects in the database that depend on object are INVALID ³
REVOKE <i>system privilege</i> ⁴ TO FROM <i>user</i>	No change	All objects of user are INVALID ⁴
REVOKE <i>system privilege</i> ⁴ TO FROM PUBLIC	No change	All objects in the database are INVALID ⁴
¹ Can cause dependent objects to be made INVALID, if object did not exist earlier. ² Standalone procedures and functions, packages, and triggers. ³ Only DML object privileges, including SELECT, INSERT, UPDATE, DELETE, and EXECUTE; revalidation does not require recompiling. ⁴ Only DML system privileges, including SELECT, INSERT, UPDATE, DELETE ANY TABLE, and EXECUTE ANY PROCEDURE; revalidation does not require recompiling.		

Oracle automatically recompiles an invalid view or PL/SQL program unit the next time it is used. In addition, a user can force Oracle to recompile a view or program unit using the appropriate SQL statement with the `COMPILE` clause. Forced compilations are most often used to test for errors when a dependent view or

program unit is invalid, but is not currently being used. In these cases, automatic recompilation would not otherwise occur until the view or program unit was executed. To identify invalid dependent objects, query the views `USER/ALL/DBA_OBJECTS`.

Manually Recompiling Views

To recompile a view manually, you must have the `ALTER ANY TABLE` system privilege or the view must be contained in your schema. Use the `ALTER VIEW` statement with the `COMPILE` clause to recompile a view. The following statement recompiles the view `emp_dept` contained in your schema:

```
ALTER VIEW emp_dept COMPILE;
```

Manually Recompiling Procedures and Functions

To recompile a standalone procedure manually, you must have the `ALTER ANY PROCEDURE` system privilege or the procedure must be contained in your schema. Use the `ALTER PROCEDURE/FUNCTION` statement with the `COMPILE` clause to recompile a standalone procedure or function. The following statement recompiles the stored procedure `update_salary` contained in your schema:

```
ALTER PROCEDURE update_salary COMPILE;
```

Manually Recompiling Packages

To recompile a package manually, you must have the `ALTER ANY PROCEDURE` system privilege or the package must be contained in your schema. Use the `ALTER PACKAGE` statement with the `COMPILE` clause to recompile either a package body or both a package specification and body. The following statement recompiles just the body of the package `acct_mgmt`:

```
ALTER PACKAGE acct_mgmt COMPILE BODY;
```

The next example compiles both the body and specification of the package `acct_mgmt`:

```
ALTER PACKAGE acct_mgmt COMPILE PACKAGE;
```

Managing Object Name Resolution

Object names referenced in SQL statements can consist of several pieces, separated by periods. The following describes how Oracle resolves an object name.

1. Oracle attempts to qualify the first piece of the name referenced in the SQL statement. For example, in `scott.emp`, `scott` is the first piece. If there is only one piece, the one piece is considered the first piece.
 - a. In the current schema, Oracle searches for an object whose name matches the first piece of the object name. If it does not find such an object, it continues with Step b.
 - b. Oracle searches for a public synonym that matches the first piece of the name. If it does not find one, it continues with Step c.
 - c. Oracle searches for a schema whose name matches the first piece of the object name. If it finds one, it returns to Step b, now using the second piece of the name as the object to find in the qualified schema. If the second piece does not correspond to an object in the previously qualified schema or there is not a second piece, Oracle returns an error.

If no schema is found in Step c, the object cannot be qualified and Oracle returns an error.

2. A schema object has been qualified. Any remaining pieces of the name must match a valid part of the found object. For example, if `scott.emp.deptno` is the name, `scott` is qualified as a schema, `emp` is qualified as a table, and `deptno` must correspond to a column (because `emp` is a table). If `emp` is qualified as a package, `deptno` must correspond to a public constant, variable, procedure, or function of that package.

When global object names are used in a distributed database, either explicitly or indirectly within a synonym, the local Oracle resolves the reference locally. For example, it resolves a synonym to a remote table's global object name. The partially resolved statement is shipped to the remote database, and the remote Oracle completes the resolution of the object as described here.

Because of how Oracle resolves references, it is possible for an object to depend on the nonexistence of other objects. This situation occurs when the dependent object uses a reference that would be interpreted differently were another object present. For example, assume the following:

- At the current point in time, the `company` schema contains a table named `emp`.
- A `PUBLIC` synonym named `emp` is created for `company.emp` and the `SELECT` privilege for `company.emp` is granted to the `PUBLIC` role.
- The `jward` schema does not contain a table or private synonym named `emp`.
- The user `jward` creates a view in his schema with the following statement:

```
CREATE VIEW dept_salaries AS
  SELECT deptno, MIN(sal), AVG(sal), MAX(sal) FROM emp
  GROUP BY deptno
  ORDER BY deptno;
```

When `jward` creates the `dept_salaries` view, the reference to `emp` is resolved by first looking for `jward.emp` as a table, view, or private synonym, none of which is found, and then as a public synonym named `emp`, which is found. As a result, Oracle notes that `jward.dept_salaries` depends on the nonexistence of `jward.emp` and on the existence of `public.emp`.

Now assume that `jward` decides to create a new view named `emp` in his schema using the following statement:

```
CREATE VIEW emp AS
  SELECT empno, ename, mgr, deptno
  FROM company.emp;
```

Notice that `jward.emp` does not have the same structure as `company.emp`.

As it attempts to resolve references in object definitions, Oracle internally makes note of dependencies that the new dependent object has on "nonexistent" objects--schema objects that, if they existed, would change the interpretation of the object's definition. Such dependencies must be noted in case a nonexistent object is later created. If a nonexistent object is created, all dependent objects must be invalidated so that dependent objects can be recompiled and verified and all dependent function-based indexes must be marked unusable.

Therefore, in the previous example, as `jward.emp` is created, `jward.dept_salaries` is invalidated because it depends on `jward.emp`. Then when `jward.dept_salaries` is used, Oracle attempts to recompile the view. As Oracle resolves the reference to `emp`, it finds `jward.emp` (`public.emp` is no longer the referenced object). Because `jward.emp` does not have a `sal` column, Oracle finds errors when replacing the view, leaving it invalid.

In summary, you must manage dependencies on nonexistent objects checked during object resolution in case the nonexistent object is later created.

Changing Storage Parameters for the Data Dictionary

If your database is very large or contains an unusually large number of objects, columns in tables, constraint definitions, users, or other definitions, the tables that make up the data dictionary might at some point be unable to acquire additional extents. For example, a data dictionary table could require an additional extent, but

there is not enough contiguous space in the `SYSTEM` tablespace. If this happens, you cannot create new objects, even though the tablespace intended to hold the objects has sufficient space. To remedy this situation, you can change the storage parameters of the underlying data dictionary tables, just as you can change the storage settings for user-created segments. This allows the data dictionary tables to be allocated more extents. For example, you can adjust the values of `NEXT` or `PCTINCREASE` for a data dictionary table.

Caution: Exercise caution when changing the storage settings for the data dictionary objects. If you choose inappropriate settings, you could damage the structure of the data dictionary and be forced to re-create your entire database. For example, if you set `PCTINCREASE` for the data dictionary table `USER$` to 0 and `NEXT` to 2K, that table will quickly reach the maximum number of extents for a segment. At that point you will not be able to create any more users or roles without exporting, re-creating, and importing the entire database.

This section describes aspects of changing data dictionary storage parameters, and contains the following topics:

- [Structures in the Data Dictionary](#)
- [Errors that Require Changing Data Dictionary Storage](#)

Structures in the Data Dictionary

The following tables and clusters contain the definitions of all the user-created objects in the database:

Table or Cluster	Contains definitions for:
SEG\$	Segments defined in the database (including temporary segments)
OBJ\$	User-defined objects in the database (including clustered tables); indexed by <code>I_OBJ1</code> and <code>I_OBJ2</code>
UNDO\$	Rollback segments defined in the database; indexed by <code>I_UNDO1</code>
FET\$	Available free extents not allocated to any segment
UET\$	Extents allocated to segments
TS\$	Tablespaces defined in the database

Table or Cluster	Contains definitions for:
FILE\$	Files that make up the database; indexed by I_FILE1
FILEXT\$	Datafiles with the AUTOEXTEND option set on
TAB\$	Tables defined in the database (includes clustered tables); indexed by I_TAB1
CLU\$	Clusters defined in the database
IND\$	Indexes defined in the database; indexed by I_IND1
ICOL\$	Columns that have indexes defined on them (includes individual entries for each column in a composite index); indexed by I_ICOL1
COL\$	Columns defined in tables in the database; indexed by I_COL1 and I_COL2
CON\$	Constraints defined in the database (includes information on constraint owner); indexed by I_CON1 and I_CON2
CDEF\$	Definitions of constraints in CON\$; indexed by I_CDEF1, I_CDEF2, and I_CDEF3
CCOL\$	Columns that have constraints defined on them (includes individual entries for each column in a composite key); indexed by I_CCOL1
USER\$	Users and roles defined in the database; indexed by I_USER1
TSQ\$	Tablespace quotas for users (contains one entry for each tablespace quota defined for each user)
C_OBJ#	Cluster containing TAB\$, CLU\$, ICOL\$, IND\$, and COL\$: indexed by I_OBJ#
C_TS#	Cluster containing FET\$, TS\$, and FILE\$; indexed by I_TS#
C_USER#	Cluster containing USER and TSQ\$\$; indexed by I_USER#
C_COBJ#	Cluster containing CDEF\$ and CCOL\$; indexed by I_COBJ#

Of all of the data dictionary segments, the following are the most likely to require change:

Table or Cluster	Comments
C_TS#	If the free space in your database is very fragmented
C_OBJ#	If you have many indexes or many columns in your tables
CON\$, C_COBJ#	If you use integrity constraints heavily

Table or Cluster	Comments
C_USER#	If you have a large number of users defined in your database

For the clustered tables, you must change the storage settings for the cluster, not for the table.

Errors that Require Changing Data Dictionary Storage

Oracle returns an error if a user tries to create a new object that requires Oracle to allocate an additional extent to the data dictionary when it is unable to allocate an extent. The following error message indicates this kind of problem.

```
ORA-1653 unable to extend table name by num in tablespace name
```

If you receive this error message and the segment you were trying to change (such as a table or rollback segment) has not reached the limits specified for it in its definition, check the storage settings for the object that contains its definition.

For example, if you received an ORA-1653 while trying to define a new PRIMARY KEY constraint on a table and there is sufficient space for the index that Oracle must create for the key, check if CON\$ or C_COBJ# cannot be allocated another extent. To do this, query DBA_SEGMENTS. If another extent cannot be allocated, consider changing the storage parameters for CON\$ or C_COBJ#. See ["Example 7: Displaying Segments that Cannot Allocate Additional Extents"](#) on page 21-36.

Displaying Information About Schema Objects

Oracle provides data dictionary views and PL/SQL packages that allow you to display information about schema objects. Views and packages that are unique to a particular schema object are described in the chapter of this book associated with that object. This section describes views and packages that are generic in nature and apply to multiple schema objects.

Using PL/SQL Packages to Display Information About Schema Objects

These Oracle supplied PL/SQL packages provide information about schema objects:

Package and Procedure/Function	Description
DBMS_METADATA.GET_DDL	Use to obtain metadata (in the form of DDL used to create the object) about a schema object.
The following package procedures provide information about space usage and free blocks in schema objects:	
DBMS_SPACE.UNUSED_SPACE	Returns information about unused space in an object (table, index, or cluster).
DBMS_SPACE.FREE_BLOCKS	Returns information about free data blocks in an object (table, index, or cluster) whose segment free space is managed by free lists (segment space management is MANUAL).
DBMS_SPACE.SPACE_USAGE	Returns information about free data blocks in an object (table, index, or cluster) whose segment space management is AUTO.

The following sections contain examples of using some of these packages.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of PL/SQL packages

Example 1: Using the DBMS_METADATA Package

The DBMS_METADATA package is a powerful tool for obtaining the complete definition of a schema object. It enables you to obtain all of the attributes of an object in one pass. The object is described as DDL that can be used to (re)create it.

In this example the GET_DDL function is used to fetch the DDL for all tables in the current schema, filtering out nested tables and overflow segments. The SET_TRANSFORM_PARAM (with the handle value equal to DBMS_METADATA.SESSION_TRANSFORM meaning "for the current session") is used to specify that storage clauses are not to be returned in the SQL DDL. Afterwards, the session-level transform parameters are reset to their defaults. Once set, transform parameter values remain in effect until specifically reset to their defaults.

```
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
    DBMS_METADATA.SESSION_TRANSFORM, 'STORAGE', false);
SELECT DBMS_METADATA.GET_DDL('TABLE', u.table_name)
FROM USER_ALL_TABLES u
WHERE u.nested='NO'
AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
    DBMS_METADATA.SESSION_TRANSFORM, 'DEFAULT');
```

See Also: *Oracle9i XML Developer's Kits Guide - XDK* for detailed information and further examples relating to the use of the DBMS_METADATA package

Example 2: Using DBMS_SPACE.UNUSED_SPACE

The following SQL*Plus example uses the DBMS_SPACE package to obtain unused space information.

```
SQL> VARIABLE total_blocks NUMBER
SQL> VARIABLE total_bytes NUMBER
SQL> VARIABLE unused_blocks NUMBER
SQL> VARIABLE unused_bytes NUMBER
SQL> VARIABLE lastextf NUMBER
SQL> VARIABLE last_extb NUMBER
SQL> exec DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, -
> :total_bytes, :unused_blocks, :unused_bytes, :lastextf, -
> :last_extb, :lastusedblock);
```

PL/SQL procedure successfully completed.

```
SQL> PRINT

TOTAL_BLOCKS
-----
              5

TOTAL_BYTES
-----
            10240

...

LASTUSEDBLOCK
-----
              3
```

Using Views to Display Information About Schema Objects

These views display information about schema objects:

View	Description
DBA_OBJECTS ALL_OBJECTS USER_OBJECTS	DBA view describes all schema objects in the database. ALL view describes objects accessible to current user. USER view describes objects owned by the current user.
DBA_CATALOG ALL_CATALOG USER_CATALOG	List the name, type, and owner (USER view does not display owner) for all tables, views, synonyms, and sequences in the database.
DBA_DEPENDENCIES ALL_DEPENDENCIES USER_DEPENDENCIES	Describe all dependencies between procedures, packages, functions, package bodies, and triggers, including dependencies on views without any database links.
The following views contain information about segments of the database:	
DBA_SEGMENTS USER_SEGMENTS	Describe storage allocated for all database segments, or for segments for the current user.
The following views contain information about extents of the database:	
DBA_EXTENTS USER_EXTENTS	Describe extents comprising all segments either in the database, or segments for the current user.
DBA_FREE_SPACE USER_FREE_SPACE	List free extents in all tablespaces, or tablespaces owned by the current user.

The following sections contain examples of using some of these views.

See Also: *Oracle9i Database Reference* for a complete description of data dictionary views

Example 1: Displaying Schema Objects By Type

The following query lists all of the objects owned by the user issuing the query:

```
SELECT OBJECT_NAME, OBJECT_TYPE
       FROM USER_OBJECTS;
```

```
OBJECT_NAME          OBJECT_TYPE
-----
EMP_DEPT             CLUSTER
EMP                  TABLE
```

```

DEPT                TABLE
EMP_DEPT_INDEX      INDEX
PUBLIC_EMP           SYNONYM
EMP_MGR              VIEW
    
```

Example 2: Displaying Column Information

Column information, such as name, datatype, length, precision, scale, and default data values can be listed using one of the views ending with the `_COLUMNS` suffix. For example, the following query lists all of the default column values for the `emp` and `dept` tables:

```

SELECT TABLE_NAME, COLUMN_NAME, DATA_DEFAULT
FROM USER_TAB_COLUMNS
WHERE TABLE_NAME = 'DEPT' OR TABLE_NAME = 'EMP';
    
```

TABLE_NAME	COLUMN_NAME	DATA_DEFAULT
DEPT	DEPTNO	
DEPT	DNAME	
DEPT	LOC	'NEW YORK'
EMP	EMPNO	
EMP	ENAME	
EMP	JOB	
EMP	MGR	
EMP	HIREDATE	SYSDATE
EMP	SAL	
EMP	COMM	
EMP	DEPTNO	

Notice that not all columns have user-specified defaults. These columns automatically have `NULL` as the default.

Example 3: Displaying Dependencies of Views and Synonyms

When you create a view or a synonym, the view or synonym is based on its underlying base object. The `ALL/USER/DBA_DEPENDENCIES` data dictionary views can be used to reveal the dependencies for a view. The `ALL/USER/DBA_SYNONYMS` data dictionary views can be used to list the base object of a synonym. For example, the following query lists the base objects for the synonyms created by the user `jward`:

```

SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
FROM DBA_SYNONYMS
WHERE OWNER = 'JWARD';
    
```

TABLE_OWNER	TABLE_NAME	SYNONYM_NAME
SCOTT	DEPT	DEPT
SCOTT	EMP	EMP

Example 4: Displaying General Segment Information

The following query returns the name of each rollback segment, the tablespace that contains each, and the size of each rollback segment:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
FROM DBA_SEGMENTS
WHERE SEGMENT_TYPE = 'ROLLBACK';
```

SEGMENT_NAME	TABLESPACE_NAME	BYTES	BLOCKS	EXTENTS
RS1	SYSTEM	20480	10	2
RS2	TS1	40960	20	3
SYSTEM	SYSTEM	184320	90	3

Example 5: Displaying General Extent Information

General information about the currently allocated extents in a database is stored in the DBA_EXTENTS data dictionary view. For example, the following query identifies the extents associated with rollback segments and the size of each of those extents:

```
SELECT SEGMENT_NAME, BYTES, BLOCKS
FROM DBA_EXTENTS
WHERE SEGMENT_TYPE = 'ROLLBACK';
```

SEGMENT_NAME	BYTES	BLOCKS
RS1	10240	5
RS1	10240	5
SYSTEM	51200	25
SYSTEM	51200	25
SYSTEM	51200	25

Notice that the `rs1` rollback segment is comprised of two extents, both 10K, while the `SYSTEM` rollback segment is comprised of three equally sized extents of 50K.

Example 6: Displaying the Free Space (Extents) of a Database

Information about the free extents (extents not allocated to any segment) in a database is stored in the `DBA_FREE_SPACE` data dictionary view. For example, the following query reveals the amount of free space available as free extents in each tablespace:

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
       FROM DBA_FREE_SPACE;
```

TABLESPACE_NAME	FILE_ID	BYTES	BLOCKS
SYSTEM	1	8120320	3965
SYSTEM	1	10240	5
TS1	2	10432512	5094

Example 7: Displaying Segments that Cannot Allocate Additional Extents

You can also use `DBA_FREE_SPACE`, in combination with the views `DBA_SEGMENTS`, `DBA_TABLES`, `DBA_CLUSTERS`, `DBA_INDEXES`, and `DBA_ROLLBACK_SEGS`, to determine if any other segment is unable to allocate additional extents for data dictionary objects only.

A segment may not be allocated to an extent for any of the following reasons:

- The tablespace containing the segment does not have enough room for the next extent.
- The segment has the maximum number of extents, as recorded in the data dictionary (in `SEG.MAX_EXTENTS`).
- The segment has the maximum number of extents allowed by the data block size, which is operating system specific.

Note: While the `STORAGE` clause value for `MAXEXTENTS` can be `UNLIMITED`, data dictionary tables cannot have `MAXEXTENTS` greater than the allowed block maximum. Thus, data dictionary tables cannot be converted to unlimited format.

The following query returns the names, owners, and tablespaces of all segments that satisfy any of the above criteria:

```
SELECT a.SEGMENT_NAME, a.SEGMENT_TYPE, a.TABLESPACE_NAME, a.OWNER
       FROM DBA_SEGMENTS a
       WHERE a.NEXT_EXTENT >= (SELECT MAX(b.BYTES)
```

```
FROM DBA_FREE_SPACE b
WHERE b.TABLESPACE_NAME = a.TABLESPACE_NAME)
OR a.EXTENTS = a.MAX_EXTENTS
OR a.EXTENTS = 'data_block_size' ;
```

Note: When you use this query, replace *data_block_size* with the data block size for your system.

Once you have identified a segment that cannot allocate additional extents, you can solve the problem in either of two ways, depending on its cause:

- If the tablespace is full, add datafiles to the tablespace.
- If the segment has too many extents, and you cannot increase `MAXEXTENTS` for the segment, perform the following steps.
 1. Export the data in the segment
 2. Drop and re-create the segment, giving it a larger `INITIAL` setting so that it does not need to allocate so many extents
 3. Import the data back into the segment.

Detecting and Repairing Data Block Corruption

This chapter explains using the `DBMS_REPAIR` PL/SQL package to repair data block corruption in database schema objects. It contains the following topics:

- [Options for Repairing Data Block Corruption](#)
- [About the `DBMS_REPAIR` Package](#)
- [Using the `DBMS_REPAIR` Package](#)
- [`DBMS_REPAIR` Examples](#)

Note: If you are not familiar with the `DBMS_REPAIR` package, it is recommended that you work with an Oracle Support Services analyst when performing any of the repair procedures included in this package.

Options for Repairing Data Block Corruption

Oracle provides different methods for detecting and correcting data block corruption. One method of correction is to drop and re-create an object after the corruption is detected. However, this is not always possible or desirable. If data block corruption is limited to a subset of rows, another option is to rebuild the table by selecting all data except for the corrupt rows.

Yet another way to manage data block corruption is to use the `DBMS_REPAIR` package. You can use `DBMS_REPAIR` to detect and repair corrupt blocks in tables and indexes. Using this approach, you can address corruptions where possible, and also continue to use objects while you attempt to rebuild or repair them.

Note: Any corruption that involves the loss of data requires analysis and understanding of how that data fits into the overall database system. `DBMS_REPAIR` is not a magic wand—you must still determine whether the repair approach provided by this package is the appropriate tool for each specific corruption problem. Depending on the nature of the repair, you might lose data and logical inconsistencies can be introduced. Thus, you must weigh the gains and losses associated with using `DBMS_REPAIR`.

About the `DBMS_REPAIR` Package

This section describes the `DBMS_REPAIR` procedures contained in the package and notes some limitations and restrictions on their use.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information on the syntax, restrictions, and exceptions for the `DBMS_REPAIR` procedures

`DBMS_REPAIR` Procedures

The following table lists the procedures included in the `DBMS_REPAIR` package.

Procedure Name	Description
<code>CHECK_OBJECT</code>	Detects and reports corruptions in a table or index
<code>FIX_CORRUPT_BLOCKS</code>	Marks blocks (that were previously identified by the <code>CHECK_OBJECT</code> procedure) as software corrupt

Procedure Name	Description
DUMP_ORPHAN_KEYS	Reports index entries (into an orphan key table) that point to rows in corrupt data blocks
REBUILD_FREELISTS	Rebuilds an object's free lists
SEGMENT_FIX_STATUS	Provides the capability to fix the corrupted state of a bitmap entry when segment space management is AUTO
SKIP_CORRUPT_BLOCKS	When used, ignores blocks marked corrupt during table and index scans. If not used, you get error ORA-1578 when encountering blocks marked corrupt.
ADMIN_TABLES	Provides administrative functions (create, drop, purge) for repair or orphan key tables. Note: These tables are always created in the SYS schema.

These procedures are further described, with examples of their use, in "[DBMS_REPAIR Examples](#)" on page 22-8.

Limitations and Restrictions

DBMS_REPAIR procedures have the following limitations:

- Tables with LOBs, nested tables, and varrays are supported, but the out of line columns are ignored.
- Clusters are supported in the SKIP_CORRUPT_BLOCKS and REBUILD_FREELISTS procedures, but not in the CHECK_OBJECT procedure.
- Index-organized tables and LOB indexes are not supported.
- The DUMP_ORPHAN_KEYS procedure does not operate on bitmap indexes or function-based indexes.
- The DUMP_ORPHAN_KEYS procedure processes keys that are, at most, 3,950 bytes long.

Using the DBMS_REPAIR Package

The following approach is recommended when considering DBMS_REPAIR for addressing data block corruption:

[Task 1: Detect and Report Corruptions](#)

[Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR](#)

Task 3: Make Objects Usable

Task 4: Repair Corruptions and Rebuild Lost Data

These tasks are discussed in succeeding sections.

Task 1: Detect and Report Corruptions

Your first task, before using `DBMS_REPAIR`, should be the detection and reporting of corruptions. Reporting not only indicates what is wrong with a block, but also identifies the associated repair directive. You have several options, in addition to `DBMS_REPAIR`, for detecting corruptions. [Table 22–1](#) describes the different detection methodologies.

Table 22–1 Comparison of Corruption Detection Methods

Detection Method	Description
<code>DBMS_REPAIR</code>	Performs block checking for a specified table, partition, or index. Populates a repair table with results.
<code>DB_VERIFY</code>	External command-line utility that performs block checking on an offline database.
<code>ANALYZE</code>	Used with the <code>VALIDATE STRUCTURE</code> option, verifies the integrity of the structure of an index, table, or cluster; checks or verifies that your tables and indexes are in sync.
<code>DB_BLOCK_CHECKING</code>	Performed when the initialization parameter <code>DB_BLOCK_CHECKING=TRUE</code> . Identifies corrupt blocks before they actually are marked corrupt. Checks are performed when changes are made to a block.

DBMS_REPAIR: Using the CHECK_OBJECT and ADMIN_TABLES Procedures

The `CHECK_OBJECT` procedure checks and reports block corruptions for a specified object. Similar to the `ANALYZE . . . VALIDATE STRUCTURE` statement for indexes and tables, block checking is performed for index and data blocks.

Not only does `CHECK_OBJECT` report corruptions, but it also identifies any fixes that would occur if `FIX_CORRUPT_BLOCKS` is subsequently run on the object. This information is made available by populating a repair table, which must first be created by the `ADMIN_TABLES` procedure.

After you run the `CHECK_OBJECT` procedure, a simple query on the repair table shows the corruptions and repair directives for the object. With this information, you can assess how best to address the problems reported.

DB_VERIFY: Performing an Offline Database Check

Typically, you use `DB_VERIFY` as an offline diagnostic utility when you encounter data corruption problems.

See Also: *Oracle9i Database Utilities* for more information about `DB_VERIFY`

ANALYZE: Corruption Reporting

The `ANALYZE TABLE ... VALIDATE STRUCTURE` statement validates the structure of the analyzed object. If Oracle successfully validates the structure, a message confirming its validation is returned to you. If Oracle encounters corruption in the structure of the object, an error message is returned to you. In this case, drop and re-create the object.

See Also: *Oracle9i SQL Reference* for more information about the `ANALYZE` statement

DB_BLOCK_CHECKING (Block Checking Initialization Parameter)

You can set block checking for instances using the `DB_BLOCK_CHECKING` initialization parameter (the default value is `FALSE`). This checks data and index blocks whenever they are modified. `DB_BLOCK_CHECKING` is a dynamic parameter, modifiable by the `ALTER SYSTEM SET` statement. Block checking is always enabled for the system tablespace.

See Also: *Oracle9i Database Reference* for more information about the `DB_BLOCK_CHECKING` initialization parameter

Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR

Before using `DBMS_REPAIR` you must weigh the benefits of its use in relation to the liabilities. You should also examine other options available for addressing corrupt objects.

A first step is to answer the following questions:

1. What is the extent of the corruption?

To determine if there are corruptions and repair actions, execute the `CHECK_OBJECT` procedure, and query the repair table.

2. What other options are available for addressing block corruptions? Consider the following:

- Assuming the data is available from another source, drop, re-create and re-populate the object.
 - Issue the `CREATE TABLE ... AS SELECT` statement from the corrupt table to create a new one.
 - Ignore the corruption by excluding corrupt rows from select statements.
 - Perform media recovery.
3. What logical corruptions or side effects are introduced when you use `DBMS_REPAIR` to make an object usable? Can these be addressed? What is the effort required to do so?

It is possible that you do not have access to rows in blocks marked corrupt. However, a block could be marked corrupt even though there are still rows that you can validly access.

It is also possible that referential integrity constraints are broken when blocks are marked corrupt. If this occurs, disable and re-enable the constraint; any inconsistencies are reported. After fixing all problems, you should be able to successfully re-enable the constraint.

Logical corruption can occur when there are triggers defined on the table. For example, if rows are re-inserted, should insert triggers be fired or not? You can address these issues only if you understand triggers and their use in your installation.

Free list blocks can become inaccessible. If a corrupt block is at the head or tail of a free list, space management reinitializes the free list. There then can be blocks that should be on a free list, but are not. You can address this by running the `REBUILD_FREELISTS` procedure.

Indexes and tables are out of sync. You can address this by first executing the `DUMP_ORPHAN_KEYS` procedure (to obtain information from the keys that might be useful in rebuilding corrupted data). Then issue the `ALTER INDEX ... REBUILD ONLINE` statement to get the table and its indexes back in sync.

4. If repair involves loss of data, can this data be retrieved?

You can retrieve data from the index when a data block is marked corrupt. The `DUMP_ORPHAN_KEYS` procedure can help you retrieve this information. Of course, retrieving data in this manner depends on the amount of redundancy between the indexes and the table.

Task 3: Make Objects Usable

In this task DBMS_REPAIR makes the object usable by ignoring corruptions during table and index scans.

Corruption Repair: Using the FIX_CORRUPT_BLOCKS and SKIP_CORRUPT_BLOCKS Procedures

You make a corrupt object usable by establishing an environment that skips corruptions that remain outside the scope of DBMS_REPAIR's repair capabilities.

If corruptions involve a loss of data, such as a bad row in a data block, all such blocks are marked corrupt by the FIX_CORRUPT_BLOCKS procedure. Then, you can run the SKIP_CORRUPT_BLOCKS procedure, which skips blocks marked corrupt for the object. When skip is set, table and index scans skip all blocks marked corrupt. This applies to both media and software corrupt blocks.

Implications when Skipping Corrupt Blocks

If an index and table are out of sync, then a SET TRANSACTION READ ONLY transaction can be inconsistent in situations where one query probes only the index, and then a subsequent query probes both the index and the table. If the table block is marked corrupt, then the two queries return different results, thereby breaking the rules of a read-only transaction. One way to approach this is to not skip corruptions when in a SET TRANSACTION READ ONLY transaction.

A similar issue occurs when selecting rows that are chained. Essentially, a query of the same row may or may not access the corruption, thereby producing different results.

Task 4: Repair Corruptions and Rebuild Lost Data

After making an object usable, you can perform the following repair activities.

Recover Data Using the DUMP_ORPHAN_KEYS Procedures

The DUMP_ORPHAN_KEYS procedure reports on index entries that point to rows in corrupt data blocks. All such index entries are inserted into an orphan key table that stores the key and rowid of the corruption.

After the index entry information has been retrieved, you can rebuild the index using the ALTER INDEX ... REBUILD ONLINE statement.

Repair Free Lists Using the REBUILD_FREELISTS Procedure

Use this procedure if free space in segments is being managed using free lists (SEGMENT SPACE MANAGEMENT MANUAL).

When a block marked "corrupt" is found at the head or tail of a free list, the free list is reinitialized and an error is returned. Although this takes the offending block off the free list, it causes you to lose free list access to all blocks that followed the corrupt block.

You can use the REBUILD_FREELISTS procedure to reinitialize the free lists. The object is scanned, and if it is appropriate for a block to be on the free list, it is added to the master free list. Free list groups are handled by distributing blocks in an equitable fashion, one block at a time. Any blocks marked "corrupt" in the object are ignored during the rebuild.

Fix Segment Bitmaps Using the SEGMENT_FIX_STATUS Procedure

Use this procedure if free space in segments is being managed using bitmaps (SEGMENT SPACE MANAGEMENT AUTO).

This procedure either recalculates the state of a bitmap entry based on the corresponding block's current contents, or you can specify that a bitmap entry be set to a specific value. Usually, the state is recalculated correctly and there is no need to force a setting.

DBMS_REPAIR Examples

In this section, examples are presented reflecting the use of the DBMS_REPAIR procedures.

- [Using ADMIN_TABLES to Build a Repair Table or Orphan Key Table](#)
- [Using the CHECK_OBJECT Procedure to Detect Corruption](#)
- [Fixing Corrupt Blocks with the FIX_CORRUPT_BLOCKS Procedure](#)
- [Finding Index Entries Pointing into Corrupt Data Blocks: DUMP_ORPHAN_KEYS](#)
- [Rebuilding Free Lists Using the REBUILD_FREELISTS Procedure](#)
- [Enabling or Disabling the Skipping of Corrupt Blocks: SKIP_CORRUPT_BLOCKS](#)

Using ADMIN_TABLES to Build a Repair Table or Orphan Key Table

A repair table provides information about what corruptions were found by the CHECK_OBJECT procedure and how these will be addressed if the FIX_CORRUPT_BLOCKS procedure is run. Further, it is used to drive the execution of the FIX_CORRUPT_BLOCKS procedure.

An orphan key table is used when the DUMP_ORPHAN_KEYS procedure is executed and it discovers index entries that point to corrupt rows. The DUMP_ORPHAN_KEYS procedure populates the orphan key table by logging its activity and providing the index information in a usable manner.

The ADMIN_TABLE procedure is used to create, purge, or drop a repair table or an orphan key table.

Creating a Repair Table

The following example creates a repair table for the users tablespace.

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
    TABLE_NAME => 'REPAIR_TABLE',
    TABLE_TYPE => dbms_repair.repair_table,
    ACTION      => dbms_repair.create_action,
    TABLESPACE => 'USERS');
END;
/
```

For each repair or orphan key table, a view is also created that eliminates any rows that pertain to objects that no longer exist. The name of the view corresponds to the name of the repair or orphan key table, but is prefixed by DBA_ (for example DBA_REPAIR_TABLE or DBA_ORPHAN_KEY_TABLE).

The following query describes the repair table created in the previous example.

```
SQL> DESC REPAIR_TABLE
Name                               Null?    Type
-----
OBJECT_ID                           NOT NULL NUMBER
TABLESPACE_ID                       NOT NULL NUMBER
RELATIVE_FILE_ID                   NOT NULL NUMBER
BLOCK_ID                             NOT NULL NUMBER
CORRUPT_TYPE                       NOT NULL NUMBER
SCHEMA_NAME                         NOT NULL VARCHAR2(30)
OBJECT_NAME                         NOT NULL VARCHAR2(30)
BASEOBJECT_NAME                     VARCHAR2(30)
```

PARTITION_NAME		VARCHAR2(30)
CORRUPT_DESCRIPTION		VARCHAR2(2000)
REPAIR_DESCRIPTION		VARCHAR2(200)
MARKED_CORRUPT	NOT NULL	VARCHAR2(10)
CHECK_TIMESTAMP	NOT NULL	DATE
FIX_TIMESTAMP		DATE
REFORMAT_TIMESTAMP		DATE

Creating an Orphan Key Table

This example illustrates the creation of an orphan key table for the `users` tablespace.

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
    TABLE_NAME => 'ORPHAN_KEY_TABLE',
    TABLE_TYPE => dbms_repair.orphan_table,
    ACTION      => dbms_repair.create_action,
    TABLESPACE => 'USERS');
END;
/
```

The orphan key table is described in the following query:

```
SQL> DESC ORPHAN_KEY_TABLE
```

Name	Null?	Type
-----	-----	-----
SCHEMA_NAME	NOT NULL	VARCHAR2(30)
INDEX_NAME	NOT NULL	VARCHAR2(30)
IPART_NAME		VARCHAR2(30)
INDEX_ID	NOT NULL	NUMBER
TABLE_NAME	NOT NULL	VARCHAR2(30)
PART_NAME		VARCHAR2(30)
TABLE_ID	NOT NULL	NUMBER
KEYROWID	NOT NULL	ROWID
KEY	NOT NULL	ROWID
DUMP_TIMESTAMP	NOT NULL	DATE

Using the CHECK_OBJECT Procedure to Detect Corruption

The `CHECK_OBJECT` procedure checks the specified objects, and populates the repair table with information about corruptions and repair directives. You can optionally specify a range, partition name, or subpartition name when you would like to check a portion of an object.

Validation consists of checking all blocks in the object that have not previously been marked corrupt. For each block, the transaction and data layer portions are checked for self consistency. During `CHECK_OBJECT`, if a block is encountered that has a corrupt buffer cache header, then that block is skipped.

Here is an example of executing the `CHECK_OBJECT` procedure for the `scott.dept` table.

```
SET SERVEROUTPUT ON
DECLARE num_corrupt INT;
BEGIN
num_corrupt := 0;
DBMS_REPAIR.CHECK_OBJECT (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'DEPT',
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    CORRUPT_COUNT => num_corrupt);
DBMS_OUTPUT.PUT_LINE('number corrupt: ' || TO_CHAR (num_corrupt));
END;
/
```

SQL*PLUS outputs the following line, indicating one corruption:

```
number corrupt: 1
```

Querying the repair table produces information describing the corruption and suggesting a repair action.

```
SELECT OBJECT_NAME, BLOCK_ID, CORRUPT_TYPE, MARKED_CORRUPT,
       CORRUPT_DESCRIPTION, REPAIR_DESCRIPTION
FROM REPAIR_TABLE;
```

```
OBJECT_NAME                BLOCK_ID CORRUPT_TYPE MARKED_COR
-----
CORRUPT_DESCRIPTION
-----
REPAIR_DESCRIPTION
-----
DEPT                        3          1 FALSE
kdbchk: row locked by non-existent transaction
         table=0  slot=0
         lockid=32  ktbhhitc=1
mark block software corrupt
```

At this point, the corrupted block has not yet been marked corrupt, so this is the time to extract any meaningful data. After the block is marked corrupt, the entire block must be skipped.

Fixing Corrupt Blocks with the `FIX_CORRUPT_BLOCKS` Procedure

Use the `FIX_CORRUPT_BLOCKS` procedure to fix the corrupt blocks in specified objects based on information in the repair table that was previously generated by the `CHECK_OBJECT` procedure. Prior to effecting any change to a block, the block is checked to ensure the block is still corrupt. Corrupt blocks are repaired by marking the block software corrupt. When a repair is performed, the associated row in the repair table is updated with a fix timestamp.

This example fixes the corrupt block in table `scott.dept` that was reported by the `CHECK_OBJECT` procedure.

```
SET SERVEROUTPUT ON
DECLARE num_fix INT;
BEGIN
num_fix := 0;
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME=> 'DEPT',
    OBJECT_TYPE => dbms_repair.table_object,
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    FIX_COUNT=> num_fix);
DBMS_OUTPUT.PUT_LINE('num fix: ' || TO_CHAR(num_fix));
END;
/
```

SQL*Plus outputs the following line:

```
num fix: 1
```

The following query confirms that the repair was done.

```
SELECT OBJECT_NAME, BLOCK_ID, MARKED_CORRUPT
       FROM REPAIR_TABLE;
```

OBJECT_NAME	BLOCK_ID	MARKED_COR
DEPT	3	TRUE

Finding Index Entries Pointing into Corrupt Data Blocks: DUMP_ORPHAN_KEYS

The `DUMP_ORPHAN_KEYS` procedure reports on index entries that point to rows in corrupt data blocks. For each such index entry encountered, a row is inserted into the specified orphan key table. The orphan key table must have been previously created.

This information can be useful for rebuilding lost rows in the table and for diagnostic purposes.

Note: This should be run for every index associated with a table identified in the repair table.

In this example, `pk_dept` is an index on the `scott.dept` table. It is scanned to determine if there are any index entries pointing to rows in the corrupt data block.

```
SET SERVEROUTPUT ON
DECLARE num_orphans INT;
BEGIN
num_orphans := 0;
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'PK_DEPT',
    OBJECT_TYPE => dbms_repair.index_object,
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    ORPHAN_TABLE_NAME=> 'ORPHAN_KEY_TABLE',
    KEY_COUNT => num_orphans);
DBMS_OUTPUT.PUT_LINE('orphan key count: ' || TO_CHAR(num_orphans));
END;
/
```

The following line is output, indicating there are three orphan keys:

```
orphan key count: 3
```

Index entries in the orphan key table implies that the index should be rebuilt. This guarantees that a table probe and an index probe return the same result set.

Rebuilding Free Lists Using the REBUILD_FREELISTS Procedure

The `REBUILD_FREELISTS` procedure rebuilds the free lists for the specified object. All free blocks are placed on the master free list. All other free lists are zeroed. If the

object has multiple free list groups, then the free blocks are distributed among all free lists, allocating to the different groups in round-robin fashion.

This example rebuilds the free lists for the table `scott.dept`.

```
BEGIN
DBMS_REPAIR.REBUILD_FREELISTS (
  SCHEMA_NAME => 'SCOTT',
  OBJECT_NAME => 'DEPT',
  OBJECT_TYPE => dbms_repair.table_object);
END;
/
```

Enabling or Disabling the Skipping of Corrupt Blocks: SKIP_CORRUPT_BLOCKS

The `SKIP_CORRUPT_BLOCKS` procedure enables or disables the skipping of corrupt blocks during index and table scans of the specified object. When the object is a table, skip applies to the table and its indexes. When the object is a cluster, it applies to all of the tables in the cluster, and their respective indexes.

The following example enables the skipping of software corrupt blocks for the `scott.dept` table:

```
BEGIN
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
  SCHEMA_NAME => 'SCOTT',
  OBJECT_NAME => 'DEPT',
  OBJECT_TYPE => dbms_repair.table_object,
  FLAGS => dbms_repair.skip_flag);
END;
/
```

Querying `scott`'s tables using the `DBA_TABLES` view shows that `SKIP_CORRUPT` is enabled for table `scott.dept`.

```
SELECT OWNER, TABLE_NAME, SKIP_CORRUPT FROM DBA_TABLES
WHERE OWNER = 'SCOTT';
```

OWNER	TABLE_NAME	SKIP_COR
SCOTT	ACCOUNT	DISABLED
SCOTT	BONUS	DISABLED
SCOTT	DEPT	ENABLED
SCOTT	DOCINDEX	DISABLED
SCOTT	EMP	DISABLED
SCOTT	RECEIPT	DISABLED

SCOTT	SALGRADE	DISABLED
SCOTT	SCOTT_EMP	DISABLED
SCOTT	SYS_IOT_OVER_12255	DISABLED
SCOTT	WORK_AREA	DISABLED

10 rows selected.

Part IV

Database Security

Part IV addresses issues of user and privilege management affecting the security of the database. It includes the following chapters:

- [Chapter 23, "Establishing Security Policies"](#)
- [Chapter 24, "Managing Users and Resources"](#)
- [Chapter 25, "Managing User Privileges and Roles"](#)
- [Chapter 26, "Auditing Database Use"](#)

Establishing Security Policies

This chapter provides guidelines for developing security policies for database operation, and contains the following topics:

- [System Security Policy](#)
- [Data Security Policy](#)
- [User Security Policy](#)
- [Password Management Policy](#)
- [Auditing Policy](#)
- [A Security Checklist](#)

System Security Policy

This section describes aspects of system security policy, and contains the following topics:

- [Database User Management](#)
- [User Authentication](#)
- [Operating System Security](#)

Each database has one or more administrators who are responsible for maintaining all aspects of the security policy: the security administrators. If the database system is small, the database administrator may have the responsibilities of the security administrator. However, if the database system is large, a special person or group of people may have responsibilities limited to those of a security administrator.

After deciding who will manage the security of the system, a security policy must be developed for every database. A database's security policy should include several sub-policies, as explained in the following sections.

Database User Management

Database users are the access paths to the information in an Oracle database. Therefore, tight security should be maintained for the management of database users. Depending on the size of a database system and the amount of work required to manage database users, the security administrator may be the only user with the privileges required to create, alter, or drop database users. On the other hand, there may be a number of administrators with privileges to manage database users. Regardless, only trusted individuals should have the powerful privileges to administer database users.

User Authentication

Database users can be **authenticated** (verified as the correct person) by Oracle using database passwords, the host operating system, network services, or by Secure Sockets Layer (SSL).

Note: To be authenticated using network authentication services or SSL, requires that you have installed Oracle Advanced Security. Refer to the *Oracle Advanced Security Administrator's Guide* for information about these types of authentication.

User authentication and how it is specified is discussed in "[User Authentication Methods](#)" on page 24-8.

Operating System Security

If applicable, the following security issues must also be considered for the operating system environment executing Oracle and any database applications:

- Database administrators must have the operating system privileges to create and delete files.
- Typical database users should not have the operating system privileges to create or delete files related to the database.
- If the operating system identifies database roles for users, the security administrators must have the operating system privileges to modify the security domain of operating system accounts.

See Also: Your operating system specific Oracle documentation contains more information about operating system security issues

Data Security Policy

Data security includes the mechanisms that control the access to and use of the database at the object level. Your data security policy determines which users have access to a specific schema object, and the specific types of actions allowed for each user on the object. For example, user `scott` can issue `SELECT` and `INSERT` statements but not `DELETE` statements using the `emp` table. Your data security policy should also define the actions, if any, that are audited for each schema object.

Your data security policy is determined primarily by the level of security you want to establish for the data in your database. For example, it may be acceptable to have little data security in a database when you want to allow any user to create any schema object, or grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary for data security to be very controlled when you want to make a database or security administrator the only person with the privileges to create objects and grant access privileges for objects to roles and users.

Overall data security should be based on the sensitivity of data. If information is not sensitive, then the data security policy can be more lax. However, if data is sensitive, a security policy should be developed to maintain tight control over access to objects.

Some means of implementing data security include system and object privileges, and through roles. A role is a set of privileges grouped together that can be granted to users. Privileges and roles are discussed in [Chapter 25, "Managing User Privileges and Roles"](#).

Views can also implement data security because their definition can restrict access to table data. They can exclude columns containing sensitive data. Views are discussed in [Chapter 20, "Managing Views, Sequences, and Synonyms"](#).

Another means of implementing data security is through fine-grained access control and use of an associated application context. Fine-grained access control is a feature of Oracle that enables you to implement security policies with functions, and to associate those security policies with tables or views. In effect, the security policy function generates a `WHERE` condition that is appended to a SQL statement, thereby restricting the users access to rows of data in the table or view. An application context is a secure data cache for storing information used to make access control decisions.

See Also:

- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*

The above manuals contain information about implementing fine-grained access control and an application context.

User Security Policy

This section describes aspects of user security policy, and contains the following topics:

- [General User Security](#)
- [End-User Security](#)
- [Administrator Security](#)
- [Application Developer Security](#)
- [Application Administrator Security](#)

General User Security

For all types of database users, consider the following general user security issues:

- [Password Security](#)

- [Privilege Management](#)

Password Security

If user authentication is managed by the database, security administrators should develop a password security policy to maintain database access security. For example, database users should be required to change their passwords at regular intervals, and of course, when their passwords are revealed to others. By forcing a user to modify passwords in such situations, unauthorized database access can be reduced.

To better protect the confidentiality of your password, Oracle can be configured to use encrypted passwords for client/server and server/server connections.

Note: It is *strongly* recommended that you configure Oracle to encrypt passwords in client/server and server/server connections. Otherwise, a malicious user "snooping" on the network can grab an unencrypted password, and use it to connect to the database as another user, thereby "impersonating" that user.

By setting the following values, you can require that the password used to verify a connection always be encrypted:

- Set the `ORA_ENCRYPT_LOGIN` environment variable to `TRUE` on the client machine.
- Set the `DBLINK_ENCRYPT_LOGIN` server initialization parameter to `TRUE`.

If enabled at both the client and server, passwords will not be sent across the network "in the clear", but will be encrypted using a modified DES (Data Encryption Standard) algorithm.

The `DBLINK_ENCRYPT_LOGIN` initialization parameter is used for connections between two Oracle servers (for example, when performing distributed queries). If you are connecting from a client, Oracle checks the `ORA_ENCRYPT_LOGIN` environment variable.

Whenever you attempt to connect to a server using a password, Oracle encrypts the password before sending it to the server. If the connection fails and auditing is enabled, the failure is noted in the audit log. Oracle then checks the appropriate `DBLINK_ENCRYPT_LOGIN` or `ORA_ENCRYPT_LOGIN` value. If it set to `FALSE`, Oracle attempts the connection again using an unencrypted version of the password. If the connection is successful, the connection replaces the previous

failure in the audit log, and the connection proceeds. To prevent malicious users from forcing Oracle to re-attempt a connection with an unencrypted version of the password, you must set the appropriate values to `TRUE`.

Privilege Management

Security administrators should consider issues related to privilege management for all types of users. For example, in a database with many usernames, it may be beneficial to use roles (which are named groups of related privileges that you grant to users or other roles) to manage the privileges available to users. Alternatively, in a database with a handful of usernames, it may be easier to grant privileges explicitly to users and avoid the use of roles.

Security administrators managing a database with many users, applications, or objects should take advantage of the benefits offered by roles. Roles greatly simplify the task of privilege management in complicated environments.

End-User Security

Security administrators must define a policy for end-user security. If a database has many users, the security administrator can decide which groups of users can be categorized into user groups, and then create user roles for these groups. The security administrator can grant the necessary privileges or application roles to each user role, and assign the user roles to the users. To account for exceptions, the security administrator must also decide what privileges must be explicitly granted to individual users.

Using Roles for End-User Privilege Management

Roles are the easiest way to grant and manage the common privileges needed by different groups of database users.

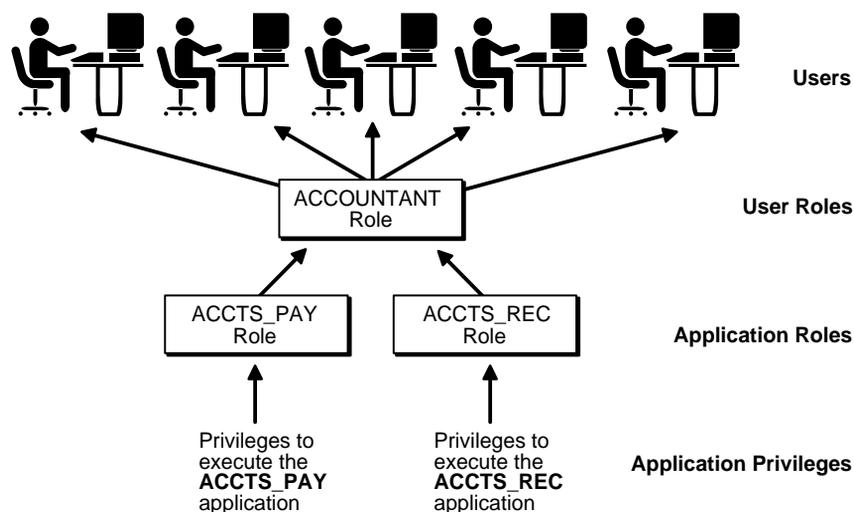
Consider a situation where every user in the accounting department of a company needs the privileges to run the `accts_receivable` and `accts_payable` database applications. Roles are associated with both applications, and they contain the object privileges necessary to execute those applications.

The following actions, performed by the database or security administrator, address this simple security situation:

1. Create a role named `accountant`.
2. Grant the roles for the `accts_receivable` and `accts_payable` database applications to the `accountant` role.

3. Grant each user of the accounting department the `accountant` role.
This security model is illustrated in [Figure 23-1](#).

Figure 23-1 User Role



This plan addresses the following potential situations:

- If accountants subsequently need a role for a new database application, that application's role can be granted to the `accountant` role, and all users in the accounting department will automatically receive the privileges associated with the new database application. The application's role does not need to be granted to individual users requiring use of the application.
- Similarly, if the accounting department no longer requires the need for a specific application, the application's role can be dropped from the `accountant` role.
- If the privileges required by the `accts_receivable` or `accts_payable` applications change, the new privileges can be granted to, or revoked from, the application's role. The security domain of the `accountant` role, and all users granted the `accountant` role, automatically reflect the privilege modification.

Utilize roles in all possible situations to make end-user privilege management efficient and simple.

Using a Directory Service for End-User Privilege Management

You can also manage users and their authorizations centrally, in a directory service, through the enterprise user and enterprise role features of Oracle Advanced Security. See the *Oracle Advanced Security Administrator's Guide* for information about this functionality.

Administrator Security

Security administrators should have a policy addressing database administrator security. For example, when the database is large and there are several types of database administrators, the security administrator may decide to group related administrative privileges into several administrative roles. The administrative roles can then be granted to appropriate administrator users. Alternatively, when the database is small and has only a few administrators, it may be more convenient to create one administrative role and grant it to all administrators.

See Also: [Chapter 1, "The Oracle Database Administrator"](#) contains a more thorough discussion of administrator security

Protection for Connections as SYS and SYSTEM

After database creation, and if you used the default passwords for `SYS` and `SYSTEM`, *immediately* change the passwords for the `SYS` and `SYSTEM` administrative usernames. Connecting as `SYS` or `SYSTEM` gives a user powerful privileges to modify a database. For example, connecting as `SYS` allows a user to alter data dictionary tables. The privileges associated with these usernames are extremely sensitive, and should only be available to select database administrators.

If you have installed options that have caused other administrative usernames to be created, such username accounts are initially created locked. To unlock these accounts, use the `ALTER USER` statement. The `ALTER USER` statement should also be used to change the associated passwords for these accounts.

The passwords for these accounts can be modified using the procedures described in "[Altering Users](#)" on page 24-6.

Protection for Administrator Connections

Only database administrators should have the capability to connect to a database with administrative privileges. For example:

```
CONNECT username/password AS SYSDBA/SYSOPER
```

Connecting as `SYSOPER` gives a user the ability to perform basic operational tasks (such as `STARTUP`, `SHUTDOWN`, and recovery operations). Connecting as `SYSDBA` gives the user these abilities plus unrestricted privileges to do anything to a database or the objects within a database (including, `CREATE`, `DROP`, and `DELETE`). Connecting as `SYSDBA` places a user in the `SYS` schema, where they can alter data dictionary tables.

Using Roles for Administrator Privilege Management

Roles are the easiest way to restrict the powerful system privileges and roles required by personnel administrating the database.

Consider a scenario where the database administrator responsibilities at a large installation are shared among several database administrators, each responsible for the following specific database management jobs:

- Object creation and maintenance
- Database tuning and performance
- Creation of new users and granting roles and privileges to database users
- Routine database operation (for example: `STARTUP`, `SHUTDOWN`, and backup and recovery operations)
- Emergency situations, such as database recovery

There are also new, inexperienced database administrators needing limited capabilities to experiment with database management

In this scenario, the security administrator should structure the security for administrative personnel as follows:

1. Define six roles to contain the distinct privileges required to accomplish each type of job (for example, `dba_objects`, `dba_tune`, `dba_security`, `dba_maintain`, `dba_recov`, `dba_new`).
2. Grant each role the appropriate privileges.
3. Grant each type of database administrator the corresponding role.

This plan diminishes the likelihood of future problems in the following ways:

- If a database administrator's job description changes to include more responsibilities, that database administrator can be granted other administrative roles corresponding to the new responsibilities.

- If a database administrator's job description changes to include fewer responsibilities, that database administrator can have the appropriate administrative roles revoked.
- The data dictionary always stores information about each role and each user, so information is available to disclose the task of each administrator.

Application Developer Security

Security administrators must define a special security policy for the application developers using a database. A security administrator could grant the privileges to create necessary objects to application developers. Or, alternatively, the privileges to create objects could be granted only to a database administrator, who then receives requests for object creation from developers.

Application Developers and Their Privileges

Database application developers are unique database users who require special groups of privileges to accomplish their jobs. Unlike end users, developers need system privileges, such as `CREATE TABLE`, `CREATE PROCEDURE`, and so on. However, only specific system privileges should be granted to developers to restrict their overall capabilities in the database.

The Application Developer's Environment: Test and Production Databases

In many cases, application development is restricted to test databases and is not allowed on production databases. This restriction ensures that application developers do not compete with end users for database resources, and that they cannot detrimentally affect a production database.

After an application has been thoroughly developed and tested, it is permitted access to the production database and made available to the appropriate end users of the production database.

Free Versus Controlled Application Development

The database administrator can define the following options when determining which privileges should be granted to application developers:

- Free development
An application developer is allowed to create new schema objects, including tables, indexes, procedures, packages, and so on. This option allows the application developer to develop an application independent of other objects.

- **Controlled Development**

An application developer is not allowed to create new schema objects. All required tables, indexes, procedures, and so on are created by a database administrator, as requested by an application developer. This option allows the database administrator to completely control a database's space usage and the access paths to information in the database.

Although some database systems use only one of these options, other systems could mix them. For example, application developers can be allowed to create new stored procedures and packages, but not allowed to create tables or indexes. A security administrator's decision regarding this issue should be based on the following:

- The control desired over a database's space usage
- The control desired over the access paths to schema objects
- The database used to develop applications—if a test database is being used for application development, a more liberal development policy would be in order

Roles and Privileges for Application Developers

Security administrators can create roles to manage the privileges required by the typical application developer. For example, a typical role named `APPLICATION_DEVELOPER` might include the `CREATE TABLE`, `CREATE VIEW`, and `CREATE PROCEDURE` system privileges. Consider the following when defining roles for application developers:

- `CREATE` system privileges are usually granted to application developers so that they can create their own objects. However, `CREATE ANY` system privileges, which allow a user to create an object in any user's schema, are not usually granted to developers. This restricts the creation of new objects only to the developer's user account.
- Object privileges are rarely granted to roles used by application developers. This is because granting object privileges through roles often restricts their usability in the creation of other objects (primarily views and stored procedures). It is more practical to allow application developers to create their own objects for development purposes.

Space Restrictions Imposed on Application Developers

While application developers are typically given the privileges to create objects as part of the development process, security administrators must maintain limits on what and how much database space can be used by each application developer. For

example, as the security administrator, you should specifically set or restrict the following limits for each application developer:

- The tablespaces in which the developer can create tables or indexes
- The quota for each tablespace accessible to the developer

Both limitations can be set by altering a developer's security domain. This is discussed in "[Altering Users](#)" on page 24-6.

Application Administrator Security

In large database systems with many database applications, you might consider assigning application administrators. An application administrator is responsible for the following types of tasks:

- Creating roles for an application and managing the privileges of each application role
- Creating and managing the objects used by a database application
- Maintaining and updating the application code and Oracle procedures and packages, as necessary

Often, an application administrator is also the application developer who designed an application. However, an application administrator could be any individual familiar with the database application.

Password Management Policy

Database security systems that are dependent on passwords require that passwords be kept secret at all times. But, passwords are vulnerable to theft, forgery, and misuse. To allow for greater control over database security, Oracle's password management policy is controlled by DBAs and security officers through user profiles.

You use the `CREATE PROFILE` statement to create a user profile. The profile is assigned to a user with the `CREATE USER` or `ALTER USER` statement. Details of creating and altering database users are not discussed in this section. This section is concerned with the password parameters that can be specified using the `CREATE PROFILE` (or `ALTER PROFILE`) statement.

This section contains the following topics relating to password management:

This section describes the following aspects of Oracle password management:

- [Account Locking](#)
- [Password Aging and Expiration](#)
- [Password History](#)
- [Password Complexity Verification](#)

See Also:

- ["Managing Resources with Profiles"](#) on page 24-18
- ["Managing Oracle Users"](#) on page 24-2
- *Oracle9i SQL Reference* for syntax and specific information about SQL statements discussed in this section

Account Locking

When a particular user exceeds a designated number of failed login attempts, the server automatically locks that user's account. You specify the permissible number of failed login attempts using the `CREATE PROFILE` statement. You can also specify the amount of time accounts remain locked.

In the following example, the maximum number of failed login attempts for the user `ashwini` is four, and the amount of time the account will remain locked is 30 days. The account will unlock automatically after the passage of 30 days.

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30;
ALTER USER ashwini PROFILE prof;
```

If you do not specify a time interval for unlocking the account, `PASSWORD_LOCK_TIME` assumes the value specified in a default profile. If you specify `PASSWORD_LOCK_TIME` as `UNLIMITED`, the account must be explicitly unlocked using an `ALTER USER` statement. For example, assuming that `PASSWORD_LOCK_TIME UNLIMITED` is specified for `ashwini`, then the following statement must be used to unlock the account:

```
ALTER USER ashwini ACCOUNT UNLOCK;
```

After a user successfully logs into an account, that user's unsuccessful login attempt count, if there is one, is reset to 0.

The security officer can also explicitly lock user accounts. When this occurs, the account cannot be unlocked automatically, and only the security officer should

unlock the account. The `CREATE USER` or `ALTER USER` statements are used to explicitly lock or unlock user accounts. For example, the following statement locks user account `susan`:

```
ALTER USER susan ACCOUNT LOCK;
```

Password Aging and Expiration

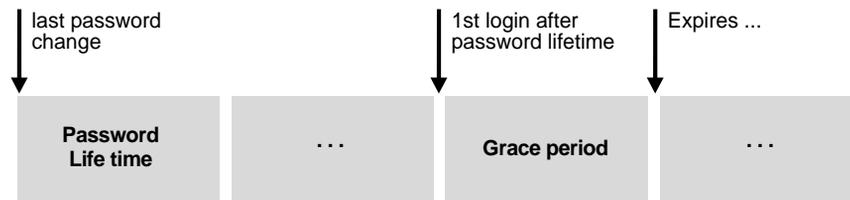
Use the `CREATE PROFILE` statement to specify a maximum lifetime for passwords. When the specified amount of time passes and the password expires, the user or DBA must change the password. The following statements create and assign a profile to user `ashwini`, and the `PASSWORD_LIFE_TIME` clause specifies that `ashwini` can use the same password for 90 days before it expires.

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30
  PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

You can also specify a grace period for password expiration. Users enter the grace period upon the first attempt to log in to a database account after their password has expired. During the grace period, a warning message appears each time users try to log in to their accounts, and continues to appear until the grace period expires. Users must change the password within the grace period. If the password is not changed within the grace period, thereafter users are prompted for a new password each time an attempt is made to access their accounts. Access to an account is denied until a new password is supplied.

Figure 23–2 shows the chronology of the password lifetime and grace period.

Figure 23–2 Chronology of Password Lifetime and Grace Period



In the following example, the profile assigned to `ashwini` includes the specification of a grace period: `PASSWORD_GRACE_TIME = 3`. The first time `ashwini` tries to

log in to the database after 90 days (this can be any day after the 90th day; that is, the 70th day, 100th day, or another day), she receives a warning message that her password will expire in three days. If three days pass, and she does not change her password, the password expires. Thereafter, she receives a prompt to change her password on any attempt to log in, and cannot log in until she does so.

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30
  PASSWORD_LIFE_TIME 90
  PASSWORD_GRACE_TIME 3;
ALTER USER ashwini PROFILE prof;
```

Oracle provides a means of explicitly expiring a password. The `CREATE USER` and `ALTER USER` statements provide this functionality. The following statement creates a user with an expired password. This setting forces the user to change the password before the user can log in to the database.

```
CREATE USER jbrown
  IDENTIFIED BY zX83yT
  ...
  PASSWORD EXPIRE;
```

Password History

Use the `CREATE PROFILE` statement to specify a time interval during which users cannot reuse a password. In the following statement, a profile is defined where the `PASSWORD_REUSE_TIME` clause specifies that the user cannot reuse the password for 60 days.

```
CREATE PROFILE prof LIMIT
  PASSWORD_REUSE_TIME 60
  PASSWORD_REUSE_MAX UNLIMITED;
```

In the next statement, the `PASSWORD_REUSE_MAX` clause specifies that the number of password changes the user must make before the current password can be used again is three.

```
CREATE PROFILE prof LIMIT
  PASSWORD_REUSE_MAX 3
  PASSWORD_REUSE_TIME UNLIMITED;
```

Note: If you specify `PASSWORD_REUSE_TIME` or `PASSWORD_REUSE_MAX`, you must set the other to `UNLIMITED` or not specify it at all.

Password Complexity Verification

Oracle's password complexity verification routine can be specified using a PL/SQL script (`UTLPPWDMG.SQL`), which sets the default profile parameters.

The password complexity verification routine performs the following checks:

- The password has a minimum length of four.
- The password is not the same as the username.
- The password has at least one alpha, one numeric, and one punctuation mark character.
- The password is not a simple or obvious word, such as `welcome`, `account`, `database`, or `user`.
- The password differs from the previous password by at least 3 characters.

Note: Oracle recommends that you do not change passwords using the `ALTER USER` statement because it does not fully support the password verification function. Instead, you should use `LNOCIPasswordChange()` to change passwords.

Password Verification Routine Formatting Guidelines

You can enhance the existing password verification complexity routine or create other password verification routines using PL/SQL or third-party tools.

The PL/SQL call must adhere to the following format:

```
routine_name
(
  userid_parameter IN VARCHAR(30),
  password_parameter IN VARCHAR (30),
  old_password_parameter IN VARCHAR (30)
)
RETURN BOOLEAN
```

After a new routine is created, it must be assigned as the password verification routine using the user's profile or the system default profile.

```
CREATE/ALTER PROFILE profile_name LIMIT
PASSWORD_VERIFY_FUNCTION routine_name
```

The password verify routine must be owned by SYS.

Sample Password Verification Routine

You can use this sample password verification routine as a model when developing your own complexity checks for a new password.

The default password complexity function performs the following minimum complexity checks:

- The password satisfies minimum length requirements.
- The password is not the username. You can modify this function based on your requirements.

This function must be created in SYS schema, and you must connect SYS/password AS SYSDBA before running the script.

```
CREATE OR REPLACE FUNCTION verify_function
(username varchar2,
 password varchar2,
 old_password varchar2)
RETURN boolean IS
n boolean;
m integer;
differ integer;
isdigit boolean;
ischar boolean;
ispunct boolean;
digitarray varchar2(20);
punctarray varchar2(25);
chararray varchar2(52);

BEGIN
    digitarray:= '0123456789';
    chararray:= 'abcdefghijklmnopqrstuvwxyzaBcDEFGHIJKLmNOPQRStUVWxyz';
    punctarray:= '!"#$%&'()*+,-/:;<=>?_';

    --Check if the password is same as the username
    IF password = username THEN
        raise_application_error(-20001, 'Password same as user');
```

```

END IF;

--Check for the minimum length of the password
IF length(password) < 4 THEN
    raise_application_error(-20002, 'Password length less than 4');
END IF;

--Check if the password is too simple. A dictionary of words may be
--maintained and a check may be made so as not to allow the words
--that are too simple for the password.
IF NLS_LOWER(password) IN ('welcome', 'database', 'account', 'user',
    'password', 'oracle', 'computer', 'abcd')
    THEN raise_application_error(-20002, 'Password too simple');
END IF;

--Check if the password contains at least one letter,
--one digit and one punctuation mark.
--1. Check for the digit
--You may delete 1. and replace with 2. or 3.
isdigit:=FALSE;
m := length(password);
FOR i IN 1..10 LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(digitarray,i,1) THEN
            isdigit:=TRUE;
            GOTO findchar;
        END IF;
    END LOOP;
END LOOP;
IF isdigit = FALSE THEN
    raise_application_error(-20003, 'Password should contain at least one \
    digit, one character and one punctuation');
END IF;
--2. Check for the character

<<findchar>>
ischar:=FALSE;
FOR i IN 1..length(chararray) LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(chararray,i,1) THEN
            ischar:=TRUE;
            GOTO findpunct;
        END IF;
    END LOOP;
END LOOP;

```

```

IF ischar = FALSE THEN
    raise_application_error(-20003, 'Password should contain at least one digit,\
    one character and one punctuation');
END IF;
--3. Check for the punctuation

<<findpunct>>
ispunct:=FALSE;
FOR i IN 1..length(punctarray) LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(punctarray,i,1) THEN
            ispunct:=TRUE;
            GOTO endsearch;
        END IF;
    END LOOP;
END LOOP;
IF ispunct = FALSE THEN raise_application_error(-20003, 'Password should \
    contain at least one digit, one character and one punctuation');
END IF;

<<endsearch>>
--Check if the password differs from the previous password by at least 3 letters
IF old_password = '' THEN
    raise_application_error(-20004, 'Old password is null');
END IF;
--Everything is fine; return TRUE ;
differ := length(old_password) - length(password);
IF abs(differ) < 3 THEN
    IF length(password) < length(old_password) THEN
        m := length(password);
    ELSE
        m:= length(old_password);
    END IF;
    differ := abs(differ);
    FOR i IN 1..m LOOP
        IF substr(password,i,1) != substr(old_password,i,1) THEN
            differ := differ + 1;
        END IF;
    END LOOP;
    IF differ < 3 THEN
        raise_application_error(-20004, 'Password should differ by at \
            least 3 characters');
    END IF;
END IF;
--Everything is fine; return TRUE ;

```

```
RETURN(TRUE);  
END;
```

Auditing Policy

Security administrators should define a policy for the auditing procedures of each database. You may, for example, decide to have database auditing disabled unless questionable activities are suspected. When auditing is required, the security administrator must decide what level of detail to audit the database; usually, general system auditing is followed by more specific types of auditing after the origins of suspicious activity are determined. Auditing is discussed in [Chapter 26](#), "Auditing Database Use".

A Security Checklist

Information security and privacy and protection of corporate assets and data are of pivotal importance in any business. Oracle9i comprehensively addresses the need for information security by offering cutting-edge security features such as deep data protection, auditing, scalable security, secure hosting and data exchange.

The Oracle9i database server leads the industry in security. However, in order to fully maximize the security features offered by Oracle9i in any business environment, it is imperative that Oracle9i itself is well-protected. Furthermore, proper use of its security features and adherence to basic security practices will help protect against database-related threats and attacks and provide a much more secure operating environment for the Oracle9i database.

This security checklist provides guidance on configuring Oracle9i in a secure manner by adhering to and recommending industry-standard "best security practices" for operational database deployments.

Details on specific database-related tasks and actions can be found throughout the Oracle documentation set.

1. INSTALL ONLY WHAT IS REQUIRED

The Oracle9i CD pack contains a host of options and products in addition to the database server. Install additional products and options only as necessary. Or, following a typical installation (if avoiding a custom installation), deinstall options and products that are not necessary. There is no need to maintain the additional products and options if they are not being used. They can always be properly and easily reinstalled as required.

2. LOCK AND EXPIRE DEFAULT USER ACCOUNTS

Oracle9i installs with a number of default (preset) database server user accounts. The Database Client Administration tool (DBCA) automatically locks and expires all default database user accounts except the following upon successful installation of the database server:

- SYS
- SYSTEM
- SCOTT
- DBSNMP
- OUTLN
- The three JSERV users

If a manual (not utilizing DBCA) installation of Oracle9i is performed, none of the default database users are locked upon successful installation of the database server. If left open in their default states, these user accounts can be exploited to gain unauthorized access to data or disrupt database operations. *Lock and expire* all default database user accounts except SYS, SYSTEM, SCOTT, DBSNMP, OUTLN and the three JSERV database users after performing any kind of initial installation that does not utilize DBCA. Oracle9i provides SQL to perform such operations.

Provided below is the table of database users after a typical Oracle9i installation utilizing DBCA.

USERNAME	ACCOUNT_STATUS
ADAMS	EXPIRED & LOCKED
AURORA\$JIS\$UTILITY\$	OPEN
AURORA\$ORB\$UNAUTHENTICATED	OPEN
BLAKE	EXPIRED & LOCKED
CLARK	EXPIRED & LOCKED
CTXSYS	EXPIRED & LOCKED
DBSNMP	OPEN
HR	EXPIRED & LOCKED
JONES	EXPIRED & LOCKED
LBACSYS	EXPIRED & LOCKED

USERNAME	ACCOUNT_STATUS
MDSYS	EXPIRED & LOCKED
OE	EXPIRED & LOCKED
OLAPDBA	EXPIRED & LOCKED
OLAPSVR	EXPIRED & LOCKED
OLAPSYS	EXPIRED & LOCKED
ORDPLUGINS	EXPIRED & LOCKED
ORDSYS	EXPIRED & LOCKED
OSE\$HTTP\$ADMIN	OPEN
OUTLN	OPEN
PM	EXPIRED & LOCKED
QS	EXPIRED & LOCKED
QS_ADM	EXPIRED & LOCKED
QS_CB	EXPIRED & LOCKED
QS_CBADM	EXPIRED & LOCKED
QS_CS	EXPIRED & LOCKED
QS_ES	EXPIRED & LOCKED
QS_OS	EXPIRED & LOCKED
QS_WS	EXPIRED & LOCKED
SCOTT	OPEN
SH	EXPIRED & LOCKED
SYS	OPEN
SYSTEM	OPEN

If any default database server user account other the ones left open is required for any reason, a database administrator (DBA) need simply unlock and activate that account with a new, meaningful password.

3. CHANGE DEFAULT USER PASSWORDS

The most trivial method by which Oracle9i can be compromised is a default database server user account which still has a default password associated with it *even after installation*.

a. Change default passwords of administrative users

In Oracle9i, SYS installs with a default password of CHANGE_ON_INSTALL and SYSTEM installs with a default password of MANAGER. Change the default passwords associated with users SYS and SYSTEM immediately upon installation of the database server.

b. Change default passwords of all users

In Oracle9i, SCOTT installs with default password TIGER and the three JSERV accounts (AURORA\$JIS\$UTILITY\$, AURORA\$ORB\$UNAUTHENTICATED and OSE\$HTTP\$ADMIN) each install with randomly-generated passwords. Each of the other accounts install with a default password that is exactly the same as that user account (for example, user MDSYS installs with password MDSYS).

Change the passwords for SCOTT, DBSNMP, OUTLN and the three JSERV user accounts immediately upon installation as well. If any of the other default user accounts that were locked and expired upon installation need to be activated, assign a new meaningful password to that user account.

Even though Oracle does not explicitly mandate changing the default password for user SCOTT, Oracle nevertheless recommends that this user account also be locked unless it is being actively used.

c. Enforce password management

Oracle recommends that basic password management rules (such as password length, history, complexity, and so forth) as provided by the database be applied to all user passwords and that all users be required to change their passwords periodically.

Oracle also recommends, if possible, utilizing Oracle Advanced Security (an option to the Enterprise Edition of Oracle9i) with network authentication services (such as Kerberos), token cards, smart cards or X.509 certificates. These services enable strong authentication of users to provide better protection against unauthorized access to Oracle9i.

4. ENABLE DATA DICTIONARY PROTECTION

Oracle recommends that customers implement data dictionary protection to prevent users having the ANY system privileges from using such privileges on the data dictionary.

To enable dictionary protection, set the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter, in the following manner:

```
O7_DICTIONARY_ACCESSIBILITY = FALSE
```

By doing so, only those authorized users making DBA-privileged (for example `CONNECT / AS SYSDBA`) connections can use the `ANY` system privilege on the data dictionary. If this parameter is not set to the value recommended above, any user with a `DROP ANY TABLE` (for example) system privilege will be able to maliciously drop parts of the data dictionary.

However, if a user requires view access to the data dictionary, it is permissible to grant that user the `SELECT ANY DICTIONARY` system privilege.

Note that in Oracle9i, `O7_DICTIONARY_ACCESSIBILITY = FALSE` by default; in Oracle8i, the parameter is set to `TRUE` by default and must specifically be changed to `FALSE` to enable this security feature.

5. PRACTICE PRINCIPLE OF LEAST PRIVILEGE

a. Grant necessary privileges only

Do not provide database users more privileges than are necessary. In other words, *principle of least privilege* is that a user be given only those privileges that are actually required to efficiently and succinctly perform his or her job.

To implement least privilege, restrict: 1) the number of `SYSTEM` and `OBJECT` privileges granted to database users, and 2) the number of `SYS`-privileged connections to the database as much as possible. For example, there is generally no need to grant `CREATE ANY TABLE` to any non DBA-privileged user.

b. Revoke unnecessary privileges from `PUBLIC`

Revoke all unnecessary privileges and roles from the database server user group `PUBLIC`. `PUBLIC` acts as a default role granted to every user in an Oracle database. Any database user can exercise privileges that are granted to `PUBLIC`. Such privileges include `EXECUTE` on various PL/SQL packages that may permit a minimally privileged user to access and execute packages that he may not directly be permitted to access. The more powerful packages that may potentially be misused are listed in the following table:

Package	Description
UTL_SMTP	This package permits arbitrary mail messages to be sent from one arbitrary user to another arbitrary user. Granting this package to PUBLIC may permit unauthorized exchange of mail messages.
UTL_TCP	This package permits outgoing network connections to be established by the database server to any receiving (or waiting) network service. Thus, arbitrary data may be sent between the database server and any waiting network service.
UTL_HTTP	This package allows the database server to request and retrieve data using HTTP. Granting this package to PUBLIC may permit data to be sent using HTML forms to a malicious Web site.
UTL_FILE	If configured improperly, this package allows text level access to any file on the host operating system. Even when properly configured, this package does not distinguish between its calling applications with the result that one application with access to UTL_FILE may write arbitrary data into the same location that is written to by another application.
DBMS_RANDOM	This package can be used to encrypt stored data. Generally, most users should not have the privilege to encrypt data since encrypted data may be non-recoverable if the keys are not securely generated, stored, and managed.

These packages are extremely useful to some applications that need them and require proper configuration and usage. These packages may not be suitable for other applications. Thus, unless absolutely necessary, revoke them from PUBLIC.

c. Restrict permissions on run-time facilities

Do not assign "all permissions" to any database server run-time facility such as the Oracle Java Virtual Machine (OJVM). Grant specific permissions to the explicit document root file paths for such facilities that may execute files and packages outside the database server.

An example of a vulnerable run-time call:

```
call dbms_java.grant_permission('SCOTT',
'SYS:java.io.FilePermission','<<ALL FILES>>','read');
```

An example of a better (more secure) run-time call:

```
call dbms_java.grant_permission('SCOTT',
```

```
'SYS:java.io.FilePermission','<<actual directory path>>','read');
```

6. ENFORCE ACCESS CONTROLS EFFECTIVELY

Authenticate clients properly.

Remote authentication is a security feature provided by Oracle9i such that if turned on (`TRUE`), it defers authentication of users to the remote client connecting to an Oracle database. Thus, the database implicitly trusts any client to have authenticated itself properly. Note that clients, in general, such as PCs, are not trusted to perform operating system authentication properly and therefore, it is very poor security practice to turn on this feature.

In a more secure configuration where this feature is turned off (`FALSE`), it enforces proper, server-based authentication of clients connecting to an Oracle database.

To restrict remote authentication and thereby defer client trust to the database, set the `REMOTE_OS_AUTHENT` initialization parameter in the following manner:

```
REMOTE_OS_AUTHENT = FALSE
```

7. RESTRICT OPERATING SYSTEM ACCESS

Limit the number of operating system users.

Limit the privileges of the operating system accounts (administrative, root-privileged or DBA) on the Oracle9i host (physical machine) to the least required for the user.

Oracle also recommends:

- Restricting the ability to modify the default file and directory permissions for the Oracle9i home (installation) directory or its contents. Even privileged operating system users and the Oracle owner should not modify these permissions, unless instructed otherwise by Oracle Corporation.
- Ensuring that when providing a path or file to the database, neither the file nor any part of the path is modifiable by an untrusted user. The file and all components of the path should be owned by the DBA or some trusted account, such as root. This recommendation applies to all types of files: data files, log files, trace files, external tables, bfiles, etc.

8. RESTRICT NETWORK ACCESS

- a. Utilize a firewall

Keep the database server behind a firewall. Oracle9i's network infrastructure, Oracle Net (formerly known as Net8 and SQL*Net), offers support for a variety of firewalls from various vendors. Supported proxy-enabled firewalls include Network Associates' Gauntlet and Axent's Raptor. Supported packet-filtered firewalls include Cisco's PIX Firewall and supported stateful inspection firewalls (more sophisticated packet-filtered firewalls) include CheckPoint's Firewall-1.

b. Never poke a hole through a firewall

If Oracle9i is behind a firewall, do not, under any circumstances, poke a hole through the firewall; for example, do not leave open Oracle Listener's 1521 port to make a connection to the Internet or vice versa.

Doing so will introduce a number of significant security vulnerabilities including more port openings through the firewall, multi-threaded operating system server issues and revelation of crucial information on database(s) behind the firewall. Furthermore, an Oracle Listener running without an established password may be probed for critical details about the database(s) on which it is listening such as trace and logging information, banner information and database descriptors and service names.

Such a plethora of information and the availability of an ill-configured firewall will provide an attacker ample opportunity to launch malicious attacks on the target database(s).

c. Prevent unauthorized administration of the Oracle Listener

Always establish a meaningful, well-formed password for the Oracle Listener to prevent remote configuration of the Oracle Listener. Additionally, set the `listener.ora` (Oracle Listener control file) security configuration parameter in the following manner:

```
ADMIN_RESTRICTIONS_listener_name = ON
```

Doing so will also prevent unauthorized administration of the Oracle Listener.

d. Check network IP addresses

Utilize the Oracle Net "valid node checking" security feature to allow or deny access to Oracle server processes from network clients with specified IP addresses. To use this feature, set the following `protocol.ora` (Oracle Net configuration file) parameters:

```
tcp.validnode_checking = YES  
  
tcp.excluded_nodes = {list of IP addresses}  
  
tcp.invited_nodes = {list of IP addresses}
```

The first parameter turns on the feature whereas the latter two parameters respectively deny or allow specific client IP addresses from making connections to the Oracle Listener (and thereby preventing potential Denial of Service attacks).

e. Encrypt network traffic

If possible, utilize Oracle Advanced Security to encrypt network traffic between clients, databases and application servers. (Note that Oracle Advanced Security is available only with the Enterprise Edition of the Oracle database).

f. Harden the operating system

Harden the host operating system by disabling all unnecessary operating system services. Both UNIX and Windows platforms provide a variety of operating system services, most of which are not necessary for most deployments. Such services include FTP, TFTP, TELNET, and so forth. Be sure to close both the UDP and TCP ports for each service that is being disabled. Disabling one type of port and not the other does not make the operating system more secure.

9. APPLY ALL SECURITY PATCHES AND WORKAROUNDS

Always apply all relevant and current security patches for both the operating system on which Oracle9i resides and Oracle9i itself, and for all installed Oracle9i options and components thereof.

Periodically check the security site on Oracle Technology Network for details on security alerts released by Oracle Corporation.

<http://otn.oracle.com/deploy/security/alerts.htm>

Also check Oracle Worldwide Support Service's site, Metalink, for details on available and upcoming security-related patches.

<http://metalink.oracle.com>

10. CONTACT ORACLE SECURITY PRODUCTS

If you believe that you have found a security vulnerability in Oracle9i, submit an iTAR to Oracle Worldwide Support Services using Metalink, or e-mail a complete description of the problem, including product version and platform, together with any exploit scripts and examples to the following address:

`secalert_us@oracle.com`

Managing Users and Resources

This chapter describes how to control access to an Oracle database, and contains the following topics:

- [Managing Oracle Users](#)
- [User Authentication Methods](#)
- [Managing Resources with Profiles](#)
- [Viewing Information About Database Users and Profiles](#)

See Also:

- [Chapter 23, "Establishing Security Policies"](#)
- [Chapter 25, "Managing User Privileges and Roles"](#)

Managing Oracle Users

Each Oracle database has a list of valid database users. To access a database, a user must run a database application and connect to the database instance using a valid user name defined in the database. This section explains how to manage users for a database, and contains the following topics:

- [Creating Users](#)
- [Altering Users](#)
- [Dropping Users](#)

See Also: *Oracle9i SQL Reference* for more information about SQL statements used for managing users

Creating Users

You create a database user with the `CREATE USER` statement. To create a user, you must have the `CREATE USER` system privilege. Because it is a powerful privilege, a DBA or security administrator is normally the only user who has the `CREATE USER` system privilege.

The following example creates a user and specifies that user's password, default tablespace, temporary tablespace where temporary segments are created, tablespace quotas, and profile.

```
CREATE USER jward
  IDENTIFIED BY az7bc2
  DEFAULT TABLESPACE data_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  TEMPORARY TABLESPACE temp_ts
  PROFILE clerk;
GRANT connect TO jward;
```

A newly created user cannot connect to the database until granted the `CREATE SESSION` system privilege. Usually, a newly created user is granted a role similar to the predefined role `CONNECT` (used in this example) that specifies the `CREATE SESSION` and other basic privileges required to access a database.

This section refers to the above example as it discusses the following aspects of creating a user:

- [Specifying a Name](#)
- [Setting a User's Authentication](#)

- [Assigning a Default Tablespace](#)
- [Assigning Tablespace Quotas](#)
- [Assigning a Temporary Tablespace](#)
- [Specifying a Profile](#)
- [Setting Default Roles](#)

See Also: ["Granting System Privileges and Roles"](#) on page 25-11

Specifying a Name

Within each database a user name must be unique with respect to other user names and roles. A user and role cannot have the same name. Furthermore, each user has an associated schema. Within a schema, each schema object must have a unique name.

Setting a User's Authentication

In the previous `CREATE USER` statement, the new user is to be authenticated using the database. In this case, the connecting user must supply the correct password to the database to connect successfully.

Selecting and specifying the method of user authentication is discussed in ["User Authentication Methods"](#) on page 24-8.

Assigning a Default Tablespace

Each user should have a default tablespace. When a user creates a schema object and specifies no tablespace to contain it, Oracle stores the object in the user's default tablespace.

The default setting for every user's default tablespace is the `SYSTEM` tablespace. If a user does not create objects, and has no privileges to do so, this default setting is fine. However, if a user creates any type of object, you should specifically assign the user a default tablespace. Using a tablespace other than `SYSTEM` reduces contention between data dictionary objects and user objects for the same datafiles. In general, it is not advisable for user data to be stored in the `SYSTEM` tablespace.

You can set a user's default tablespace during user creation, and change it later with the `ALTER USER` statement. Changing the user's default tablespace affects only objects created after the setting is changed.

When you specify the user's default tablespace, also specify a quota on that tablespace.

In the previous `CREATE USER` statement, `jward`'s default tablespace is `data_ts`, and his quota on that tablespace is `500K`.

Assigning Tablespace Quotas

You can assign each user a tablespace quota for any tablespace (except a temporary tablespace). Assigning a quota does two things:

- Users with privileges to create certain types of objects can create those objects in the specified tablespace.
- Oracle limits the amount of space that can be allocated for storage of a user's objects within the specified tablespace to the amount of the quota.

By default, a user has no quota on any tablespace in the database. If the user has the privilege to create a schema object, you must assign a quota to allow the user to create objects. Minimally, assign users a quota for the default tablespace, and additional quotas for other tablespaces in which they can create objects.

You can assign a user either individual quotas for a specific amount of disk space in each tablespace or an unlimited amount of disk space in all tablespaces. Specific quotas prevent a user's objects from consuming too much space in the database.

You can assign a user's tablespace quotas when you create the user, or add or change quotas later. If a new quota is less than the old one, then the following conditions hold true:

- If a user has already exceeded a new tablespace quota, the user's objects in the tablespace cannot be allocated more space until the combined space of these objects falls below the new quota.
- If a user has not exceeded a new tablespace quota, or if the space used by the user's objects in the tablespace falls under a new tablespace quota, the user's objects can be allocated space up to the new quota.

Revoking Users Ability to Create Objects in a Tablespace You can revoke a user's ability to create objects in a tablespace by changing the user's current quota to zero. After a quota of zero is assigned, the user's objects in the tablespace remain, but new objects cannot be created and existing objects cannot be allocated any new space.

UNLIMITED TABLESPACE System Privilege To permit a user to use an unlimited amount of any tablespace in the database, grant the user the `UNLIMITED TABLESPACE` system privilege. This overrides all explicit tablespace quotas for the user. If you later revoke the privilege, explicit quotas again take effect. You can grant this privilege only to users, not to roles.

Before granting the `UNLIMITED TABLESPACE` system privilege, consider the consequences of doing so.

Advantage:

- You can grant a user unlimited access to all tablespaces of a database with one statement.

Disadvantages:

- The privilege overrides all explicit tablespace quotas for the user.
- You cannot selectively revoke tablespace access from a user with the `UNLIMITED TABLESPACE` privilege. You can grant access selectively only after revoking the privilege.

Assigning a Temporary Tablespace

Each user also should be assigned a temporary tablespace. When a user executes a SQL statement that requires a temporary segment, Oracle stores the segment in the user's temporary tablespace. These temporary segments are created by the system when doing sorts or joins and are owned by `SYS`, which has resource privileges in all tablespaces.

In the previous `CREATE USER` statement, `jward`'s temporary tablespace is `temp_ts`, a tablespace created explicitly to contain only temporary segments. Such a tablespace is created using the `CREATE TEMPORARY TABLESPACE` statement.

If a user's temporary tablespace is not explicitly set, the user is assigned the default temporary tablespace that was specified at database creation, or by an `ALTER DATABASE` statement at a later time. If there is no default temporary tablespace, the default is the `SYSTEM` tablespace. It is not advisable for user data to be stored in the `SYSTEM` tablespace. Also, assigning a tablespace to be used specifically as a temporary tablespace eliminates file contention among temporary segments and other types of segments.

Note: If your `SYSTEM` tablespace is a locally managed tablespace, then users must be assigned a specific default (locally managed) temporary tablespace and not be allowed to default to using the `SYSTEM` tablespace. This is because temporary objects cannot be placed in permanent locally managed tablespaces.

You can set a user's temporary tablespace at user creation, and change it later using the `ALTER USER` statement. Do not set a quota for temporary tablespaces.

See Also:

- ["Temporary Tablespaces"](#) on page 11-12
- ["Creating a Default Temporary Tablespace"](#) on page 2-24

Specifying a Profile

You also specify a profile when you create a user. A profile is a set of limits on database resources and password access to the database. If no profile is specified, the user is assigned a default profile.

See Also:

- ["Managing Resources with Profiles"](#) on page 24-18
- ["Password Management Policy"](#) on page 23-12

Setting Default Roles

You cannot set a user's default roles in the `CREATE USER` statement. When you first create a user, the user's default role setting is `ALL`, which causes all roles subsequently granted to the user to be default roles. Use the `ALTER USER` statement to change the user's default roles.

See Also: ["Specifying Default Roles"](#) on page 25-21

Altering Users

Users can change their own passwords. However, to change any other option of a user's security domain, you must have the `ALTER USER` system privilege. Security administrators are normally the only users that have this system privilege, as it allows a modification of *any* user's security domain. This privilege includes the ability to set tablespace quotas for a user on any tablespace in the database, even if the user performing the modification does not have a quota for a specified tablespace.

You can alter a user's security settings with the `ALTER USER` statement. Changing a user's security settings affects the user's future sessions, not current sessions.

The following statement alters the security settings for user `avyrros`:

```
ALTER USER avyrros
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE data_ts
  TEMPORARY TABLESPACE temp_ts
  QUOTA 100M ON data_ts
```

```
QUOTA 0 ON test_ts  
PROFILE clerk;
```

The `ALTER USER` statement here changes `avyrros`'s security settings as follows:

- Authentication is changed to use `avyrros`'s operating system account.
- `avyrros`'s default and temporary tablespaces are explicitly set.
- `avyrros` is given a 100M quota for the `data_ts` tablespace.
- `avyrros`'s quota on the `test_ts` is revoked.
- `avyrros` is assigned the `clerk` profile.

Changing a User's Authentication Mechanism

Most non-DBA users can still change their own passwords with the `ALTER USER` statement, as follows:

```
ALTER USER andy  
IDENTIFIED BY swordfish;
```

No special privileges (other than those to connect to the database) are required for a user to change passwords. Users should be encouraged to change their passwords frequently.

Users must have the `ALTER USER` privilege to switch between methods of authentication. Usually, only an administrator has this privilege.

See Also: ["User Authentication Methods"](#) on page 24-8 for information about the authentication methods that are available for Oracle users

Changing a User's Default Roles

A default role is one that is automatically enabled for a user when the user creates a session. You can assign a user zero or more default roles.

See Also: [Chapter 25, "Managing User Privileges and Roles"](#) for information about changing users' default roles

Dropping Users

When a user is dropped, the user and associated schema are removed from the data dictionary and all schema objects contained in the user's schema, if any, are immediately dropped.

Note: If a user's schema and associated objects must remain but the user must be denied access to the database, revoke the `CREATE SESSION` privilege from the user.

A user that is currently connected to a database cannot be dropped. To drop a connected user, you must first terminate the user's sessions using the SQL statement `ALTER SYSTEM` with the `KILL SESSION` clause.

You can drop a user from a database using the `DROP USER` statement. To drop a user and all the user's schema objects (if any), you must have the `DROP USER` system privilege. Because the `DROP USER` system privilege is so powerful, a security administrator is typically the only type of user that has this privilege.

If the user's schema contains any schema objects, use the `CASCADE` option to drop the user and all associated objects and foreign keys that depend on the tables of the user successfully. If you do not specify `CASCADE` and the user's schema contains objects, an error message is returned and the user is not dropped. Before dropping a user whose schema contains objects, thoroughly investigate which objects the user's schema contains and the implications of dropping them. Pay attention to any unknown cascading effects. For example, if you intend to drop a user who owns a table, check whether any views or procedures depend on that particular table.

The following statement drops user `jones` and all associated objects and foreign keys that depend on the tables owned by `jones`.

```
DROP USER jones CASCADE;
```

See Also: "[Terminating Sessions](#)" on page 5-21 for more information about terminating sessions

User Authentication Methods

Oracle provides several means for users to be authenticated before they are allowed to create a database session:

1. You can define users such that the database performs both identification and authentication of users. This is called **database authentication**.
2. You can define users such that authentication is performed by the operating system or network service. This is called **external authentication**.
3. You can define users such that they are authenticated **globally** by SSL (Secure Sockets Layer). These users are called **global users**. For global users, an

enterprise directory can be used to authorize their access to the database through **global roles**.

4. You can specify users who are allowed to connect through a middle-tier server. The middle-tier server authenticates and assumes the identity of the user and is allowed to enable specific roles for the user. This is called **proxy authentication** and authorization.

These means of authentication are discussed in the following sections:

- [Database Authentication](#)
- [External Authentication](#)
- [Global Authentication and Authorization](#)
- [Proxy Authentication and Authorization](#)

Database Authentication

If you choose database authentication for a user, administration of the user account including authentication of that user is performed entirely by Oracle. To have Oracle authenticate a user, specify a password for the user when you create or alter the user. Users can change their password at any time. Passwords are stored in an encrypted format. Each password must be made up of single-byte characters, even if your database uses a multibyte character set.

Note: Oracle Corporation recommends that you encode user names and passwords in ASCII or EBCDIC characters only, depending on your platform. This will maintain compatibility for supporting future changes to your database character set.

If user names or passwords are created based on characters that will have size expansion when migrating to a new target character set, then users can experience login difficulties due to authentication failures after the migration. This is because the encrypted user names and passwords stored in the data dictionary do not get updated during the migration to the new database character set.

For example, assuming the current database character set is WE8MSWIN1252 and the target database character set is UTF8, the user name `scött` (o with an umlaut) will change from 5 bytes to 6 bytes in UTF8. the user `scött` will no longer be able to log in.

If user names and passwords are not based on ASCII or EBCDIC characters, then the affected user names and passwords will need to be reset upon the migration to a new character set.

To enhance security when using database authentication, Oracle recommends the use of password management, including account locking, password aging and expiration, password history, and password complexity verification.

See Also: ["Password Management Policy"](#) on page 23-12

Creating a User Who is Authenticated by the Database

The following statement creates a user who is identified and authenticated by Oracle. User `scott` must specify the password `tiger` whenever connecting to Oracle.

```
CREATE USER scott IDENTIFIED BY tiger;
```

See Also: *Oracle9i SQL Reference* for more information about valid passwords, and how to specify the `IDENTIFIED BY` clause in the `CREATE USER` and `ALTER USER` statements

Advantages of Database Authentication

Following are advantages of database authentication:

- User accounts and all authentication are controlled by the database. There is no reliance on anything outside of the database.
- Oracle provides strong password management features to enhance security when using database authentication.
- It is easier to administer when there are small user communities.

External Authentication

When you choose external authentication for a user, the user account is maintained by Oracle, but password administration and user authentication is performed by an external service. This external service can be the operating system or a network service, such as Oracle Net.

With external authentication, your database relies on the underlying operating system or network authentication service to restrict access to database accounts. A database password is not used for this type of login. If your operating system or network service permits, you can have it authenticate users. If you do so, set the initialization parameter `OS_AUTHENT_PREFIX`, and use this prefix in Oracle user names. The `OS_AUTHENT_PREFIX` parameter defines a prefix that Oracle adds to the beginning of every user's operating system account name. Oracle compares the prefixed user name with the Oracle user names in the database when a user attempts to connect.

For example, assume that `OS_AUTHENT_PREFIX` is set as follows:

```
OS_AUTHENT_PREFIX=OPS$
```

Note: The text of the `OS_AUTHENT_PREFIX` initialization parameter is case sensitive on some operating systems. See your operating system specific Oracle documentation for more information about this initialization parameter.

If a user with an operating system account named `tsmith` is to connect to an Oracle database and be authenticated by the operating system, Oracle checks that there is a corresponding database user `OPS$tsmith` and, if so, allows the user to connect. All references to a user authenticated by the operating system must include the prefix, as seen in `OPS$tsmith`.

The default value of this parameter is `OPS$` for backward compatibility with previous versions of Oracle. However, you might prefer to set the prefix value to some other string or a null string (an empty set of double quotes: `''`). Using a null

string eliminates the addition of any prefix to operating system account names, so that Oracle user names exactly match operating system user names.

After you set `OS_AUTHENT_PREFIX`, it should remain the same for the life of a database. If you change the prefix, any database user name that includes the old prefix cannot be used to establish a connection, unless you alter the user name to have it use password authentication.

Creating a User Who is Authenticated Externally

The following statement creates a user who is identified by Oracle and authenticated by the operating system or a network service. This example assumes that `OS_AUTHENT_PREFIX = ""`.

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

Using `CREATE USER ... IDENTIFIED EXTERNALLY`, you create database accounts that must be authenticated by the operating system or network service. Oracle relies on this external login authentication to ensure that a specific operating system user has access to a specific database user.

See Also: *Oracle Advanced Security Administrator's Guide* for more information about external authentication

Operating System Authentication

By default, Oracle only allows operating system authenticated logins over secure connections. Therefore, if you want the operating system to authenticate a user, by default that user cannot connect to the database over Oracle Net. This means the user cannot connect using a shared server configuration, since this connection uses Oracle Net. This default restriction prevents a remote user from impersonating another operating system user over a network connection.

If you are not concerned about remote users impersonating another operating system user over a network connection, and you want to use operating system user authentication with network clients, set the initialization parameter `REMOTE_OS_AUTHENT` (default is `FALSE`) to `TRUE` in the database's initialization parameter file. Setting the initialization parameter `REMOTE_OS_AUTHENT` to `TRUE` allows the RDBMS to accept the client operating system user name received over a nonsecure connection and use it for account access. The change takes effect the next time you start the instance and mount the database.

Generally, user authentication through the host operating system offers the following benefits:

- Users can connect to Oracle faster and more conveniently without specifying a separate database user name or password.
- User entries in the database and operating system audit trails correspond.

Network Authentication

Network authentication is performed using Oracle Advanced Security, which can be configured to use a third party service such as Kerberos. If you are using Oracle Advanced Security as your only external authentication service, the setting of the parameter `REMOTE_OS_AUTHENT` is irrelevant, since Oracle Advanced Security only allows secure connections.

Advantages of External Authentication

Following are advantages of external authentication:

- More choices of authentication mechanism are available, such as smart cards, fingerprints, Kerberos, or the operating system.
- Many network authentication services, such as Kerberos and DCE, support single sign-on. This means that users have fewer passwords to remember.
- If you are already using some external mechanism for authentication, such as one of those listed above, there may be less administrative overhead to use that mechanism with the database as well.

Global Authentication and Authorization

Oracle Advanced Security enables you to centralize management of user-related information, including authorizations, in an LDAP-based directory service. Users can be identified in the database as global users, meaning that they are authenticated by SSL and that the management of these users is done outside of the database by the centralized directory service. Global roles are defined in a database and are known only to that database, but authorizations for such roles is done by the directory service.

Note: You can also have users authenticated by SSL, whose authorizations are not managed in a directory; that is, they have local database roles only. See the *Oracle Advanced Security Administrator's Guide* for details.

This centralized management enables the creation of **enterprise users** and **enterprise roles**. Enterprise users are defined and managed in the directory. They have unique identities across the enterprise, and can be assigned enterprise roles that determine their access privileges across multiple databases. An enterprise role consists of one or more global roles, and might be thought of as a container for global roles.

Creating a User Who is Authorized by a Directory Service

You have a couple of options as to how you specify users who are authorized by a directory service.

Creating a Global User The following statement illustrates the creation of a global user, who is authenticated by SSL and authorized by the enterprise directory service:

```
CREATE USER scott
    IDENTIFIED GLOBALLY AS 'CN=scott,OU=division1,O=oracle,C=US';
```

The string provided in the **AS** clause provides an identifier (**distinguished name**, or **DN**) meaningful to the enterprise directory.

In this case, `scott` is truly a global user. But, the disadvantage here is that user `scott` must then be created in every database that he must access, plus the directory.

Creating a Schema-Independent User Creating schema-independent users allows multiple enterprise users to access a shared schema in the database. A schema-independent user is:

- Authenticated by SSL or passwords
- *Not* created in the database with a `CREATE USER` statement of any type
- A user whose privileges are managed in a directory
- A user who connects to a shared schema

The process of creating a schema-independent user is as follows:

1. Create a shared schema in the database as follows.

```
CREATE USER appschema IDENTIFIED GLOBALLY AS '';
```

2. In the directory, you now create multiple enterprise users, and a mapping object.

The mapping object tells the database how you want to map users' DNs to the shared schema. You can either do a full DN mapping (one directory entry for each unique DN), or you can map, for example, every user containing the following DN components to the `appschema`:

```
OU=division,O=Oracle,C=US
```

See the *Oracle Internet Directory Administrator's Guide* for an explanation of these mappings.

Most users do not need their own schemas, and implementing schema-independent users divorces users from databases. You create multiple users who share the same schema in a database, and as enterprise users, they can access shared schemas in other databases as well.

Advantages of Global Authentication and Global Authorization

Some of the advantages of global user authentication and authorization are the following:

- Provides strong authentication using SSL or Windows NT native authentication
- Enables centralized management of users and privileges across the enterprise
- Is easy to administer—for every user you do not have to create a schema in every database in the enterprise
- Facilitates single sign-on—users only need to sign on once to access multiple databases and services. Further, users using passwords can have a single password to access databases accepting password authenticated enterprise users.
- Because it provides password based access, previously defined password authenticated database users can be migrated to the directory (using the User Migration Utility) to be centrally administered. This makes global authentication and authorization available for prior Oracle release clients that are still supported.
- `CURRENT_USER` database links connect as a global user. A local user can connect as a global user in the context of a stored procedure—without storing the global user's password in a link definition.

See Also: The following books contain additional information about global authentication and authorization, and enterprise users and roles:

- *Oracle Advanced Security Administrator's Guide*
- *Oracle Internet Directory Administrator's Guide*

Proxy Authentication and Authorization

It is possible to design a middle-tier server to proxy clients in a secure fashion.

Oracle provides three forms of proxy authentication:

- The middle-tier server authenticates itself with the database server and a client, in this case an application user or another application, authenticates itself with the middle-tier server. Client identities can be maintained all the way through to the database.
- The client, in this case a database user, is not authenticated by the middle-tier server. The client's identity and database password are passed through the middle-tier server to the database server for authentication.
- The client, in this case a global user, is authenticated by the middle-tier server, and passes one of the following through the middle tier for retrieving the client's user name.
 - Distinguished name (DN)
 - Certificate

In all cases, the middle-tier server must be authorized to act on behalf of the client by the administrator.

To authorize a middle-tier server to proxy a client use the `GRANT CONNECT THROUGH` clause of the `ALTER USER` statement. You can also specify roles that the middle tier is permitted to activate when connecting as the client.

Operations done on behalf of a client by a middle-tier server can be audited.

The `PROXY_USERS` data dictionary view can be queried to see which users are currently authorized to connect through a middle tier.

Use the `REVOKE CONNECT THROUGH` clause of `ALTER USER` to disallow a proxy connection.

See Also:

- *Oracle Call Interface Programmer's Guide* and *Oracle9i Application Developer's Guide - Fundamentals* for details about designing a middle-tier server to proxy users
- *Oracle9i SQL Reference* for a description and syntax of the proxy clause for `ALTER USER`
- ["Auditing in a Multi-Tier Environment"](#) on page 26-13 for details of auditing operations done on behalf of a user by a middle tier

Authorizing a Middle Tier to Proxy and Authenticate a User

The following statement authorizes the middle-tier server `appserve` to connect as user `bill`. It uses the `WITH ROLE` clause to specify that `appserve` activate all roles associated with `bill`, except `payroll`.

```
ALTER USER bill
  GRANT CONNECT THROUGH appserve
  WITH ROLE ALL EXCEPT payroll;
```

To revoke the middle-tier server's (`appserve`) authorization to connect as user `bill`, the following statement is used:

```
ALTER USER bill REVOKE CONNECT THROUGH appserve;
```

Authorizing a Middle Tier to Proxy a User Authenticated by Other Means

Use the `AUTHENTICATED USING` clause of the `ALTER USER ... GRANT CONNECT THROUGH` statement to authorize a user to be proxied, but not authenticated, by a middle tier. Currently, `PASSWORD` is the only means supported.

The following statement illustrates this form of authentication:

```
ALTER USER mary
  GRANT CONNECT THROUGH midtier
  AUTHENTICATED USING PASSWORD;
```

In the above statement, middle-tier server `midtier` is authorized to connect as `mary`, and `midtier` must also pass `mary`'s password to the database server for authorization.

Authorizing a Middle Tier to Proxy a User Identified by a Distinguished Name

In this case, the following statement authorizes the middle-tier server `WebDB` to present the distinguished name for global user `jeff` to the database server. The distinguished name is used to retrieve the user name. User `jeff` has been authenticated by the middle-tier server `WebDB`.

```
ALTER USER jeff
  GRANT CONNECT THROUGH WebDB
  AUTHENTICATED USING DISTINGUISHED NAME;
```

Optionally, the middle-tier server can be authorized to present an entire certificate (containing the distinguished name). This is illustrated in the following statement:

```
ALTER USER jeff
  GRANT CONNECT THROUGH WebDB
  AUTHENTICATED USING CERTIFICATE;
```

Passing the entire certificate costs time in authentication. However, some applications use other information contained in the certificate.

Managing Resources with Profiles

A profile is a named set of resource limits. A user's profile limits database usage and instance resources as defined in the profile. You can assign a profile to each user, and a default profile to all users who do not have specific profiles. For profiles to take effect, resource limits must be turned on for the database as a whole.

This section describes aspects of profile management, and contains the following topics:

- [Enabling and Disabling Resource Limits](#)
- [Creating Profiles](#)
- [Assigning Profiles](#)
- [Altering Profiles](#)
- [Using Composite Limits](#)
- [Dropping Profiles](#)

See Also: *Oracle9i SQL Reference* for more information about the SQL statements used for managing profiles

Enabling and Disabling Resource Limits

A profile can be created, assigned to users, altered, and dropped at any time by any authorized database user, but the resource limits set for a profile are enforced only when you enable resource limitation for the associated database. Resource limitation enforcement can be enabled or disabled by two different methods, as described in the next two sections.

To alter the enforcement of resource limitation while the database remains open, you must have the `ALTER SYSTEM` system privilege.

Enabling and Disabling Resource Limits Before Startup

If a database can be temporarily shut down, resource limitation can be enabled or disabled by the `RESOURCE_LIMIT` initialization parameter in the database's initialization parameter file. Valid values for the parameter are `TRUE` (enables enforcement) and `FALSE`. By default, this parameter's value is set to `FALSE`. Once the initialization parameter file has been edited, the database instance must be restarted to take effect. Every time an instance is started, the new parameter value enables or disables the enforcement of resource limitation.

Enabling and Disabling Resource Limits While the Database is Open

If a database cannot be temporarily shut down or the resource limitation feature must be altered temporarily, you can enable or disable the enforcement of resource limitation using the SQL statement `ALTER SYSTEM`. After an instance is started, an `ALTER SYSTEM` statement overrides the value set by the `RESOURCE_LIMIT` initialization parameter. For example, the following statement enables the enforcement of resource limitation for a database:

```
ALTER SYSTEM
  SET RESOURCE_LIMIT = TRUE;
```

Note: This does not apply to password parameters.

An `ALTER SYSTEM` statement does not permanently determine the enforcement of resource limitation. If the database is shut down and restarted, the enforcement of resource limits is determined by the value set for the `RESOURCE_LIMIT` parameter.

Creating Profiles

To create a profile, you must have the `CREATE PROFILE` system privilege. You can create profiles using the SQL statement `CREATE PROFILE`. At the same time, you can explicitly set particular resource limits.

The following statement creates the profile `clerk`:

```
CREATE PROFILE clerk LIMIT
  SESSIONS_PER_USER 2
  CPU_PER_SESSION unlimited
  CPU_PER_CALL 6000
  LOGICAL_READS_PER_SESSION unlimited
  LOGICAL_READS_PER_CALL 100
  IDLE_TIME 30
  CONNECT_TIME 480;
```

All unspecified resource limits for a new profile take the limit set by a `DEFAULT` profile.

Each database has a `DEFAULT` profile, and its limits are used in two cases:

- If a user is not explicitly assigned a profile, then the user conforms to *all* the limits of the `DEFAULT` profile.
- All unspecified limits of any profile use the corresponding limit of the `DEFAULT` profile.

Initially, all limits of the `DEFAULT` profile are set to `UNLIMITED`. However, to prevent unlimited resource consumption by users of the `DEFAULT` profile, the security administrator should change the default limits using the `ALTER PROFILE` statement:

```
ALTER PROFILE default LIMIT
  ...;
```

Any user with the `ALTER PROFILE` system privilege can adjust the limits in the `DEFAULT` profile. The `DEFAULT` profile cannot be dropped.

Assigning Profiles

After a profile has been created, you can assign it to database users. Each user can be assigned only one profile at any given time. If a profile is assigned to a user who already has a profile, the new profile assignment overrides the previously assigned profile. Profile assignments do not affect current sessions. Profiles can be assigned only to users and not to roles or other profiles.

Profiles can be assigned to users with the `CREATE USER` and `ALTER USER` statements.

See Also:

- ["Creating Users"](#) on page 24-2
- ["Altering Users"](#) on page 24-6

Altering Profiles

You can alter the resource limit settings of any profile using the SQL statement `ALTER PROFILE`. To alter a profile, you must have the `ALTER PROFILE` system privilege.

Any adjusted profile limit overrides the previous setting for that profile limit. By adjusting a limit with a value of `DEFAULT`, the resource limit reverts to the default limit set for the database. All profiles not adjusted when altering a profile retain the previous settings. Any changes to a profile do not affect current sessions. New profile settings are used only for sessions created after a profile is modified.

The following statement alters the `clerk` profile:

```
ALTER PROFILE clerk LIMIT
  CPU_PER_CALL default
  LOGICAL_READS_PER_SESSION 20000;
```

Using Composite Limits

In addition to being able to use the `CREATE` or `ALTER PROFILE` statements to assign resource limits to specific resources, you can limit the total resource cost for a session by using composite limits. A composite limit is expressed as a weighted sum, measured in **service units**, of certain resources.

You can set a profile's composite limit using the `COMPOSITE_LIMIT` clause of a `CREATE PROFILE` or `ALTER PROFILE` statement. The following `CREATE PROFILE` statement specifies the `COMPOSITE_LIMIT` clause:

```
CREATE PROFILE clerk LIMIT
  COMPOSITE_LIMIT 20000
  SESSIONS_PER_USER 2
  CPU_PER_CALL 1000;
```

Notice that both explicit resource limits and a composite limit can exist concurrently for a profile. The limit that is reached first stops the activity in a session. Composite limits allow additional flexibility when limiting the use of system resources.

Determining the Value of the Composite Limit

The correct composite limit depends on the total amount of resource used by an average profile user. As with each specific resource limit, historical information should be gathered to determine the normal range of composite resource usage for a typical profile user.

See Also: *Oracle9i SQL Reference* for information on how to calculate the composite limit

Setting Resource Costs

Each Oracle database server environment has its own characteristics. Some system resources can be more valuable in one environment than another. Oracle enables you to assign the following resources a weight, which then affects their contribution to a total resource cost:

- CPU_PER_SESSION
- LOGICAL_READS_PER_SESSION
- CONNECT_TIME
- PRIVATE_SGA.

If you do not assign a weight to a resource, its weight defaults to 0, and the use of the resource does not contribute to the total resource cost.

Oracle calculates the total resource cost by first multiplying the amount of each resource used in the session by the resource's weight, and then summing the products for all four resources. For any session, this cost is limited by the value of the `COMPOSITE_LIMIT` parameter in the user's profile. Both the products and the total cost are expressed in units called service units.

To set weights for resources, use the `ALTER RESOURCE COST` statement. You must have the `ALTER RESOURCE` system privilege. The following example assigns weights to the `CPU_PER_SESSION` and `LOGICAL_READS_PER_SESSION` resources.

```
ALTER RESOURCE COST
  CPU_PER_SESSION 1
  LOGICAL_READS_PER_SESSION 50;
```

The weights establish this cost formula for a session:

$$\text{cost} = (1 * \text{CPU_PER_SESSION}) + (50 * \text{LOGICAL_READS_PER_SESSION})$$

where the values of `CPU_PER_SESSION` and `LOGICAL_READS_PER_SESSION` are either values in the `DEFAULT` profile or in the profile of the user of the session.

Because the above statement assigns no weight to the resources `CONNECT_TIME` and `PRIVATE_SGA`, these resources do not appear in the formula.

See Also:

- *Oracle9i SQL Reference*
- Your operating system specific Oracle documentation

The above sources provide additional information and recommendations on setting resource costs

Dropping Profiles

To drop a profile, you must have the `DROP PROFILE` system privilege. You can drop a profile using the SQL statement `DROP PROFILE`. To successfully drop a profile currently assigned to a user, use the `CASCADE` option.

The following statement drops the profile `clerk`, even though it is assigned to a user:

```
DROP PROFILE clerk CASCADE;
```

Any user currently assigned to a profile that is dropped is automatically assigned to the `DEFAULT` profile. The `DEFAULT` profile cannot be dropped. When a profile is dropped, the drop does not affect currently active sessions. Only sessions created after a profile is dropped abide by any modified profile assignments.

Viewing Information About Database Users and Profiles

The following data dictionary views contain information about database users and profiles:

View	Description
DBA_USERS ALL_USERS USER_USERS	DBA view describes all users of the database. ALL view lists users visible to the current user, but does not describe them. USER view describes only the current user.
DBA_TS_QUOTAS USER_TS_QUOTAS	Describes tablespace quotas for users.

View	Description
USER_PASSWORD_LIMITS	Describes the password profile parameters that are assigned to the user.
USER_RESOURCE_LIMITS	Displays the resource limits for the current user.
DBA_PROFILES	Displays all profiles and their limits.
RESOURCE_COST	Lists the cost for each resource.
V\$SESSION	Lists session information for each current session. Includes user name.
V\$SESSTAT	Lists user session statistics.
V\$STATNAME	Displays decoded statistic names for the statistics shown in the V\$SESSTAT view.
PROXY_USERS	Describes users who can assume the identity of other users.

The following sections present some example of using these views, and assume a database in which the following statements have been executed:

```
CREATE PROFILE clerk LIMIT
  SESSIONS_PER_USER 1
  IDLE_TIME 30
  CONNECT_TIME 600;
```

```
CREATE USER jfee
  IDENTIFIED BY wildcat
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp_ts
  QUOTA 500K ON users
  PROFILE clerk;
```

```
CREATE USER dcranney
  IDENTIFIED BY bedrock
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp_ts
  QUOTA unlimited ON users;
```

```
CREATE USER userscott
  IDENTIFIED BY scott1;
```

See Also: *Oracle9i SQL Reference* for complete descriptions of the above data dictionary and dynamic performance views

Listing All Users and Associated Information

The following query lists users and their associated information as defined in the database:

```
SELECT USERNAME, PROFILE, ACCOUNT_STATUS FROM DBA_USERS;
```

USERNAME	PROFILE	ACCOUNT_STATUS
SYS	DEFAULT	OPEN
SYSTEM	DEFAULT	OPEN
USERSCOTT	DEFAULT	OPEN
JFEE	CLERK	OPEN
DCRANNEY	DEFAULT	OPEN

All passwords are encrypted to preserve security. If a user queries the `PASSWORD` column, that user is not be able to determine another user's password.

Listing All Tablespace Quotas

The following query lists all tablespace quotas specifically assigned to each user:

```
SELECT * FROM DBA_TS_QUOTAS;
```

TABLESPACE	USERNAME	BYTES	MAX_BYTES	BLOCKS	MAX_BLOCKS
USERS	JFEE	0	512000	0	250
USERS	DCRANNEY	0	-1	0	-1

When specific quotas are assigned, the exact number is indicated in the `MAX_BYTES` column. Note that this number is always a multiple of the database block size, so if you specify a tablespace quota that is not a multiple of the database block size, it is rounded up accordingly. Unlimited quotas are indicated by "-1".

Listing All Profiles and Assigned Limits

The following query lists all profiles in the database and associated settings for each limit in each profile:

```
SELECT * FROM DBA_PROFILES
ORDER BY PROFILE;
```

PROFILE	RESOURCE_NAME	RESOURCE	LIMIT
CLERK	COMPOSITE_LIMIT	KERNEL	DEFAULT

CLERK	FAILED_LOGIN_ATTEMPTS	PASSWORD	DEFAULT
CLERK	PASSWORD_LIFE_TIME	PASSWORD	DEFAULT
CLERK	PASSWORD_REUSE_TIME	PASSWORD	DEFAULT
CLERK	PASSWORD_REUSE_MAX	PASSWORD	DEFAULT
CLERK	PASSWORD_VERIFY_FUNCTION	PASSWORD	DEFAULT
CLERK	PASSWORD_LOCK_TIME	PASSWORD	DEFAULT
CLERK	PASSWORD_GRACE_TIME	PASSWORD	DEFAULT
CLERK	PRIVATE_SGA	KERNEL	DEFAULT
CLERK	CONNECT_TIME	KERNEL	600
CLERK	IDLE_TIME	KERNEL	30
CLERK	LOGICAL_READS_PER_CALL	KERNEL	DEFAULT
CLERK	LOGICAL_READS_PER_SESSION	KERNEL	DEFAULT
CLERK	CPU_PER_CALL	KERNEL	DEFAULT
CLERK	CPU_PER_SESSION	KERNEL	DEFAULT
CLERK	SESSIONS_PER_USER	KERNEL	1
DEFAULT	COMPOSITE_LIMIT	KERNEL	UNLIMITED
DEFAULT	PRIVATE_SGA	KERNEL	UNLIMITED
DEFAULT	SESSIONS_PER_USER	KERNEL	UNLIMITED
DEFAULT	CPU_PER_CALL	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	KERNEL	UNLIMITED
DEFAULT	CONNECT_TIME	KERNEL	UNLIMITED
DEFAULT	IDLE_TIME	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	CPU_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	FAILED_LOGIN_ATTEMPTS	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LIFE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_MAX	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LOCK_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_GRACE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_VERIFY_FUNCTION	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_TIME	PASSWORD	UNLIMITED

32 rows selected.

Viewing Memory Use for Each User Session

The following query lists all current sessions, showing the Oracle user and current UGA (user global area) memory use for each session:

```
SELECT USERNAME, VALUE || 'bytes' "Current UGA memory"
  FROM V$SESSION sess, V$SESSTAT stat, V$STATNAME name
 WHERE sess.SID = stat.SID
       AND stat.STATISTIC# = name.STATISTIC#
       AND name.NAME = 'session uga memory';
```

USERNAME	Current UGA memory
-----	-----
	18636bytes
	17464bytes
	19180bytes
	18364bytes
	39384bytes
	35292bytes
	17696bytes
	15868bytes
USERSCOTT	42244bytes
SYS	98196bytes
SYSTEM	30648bytes

11 rows selected.

To see the maximum UGA memory ever allocated to each session since the instance started, replace 'session uga memory' in the query above with 'session uga memory max'.

Managing User Privileges and Roles

This chapter explains how to use privileges and roles to control access to schema objects and to control the ability to execute system operations. The following topics are discussed:

- [Understanding User Privileges and Roles](#)
- [Managing User Roles](#)
- [Granting User Privileges and Roles](#)
- [Revoking User Privileges and Roles](#)
- [Granting to and Revoking from the User Group PUBLIC](#)
- [When Do Grants and Revokes Take Effect?](#)
- [Granting Roles Using the Operating System or Network](#)
- [Viewing Privilege and Role Information](#)

See Also:

- [Chapter 24, "Managing Users and Resources"](#) for information about controlling access to a database
- [Chapter 23, "Establishing Security Policies"](#) for suggested general database security policies

Understanding User Privileges and Roles

A user **privilege** is a right to execute a particular type of SQL statement, or a right to access another user's object. The types of privileges are defined by Oracle.

Roles, on the other hand, are created by users (usually administrators) and are used to group together privileges or other roles. They are a means of facilitating the granting of multiple privileges or roles to users.

This section describes Oracle user privileges, and contains the following topics:

- [System Privileges](#)
- [Object Privileges](#)
- [User Roles](#)

See Also: *Oracle9i Database Concepts* for additional information about privileges and roles

System Privileges

There are over 100 distinct system privileges. Each system privilege allows a user to perform a particular database operation or class of database operations.

Caution: System privileges can be very powerful, and should be granted only when necessary to roles and trusted users of the database.

See Also: *Oracle9i SQL Reference*, for the complete list of system privileges and their descriptions

Restricting System Privileges

Because system privileges are so powerful, Oracle recommends that you configure your database to prevent regular (non-DBA) users exercising ANY system privileges (such as UPDATE ANY TABLE) on the data dictionary. In order to secure the data dictionary, ensure that the O7_DICTIONARY_ACCESSIBILITY initialization parameter is set to FALSE. This feature is called the dictionary protection mechanism.

Note: The `O7_DICTIONARY_ACCESSIBILITY` initialization parameter controls restrictions on system privileges when you upgrade from Oracle7 to Oracle8i and higher releases. If the parameter is set to `TRUE`, access to objects in the `SYS` schema is allowed (Oracle7 behavior). If this parameter is set to `FALSE`, system privileges that allow access to objects in "any schema" do not allow access to objects in `SYS` schema. The default for `O7_DICTIONARY_ACCESSIBILITY` is `FALSE`.

When this parameter is not set to `FALSE`, the `ANY` privilege applies to the data dictionary, and a malicious user with `ANY` privilege could access or alter data dictionary tables.

See the *Oracle9i Database Reference* for more information on the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter and *Oracle9i Database Migration* to understand its usage.

If you enable dictionary protection (`O7_DICTIONARY_ACCESSIBILITY` is `FALSE`), access to objects in the `SYS` schema (dictionary objects) is restricted to users with the `SYS` schema. These users are `SYS` and those who connect as `SYSDBA`. System privileges providing access to objects in other schemas do *not* give other users access to objects in the `SYS` schema. For example, the `SELECT ANY TABLE` privilege allows users to access views and tables in other schemas, but does not enable them to select dictionary objects (base tables of dynamic performance views, views, packages, and synonyms). These users can, however, be granted explicit object privileges to access objects in the `SYS` schema.

Accessing Objects in the SYS Schema

Users with explicit object privileges or those who connect with administrative privileges (`SYSDBA`) can access objects in the `SYS` schema. Another means of allowing access to objects in the `SYS` schema is by granting users any of the following roles:

- `SELECT_CATALOG_ROLE`

This role can be granted to users to allow `SELECT` privileges on all data dictionary views.

- `EXECUTE_CATALOG_ROLE`

This role can be granted to users to allow `EXECUTE` privileges for packages and procedures in the data dictionary.

- `DELETE_CATALOG_ROLE`

This role can be granted to users to allow them to delete records from the system audit table (`AUD$`).

Additionally, the following system privilege can be granted to users who require access to tables created in the `SYS` schema:

- `SELECT ANY DICTIONARY`

This system privilege allows query access to any object in the `SYS` schema, including tables created in that schema. It must be granted individually to each user requiring the privilege. It is not included in `GRANT ALL PRIVILEGES`, nor can it be granted through a role.

Caution: You should grant these roles and the `SELECT ANY DICTIONARY` system privilege with extreme care, since the integrity of your system can be compromised by their misuse.

Object Privileges

Each type of object has different privileges associated with it.

You can specify `ALL [PRIVILEGES]` to grant or revoke all available object privileges for an object. `ALL` is not a privilege; rather, it is a shortcut, or a way of granting or revoking all object privileges with one word in `GRANT` and `REVOKE` statements. Note that if all object privileges are granted using the `ALL` shortcut, individual privileges can still be revoked.

Likewise, all individually granted privileges can be revoked by specifying `ALL`. However, if you `REVOKE ALL`, and revoking causes integrity constraints to be deleted (because they depend on a `REFERENCES` privilege that you are revoking), you must include the `CASCADE CONSTRAINTS` option in the `REVOKE` statement.

See Also: *Oracle9i SQL Reference*. for the complete list of object privileges

User Roles

A **role** groups several privileges and roles, so that they can be granted to and revoked from users simultaneously. A role must be enabled for a user before it can be used by the user.

Oracle provides some predefined roles to help in database administration. These roles, listed in [Table 25–1](#), are automatically defined for Oracle databases when you run the standard scripts that are part of database creation. You can grant privileges and roles to, and revoke privileges and roles from, these predefined roles in the same way as you do with any role you define.

Table 25–1 *Predefined Roles* (Page 1 of 2)

Role Name	Created By (Script)	Description
CONNECT	SQL.BSQ	Includes the following system privileges: ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
RESOURCE	SQL.BSQ	Includes the following system privileges: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
DBA	SQL.BSQ	All system privileges WITH ADMIN OPTION
<p>Note: The previous three roles are provided to maintain compatibility with previous versions of Oracle and may not be created automatically in future versions of Oracle. Oracle Corporation recommends that you design your own roles for database security, rather than relying on these roles.</p>		
EXP_FULL_DATABASE	CATEXP.SQL	Provides the privileges required to perform full and incremental database exports. Includes: SELECT ANY TABLE, BACKUP ANY TABLE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE, ADMINISTER RESOURCE MANAGER, and INSERT, DELETE, and UPDATE on the tables SYS.INCVID, SYS.INCFIL, and SYS.INCEXP. Also the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE.
IMP_FULL_DATABASE	CATEXP.SQL	Provides the privileges required to perform full database imports. Includes an extensive list of system privileges (use view DBA_SYS_PRIVS to view privileges) and the following roles: EXECUTE_CATALOG_ROLE and SELECT_CATALOG_ROLE.

Table 25–1 Predefined Roles (Page 2 of 2)

Role Name	Created By (Script)	Description
DELETE_CATALOG_ROLE	SQL.BSQ	Provides DELETE privilege on the system audit table (AUD\$).
EXECUTE_CATALOG_ROLE	SQL.BSQ	Provides EXECUTE privilege on objects in the data dictionary. Also, HS_ADMIN_ROLE.
SELECT_CATALOG_ROLE	SQL.BSQ	Provides SELECT privilege on objects in the data dictionary. Also, HS_ADMIN_ROLE.
RECOVERY_CATALOG_OWNER	CATALOG.SQL	Provides privileges for owner of the recovery catalog. Includes: CREATE SESSION, ALTER SESSION, CREATE SYNONYM, CREATE VIEW, CREATE DATABASE LINK, CREATE TABLE, CREATE CLUSTER, CREATE SEQUENCE, CREATE TRIGGER, and CREATE PROCEDURE
HS_ADMIN_ROLE	CATHS.SQL	Used to protect access to the HS (Heterogeneous Services) data dictionary tables (grants SELECT) and packages (grants EXECUTE). It is granted to SELECT_CATALOG_ROLE and EXECUTE_CATALOG_ROLE such that users with generic data dictionary access also can access the HS data dictionary.
AQ_USER_ROLE	CATQUEUE.SQL	Obsoleted, but kept mainly for release 8.0 compatibility. Provides execute privilege on DBMS_AQ and DBMS_AQIN.
AQ_ADMINISTRATOR_ROLE	CATQUEUE.SQL	Provides privileges to administer Advance Queuing. Includes ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, and MANAGE ANY QUEUE, SELECT privileges on AQ tables and EXECUTE privileges on AQ packages.
SNMPAGENT	CATSNMP.SQL	This role is used by Enterprise Manager/Intelligent Agent. Includes ANALYZE ANY and grants SELECT on various views.

If you install other options or products, other predefined roles may be created.

Managing User Roles

This section describes aspects of managing roles, and contains the following topics:

- [Creating a Role](#)
- [Specifying the Type of Role Authorization](#)

- [Dropping Roles](#)

Creating a Role

You can create a role using the `CREATE ROLE` statement, but you must have the `CREATE ROLE` system privilege to do so. Typically, only security administrators have this system privilege.

Note: Immediately after creation, a role has no privileges associated with it. To associate privileges with a new role, you must grant privileges or other roles to the new role.

You must give each role you create a unique name among existing usernames and role names of the database. Roles are not contained in the schema of any user. In a database that uses a multibyte character set, Oracle recommends that each role name contain at least one single-byte character. If a role name contains only multibyte characters, the encrypted role name/password combination is considerably less secure.

The following statement creates the `clerk` role, which is authorized by the database using the password `bicentennial`:

```
CREATE ROLE clerk IDENTIFIED BY bicentennial;
```

The `IDENTIFIED BY` clause specifies how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If this clause is not specified, or `NOT IDENTIFIED` is specified, then no authorization is required when the role is enabled. Roles can be specified to be authorized by:

- The database using a password
- An application using a specified package
- Externally by the operating system, network, or other external source
- Globally by an enterprise directory service

These authorizations are discussed in following sections.

Later, you can set or change the authorization method for a role using the `ALTER ROLE` statement. The following statement alters the `clerk` role to specify that the user must have been authorized by an external source before enabling the role:

```
ALTER ROLE clerk IDENTIFIED EXTERNALLY;
```

To alter the authorization method for a role, you must have the `ALTER ANY ROLE` system privilege or have been granted the role with the `ADMIN OPTION`.

See Also: *Oracle9i SQL Reference* for syntax, restrictions, and authorization information about the SQL statements used to manage roles and privileges

Specifying the Type of Role Authorization

The methods of authorizing roles are presented in this section. A role must be enabled for you to use it.

See Also: ["When Do Grants and Revokes Take Effect?"](#) on page 25-20 for a discussion about enabling roles

Role Authorization by the Database

The use of a role authorized by the database can be protected by an associated password. If you are granted a role protected by a password, you can enable or disable the role by supplying the proper password for the role in a `SET ROLE` statement. However, if the role is made a default role and enabled at connect time, the user is not required to enter a password.

The following statement creates a role `manager`. When it is enabled, the password `morework` must be supplied.

```
CREATE ROLE manager IDENTIFIED BY morework;
```

Note: In a database that uses a multibyte character set, passwords for roles must include only singlebyte characters. Multibyte characters are not accepted in passwords. See the *Oracle9i SQL Reference* for information about specifying valid passwords.

Role Authorization by an Application

The `IDENTIFIED USING package_name` clause lets you create an application role, which is a role that can be enabled only by applications using an authorized package. Application developers do not need to secure a role by embedding passwords inside applications. Instead, they can create an application role and specify which PL/SQL package is authorized to enable the role.

The following example indicates that the role `admin_role` is an application role and the role can only be enabled by any module defined inside the PL/SQL package `hr.admin`.

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

When enabling the user's default roles at login as specified in the user's profile, no checking is performed for application roles.

Role Authorization by an External Source

The following statement creates a role named `accts_rec` and requires that the user be authorized by an external source before it can be enabled:

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

Role Authorization by the Operating System Role authentication through the operating system is useful only when the operating system is able to dynamically link operating system privileges with applications. When a user starts an application, the operating system grants an operating system privilege to the user. The granted operating system privilege corresponds to the role associated with the application. At this point, the application can enable the application role. When the application is terminated, the previously granted operating system privilege is revoked from the user's operating system account.

If a role is authorized by the operating system, you must configure information for each user at the operating system level. This operation is operating system dependent.

If roles are granted by the operating system, you do not need to have the operating system authorize them also; this is redundant.

See Also: ["Granting Roles Using the Operating System or Network"](#) on page 25-22 for more information about roles granted by the operating system

Role Authorization and Network Clients If users connect to the database over Oracle Net, by default their roles cannot be authenticated by the operating system. This includes connections through a shared server configuration, as this connection requires Oracle Net. This restriction is the default because a remote user could impersonate another operating system user over a network connection.

If you are not concerned with this security risk and want to use operating system role authentication for network clients, set the initialization parameter `REMOTE_OS_`

ROLES in the database's initialization parameter file to TRUE. The change will take effect the next time you start the instance and mount the database. The parameter is FALSE by default.

Role Authorization by an Enterprise Directory Service

A role can be defined as a global role, whereby a (global) user can only be authorized to use the role by an enterprise directory service. You define the global role locally in the database by granting privileges and roles to it, but you cannot grant the global role itself to any user or other role in the database. When a global user attempts to connect to the database, the enterprise directory is queried to obtain any global roles associated with the user.

The following statement creates a global role:

```
CREATE ROLE supervisor IDENTIFIED GLOBALLY;
```

Global roles are one component of enterprise user management. A global role only applies to one database, but it can be granted to an enterprise role defined in the enterprise directory. An enterprise role is a directory structure which contains global roles on multiple databases, and which can be granted to enterprise users.

A general discussion of global authentication and authorization of users, and its role in enterprise user management, was presented earlier in "[Global Authentication and Authorization](#)" on page 24-13.

See Also: *Oracle Advanced Security Administrator's Guide* and *Oracle Internet Directory Administrator's Guide* for information about enterprise user management and how to implement it

Dropping Roles

In some cases, it may be appropriate to drop a role from the database. The security domains of all users and roles granted a dropped role are immediately changed to reflect the absence of the dropped role's privileges. All indirectly granted roles of the dropped role are also removed from affected security domains. Dropping a role automatically removes the role from all users' default role lists.

Because the creation of objects is not dependent on the privileges received through a role, tables and other objects are not dropped when a role is dropped.

You can drop a role using the SQL statement `DROP ROLE`. To drop a role, you must have the `DROP ANY ROLE` system privilege or have been granted the role with the `ADMIN OPTION`.

The following statement drops the role CLERK:

```
DROP ROLE clerk;
```

Granting User Privileges and Roles

This section describes the granting of privileges and roles, and contains the following topics:

- [Granting System Privileges and Roles](#)
- [Granting Object Privileges](#)
- [Granting Privileges on Columns](#)

It is also possible to grant roles to a user connected through a middle tier or proxy. This is discussed in "[Proxy Authentication and Authorization](#)" on page 24-16.

Granting System Privileges and Roles

You can grant system privileges and roles to other users and roles using the GRANT statement. The following privileges are required:

- To grant a system privilege, you must have been granted the system privilege with the ADMIN OPTION or have been granted the GRANT ANY PRIVILEGE system privilege.
- To grant a role, you must have been granted the role with the ADMIN OPTION or have been granted the GRANT ANY ROLE system privilege.

Note: You cannot grant a roll that is IDENTIFIED GLOBALLY to anything. The granting (and revoking) of global roles is controlled entirely by the enterprise directory service.

The following statement grants the system privilege CREATE SESSION and the accts_pay role to the user jward:

```
GRANT CREATE SESSION, accts_pay TO jward;
```

Note: Object privileges *cannot* be granted along with system privileges and roles in the same GRANT statement.

Granting the ADMIN OPTION

A user or role that is granted a privilege or role specifying the `WITH ADMIN OPTION` clause has several expanded capabilities:

- The grantee can grant or revoke the system privilege or role to or from *any* user or other role in the database. Users cannot revoke a role from themselves.
- The grantee can further grant the system privilege or role with the `ADMIN OPTION`.
- The grantee of a role can alter or drop the role.

In the following statement, the security administrator grants the `new_dba` role to `michael`:

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

The user `michael` cannot only use all of the privileges implicit in the `new_dba` role, but can grant, revoke, or drop the `new_dba` role as deemed necessary. Because of these powerful capabilities, exercise caution when granting system privileges or roles with the `ADMIN OPTION`. Such privileges are usually reserved for a security administrator and rarely granted to other administrators or users of the system.

When a user creates a role, the role is automatically granted to the creator with the `ADMIN OPTION`

Creating a New User with the GRANT Statement

Oracle allows you to create a new user with the `GRANT` statement. If you specify a password using the `IDENTIFIED BY` clause, and the username/password does not exist in the database, a new user with that username and password is created. The following example creates `ssmith` as a new user while granting `ssmith` the `CONNECT` system privilege:

```
GRANT CONNECT TO ssmith IDENTIFIED BY plq2r3;
```

See Also: ["Creating Users"](#) on page 24-2

Granting Object Privileges

You also use the `GRANT` statement to grant object privileges to roles and users. To grant an object privilege, you must fulfill one of the following conditions:

- You own the object specified.

- You possess the `GRANT ANY OBJECT PRIVILEGE` system privilege that enables you to grant and revoke privileges on behalf of the object owner.
- The `WITH GRANT OPTION` clause was specified when you were granted the object privilege by its owner.

Note: System privileges and roles cannot be granted along with object privileges in the same `GRANT` statement.

The following statement grants the `SELECT`, `INSERT`, and `DELETE` object privileges for all columns of the `emp` table to the users `jfee` and `tsmith`:

```
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
```

To grant all object privileges on the `salary` view to the user `jfee`, use the `ALL` keyword, as shown in the following example:

```
GRANT ALL ON salary TO jfee;
```

Specifying the `GRANT OPTION`

Specify `WITH GRANT OPTION` to enable the grantee to grant the object privileges to other users and roles. The user whose schema contains an object is automatically granted all associated object privileges with the `GRANT OPTION`. This special privilege allows the grantee several expanded privileges:

- The grantee can grant the object privilege to any users in the database, with or without the `GRANT OPTION`, or to any role in the database.
- If both of the following are true, the grantee can create views on the table and grant the corresponding privileges on the views to any user or role in the database.
 - The grantee receives object privileges for the table with the `GRANT OPTION`.
 - The grantee has the `CREATE VIEW` or `CREATE ANY VIEW` system privilege.

The `GRANT OPTION` is not valid when granting an object privilege to a role. Oracle prevents the propagation of object privileges through roles so that grantees of a role cannot propagate object privileges received by means of roles.

Granting Object Privileges on Behalf of the Object Owner

The `GRANT ANY OBJECT PRIVILEGE` system privilege allows users to grant and revoke any object privilege on behalf of the object owner. This provides a convenient means for database and application administrators to grant access to objects in any schema without requiring that they connect to the schema. This eliminates the need to maintain login credentials for schema owners so that they can grant access to objects, and it reduces the number of connections required during configuration.

This system privilege is part of the Oracle supplied `DBA` role and is thus granted (with the `ADMIN OPTION`) to any user connecting `AS SYSDBA` (user `SYS`). As with other system privileges, the `GRANT ANY OBJECT PRIVILEGE` system privilege can only be granted by a user who possesses the `ADMIN OPTION`.

When you exercise the `GRANT ANY OBJECT PRIVILEGE` system privilege to grant an object privilege to a user, if you already possess the object privilege with the `GRANT OPTION`, then the grant is performed in the usual way. In this case, you become the grantor of the grant. If you do not possess the object privilege, then the object owner is shown as the grantor, even though you, with the `GRANT ANY OBJECTPRIVILEGE` system privilege, actually performed the grant.

Note: The audit record generated by the `GRANT` statement will always show the real user who performed the grant.

For example, consider the following. User `adams` possesses the `GRANT ANY OBJECT PRIVILEGE` system privilege. He does not possess any other grant privileges. He issues the following statement:

```
GRANT SELECT ON hr.employees TO blake WITH GRANT OPTION;
```

If you examine the `DBA_TAB_PRIVS` view, you will see that `hr` is shown as being the grantor of the privilege:

```
SQL> SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE
2>     FROM DBA_TAB_PRIVS
3>     WHERE TABLE_NAME = 'EMPLOYEES' and OWNER = 'HR';
```

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
BLAKE	HR	HR	SELECT	YES

Now assume that `blake` also has the `GRANT ANY OBJECT PRIVILEGE` system. He, issues the following statement:

```
GRANT SELECT ON hr.employees TO clark;
```

In this case, when you again query the `DBA_TAB_PRIVS` view, you see that `blake` is shown as being the grantor of the privilege:

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
BLAKE	HR	HR	SELECT	YES
CLARK	HR	BLAKE	SELECT	NO

This is because `blake` already possesses the `SELECT` privilege on `hr.employees` with the `GRANT OPTION`.

See Also: ["Revoking Object Privileges on Behalf of the Object Owner"](#) on page 25-17

Granting Privileges on Columns

You can grant `INSERT`, `UPDATE`, or `REFERENCES` privileges on individual columns in a table.

Caution: Before granting a column-specific `INSERT` privilege, determine if the table contains any columns on which `NOT NULL` constraints are defined. Granting selective insert capability without including the `NOT NULL` columns prevents the user from inserting any rows into the table. To avoid this situation, make sure that each `NOT NULL` column is either insertable or has a non-`NULL` default value. Otherwise, the grantee will not be able to insert rows into the table and will receive an error.

The following statement grants `INSERT` privilege on the `acct_no` column of the `accounts` table to `scott`:

```
GRANT INSERT (acct_no) ON accounts TO scott;
```

In another example, object privilege for the `ename` and `job` columns of the `emp` table are granted to the users `jfee` and `tsmith`:

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

Revoking User Privileges and Roles

This section describes aspects of revoking user privileges and roles, and contains the following topics:

- [Revoking System Privileges and Roles](#)
- [Revoking Object Privileges](#)
- [Cascading Effects of Revoking Privileges](#)

Revoking System Privileges and Roles

You can revoke system privileges and roles using the SQL statement `REVOKE`.

Any user with the `ADMIN OPTION` for a system privilege or role can revoke the privilege or role from any other database user or role. The revoker does not have to be the user that originally granted the privilege or role. Users with `GRANT ANY ROLE` can revoke *any* role.

The following statement revokes the `CREATE TABLE` system privilege and the `accts_rec` role from `tsmith`:

```
REVOKE CREATE TABLE, accts_rec FROM tsmith;
```

Note: The `ADMIN OPTION` for a system privilege or role cannot be selectively revoked. The privilege or role must be revoked and then the privilege or role re-granted without the `ADMIN OPTION`.

Revoking Object Privileges

The `REVOKE` statement is used to revoke object privileges. To revoke an object privilege, you must fulfill one of the following conditions:

- You previously granted the object privilege to the user or role.
- You possess the `GRANT ANY OBJECT PRIVILEGE` system privilege that enables you to grant and revoke privileges on behalf of the object owner.

You can only revoke the privileges that you, the grantor, directly authorized, not the grants made by other users to whom you granted the `GRANT OPTION`. However, there is a cascading effect. The object privilege grants propagated using the `GRANT OPTION` are revoked if a grantor's object privilege is revoked.

Assuming you are the original grantor, the following statement revokes the `SELECT` and `INSERT` privileges on the `emp` table from the users `jfee` and `tsmith`:

```
REVOKE SELECT, insert ON emp FROM jfee, tsmith;
```

The following statement revokes all object privileges for the dept table that you originally granted to the human_resource role

```
REVOKE ALL ON dept FROM human_resources;
```

Note: The GRANT OPTION for an object privilege cannot be selectively revoked. The object privilege must be revoked and then re-granted without the GRANT OPTION. Users cannot revoke object privileges from themselves.

Revoking Object Privileges on Behalf of the Object Owner

The GRANT ANY OBJECT PRIVILEGE system privilege allows you to revoke any specified object privilege where the object owner is the grantor. This occurs when the object privilege is granted by the object owner, or on behalf of the owner by any user holding the GRANT ANY OBJECT PRIVILEGE system privilege.

In a situation where the object privilege has been granted by both the owner of the object and the user executing the REVOKE statement (who has both the specific object privilege and the GRANT ANY OBJECT PRIVILEGE system privilege), Oracle only revokes the object privilege granted by the user issuing the REVOKE. This can be illustrated by continuing the example started in ["Granting Object Privileges on Behalf of the Object Owner"](#) on page 25-14.

At this point, blake has granted the SELECT privilege on hr.employees to clark. Even though blake possesses the GRANT ANY OBJECT PRIVILEGE system privilege, he also holds the specific object privilege, thus this grant is attributed to him. Assume that hr also grants the SELECT privilege on hr.employees to clark. A query of the DBA_TAB_PRIVS view shows that the following grants are in effect for the hr.employees table:

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
BLAKE	HR	HR	SELECT	YES
CLARK	HR	BLAKE	SELECT	NO
CLARK	HR	HR	SELECT	NO

User blake now issues the following REVOKE statement:

```
REVOKE SELECT ON hr.employees FROM clark;
```

Only the object privilege for `clark` granted by `blake` is removed. The grant by the object owner, `hr`, remains.

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
BLAKE	HR	HR	SELECT	YES
CLARK	HR	HR	SELECT	NO

If `blake` issues the `REVOKE` statement again, this time the effect will be to remove the object privilege granted by `hr`.

See Also: ["Granting Object Privileges on Behalf of the Object Owner"](#) on page 25-14

Revoking Column-Selective Object Privileges

Although users can grant column-selective `INSERT`, `UPDATE`, and `REFERENCES` privileges for tables and views, they cannot selectively revoke column specific privileges with a similar `REVOKE` statement. Instead, the grantor must first revoke the object privilege for all columns of a table or view, and then selectively re-grant the column-specific privileges that should remain.

For example, assume that role `human_resources` has been granted the `UPDATE` privilege on the `deptno` and `dname` columns of the table `dept`. To revoke the `UPDATE` privilege on just the `deptno` column, issue the following two statements:

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

The `REVOKE` statement revokes `UPDATE` privilege on all columns of the `dept` table from the role `human_resources`. The `GRANT` statement re-grants `UPDATE` privilege on the `dname` column to the role `human_resources`.

Revoking the REFERENCES Object Privilege

If the grantee of the `REFERENCES` object privilege has used the privilege to create a foreign key constraint (that currently exists), the grantor can revoke the privilege only by specifying the `CASCADE CONSTRAINTS` option in the `REVOKE` statement:

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

Any foreign key constraints currently defined that use the revoked `REFERENCES` privilege are dropped when the `CASCADE CONSTRAINTS` clause is specified.

Cascading Effects of Revoking Privileges

Depending on the type of privilege, there may be cascading effects when a privilege is revoked.

System Privileges

There are no cascading effects when revoking a system privilege related to DDL operations, regardless of whether the privilege was granted with or without the `ADMIN OPTION`. For example, assume the following:

1. The security administrator grants the `CREATE TABLE` system privilege to `jfee` with the `ADMIN OPTION`.
2. User `jfee` creates a table.
3. User `jfee` grants the `CREATE TABLE` system privilege to `tsmith`.
4. User `tsmith` creates a table.
5. The security administrator revokes the `CREATE TABLE` system privilege from `jfee`.
6. User `jfee`'s table continues to exist. `tsmith` still has the table and the `CREATE TABLE` system privilege.

Cascading effects can be observed when revoking a system privilege related to a DML operation. If `SELECT ANY TABLE` is revoked from a user, then all procedures contained in the users schema relying on this privilege will fail until the privilege is reauthorized.

Object Privileges

Revoking an object privilege can have cascading effects that should be investigated before issuing a `REVOKE` statement.

- Object definitions that depend on a DML object privilege can be affected if the DML object privilege is revoked. For example, assume the procedure body of the `test` procedure includes a SQL statement that queries data from the `emp` table. If the `SELECT` privilege on the `emp` table is revoked from the owner of the `test` procedure, the procedure can no longer be executed successfully.
- When a `REFERENCES` privilege for a table is revoked from a user, any foreign key integrity constraints defined by the user that require the dropped `REFERENCES` privilege are automatically dropped. For example, assume that the user `jward` is granted the `REFERENCES` privilege for the `deptno` column of the `dept` table and creates a foreign key on the `deptno` column in the `emp` table

that references the deptno column. If the references privilege on the deptno column of the dept table is revoked, the foreign key constraint on the deptno column of the emp table is dropped in the same operation.

- The object privilege grants propagated using the GRANT OPTION are revoked if a grantor's object privilege is revoked. For example, assume that user1 is granted the SELECT object privilege with the GRANT OPTION, and grants the SELECT privilege on emp to user2. Subsequently, the SELECT privilege is revoked from user1. This REVOKE is cascaded to user2 as well. Any objects that depended on user1's and user2's revoked SELECT privilege can also be affected, as described in previous bullet items.

Object definitions that require the ALTER and INDEX DDL object privileges are not affected if the ALTER or INDEX object privilege is revoked. For example, if the INDEX privilege is revoked from a user that created an index on someone else's table, the index continues to exist after the privilege is revoked.

Granting to and Revoking from the User Group PUBLIC

Privileges and roles can also be granted to and revoked from the user group PUBLIC. Because PUBLIC is accessible to every database user, all privileges and roles granted to PUBLIC are accessible to every database user.

Security administrators and database users should grant a privilege or role to PUBLIC only if every database user requires the privilege or role. This recommendation reinforces the general rule that at any given time, each database user should only have the privileges required to accomplish the group's current tasks successfully.

Revoking a privilege from PUBLIC can cause significant cascading effects. If any privilege related to a DML operation is revoked from PUBLIC (for example, SELECT ANY TABLE, UPDATE ON emp), all procedures in the database, including functions and packages, must be *reauthorized* before they can be used again. Therefore, exercise caution when granting and revoking DML-related privileges to PUBLIC.

See Also: ["Managing Object Dependencies"](#) on page 21-23 for more information about object dependencies

When Do Grants and Revokes Take Effect?

Depending on what is granted or revoked, a grant or revoke takes effect at different times:

- All grants/revokes of system and object privileges to anything (users, roles, and PUBLIC) are immediately observed.
- All grants/revokes of roles to anything (users, other roles, PUBLIC) are only observed when a current user session issues a `SET ROLE` statement to re-enable the role after the grant/revoke, or when a new user session is created after the grant/revoke.

You can see which roles are currently enabled by examining the `SESSION_ROLES` data dictionary view.

The SET ROLE Statement

During the session, the user or an application can use the `SET ROLE` statement any number of times to change the roles currently enabled for the session. You must already have been granted the roles that you name in the `SET ROLE` statement. The number of roles that can be concurrently enabled is limited by the initialization parameter `MAX_ENABLED_ROLES`.

This example enables the role `clerk`, which you have already been granted, and specifies the password.

```
SET ROLE clerk IDENTIFIED BY bicentennial;
```

You can disable all roles with the following statement:

```
SET ROLE NONE;
```

Specifying Default Roles

When a user logs on, Oracle enables all privileges granted explicitly to the user and all privileges in the user's default roles.

A user's list of default roles can be set and altered using the `ALTER USER` statement. The `ALTER USER` statement allows you to specify roles that are to be enabled when a user connects to the database, without requiring the user to specify the roles' passwords. The user must have already been directly granted the roles with a `GRANT` statement. You cannot specify as a default role any role managed by an external service including a directory service (external roles or global roles).

The following example establishes default roles for user `jane`:

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

You cannot set a user's default roles in the `CREATE USER` statement. When you first create a user, the user's default role setting is `ALL`, which causes all roles

subsequently granted to the user to be default roles. Use the `ALTER USER` statement to limit the user's default roles.

Caution: When you create a role (other than a user role), it is granted to you implicitly and added as a default role. You receive an error at login if you have more than `MAX_ENABLED_ROLES`. You can avoid this error by altering the user's default roles to be less than `MAX_ENABLED_ROLES`. Thus, you should change the `DEFAULT ROLE` settings of `SYS` and `SYSTEM` before creating user roles.

Restricting the Number of Roles that a User Can Enable

A user can enable as many roles as specified by the initialization parameter `MAX_ENABLED_ROLES`. All indirectly granted roles enabled as a result of enabling a primary role are included in this count. The database administrator can alter this limitation by modifying the value for this parameter. Higher values permit each user session to have more concurrently enabled roles. However, the larger the value for this parameter, the more memory space is required on behalf of each user session; this is because the PGA size is affected for each user session, and requires four bytes for each role. Determine the highest number of roles that will be concurrently enabled by any one user and use this value for the `MAX_ENABLED_ROLES` parameter.

Granting Roles Using the Operating System or Network

This section describes aspects of granting roles through your operating system or network, and contains the following topics:

- [Using Operating System Role Identification](#)
- [Using Operating System Role Management](#)
- [Granting and Revoking Roles When `OS_ROLES=TRUE`](#)
- [Enabling and Disabling Roles When `OS_ROLES=TRUE`](#)
- [Using Network Connections with Operating System Role Management](#)

Instead of a security administrator explicitly granting and revoking database roles to and from users using `GRANT` and `REVOKE` statements, the operating system that operates Oracle can grant roles to users at connect time. Roles can be administered using the operating system and passed to Oracle when a user creates a session. As part of this mechanism, each user's default roles and the roles granted to a user

with the `ADMIN OPTION` can be identified. Even if the operating system is used to authorize users for roles, all roles must be created in the database and privileges assigned to the role with `GRANT` statements.

Roles can also be granted through a network service.

The advantage of using the operating system to identify a user's database roles is that privilege management for an Oracle database can be externalized. The security facilities offered by the operating system control a user's privileges. This option may offer advantages of centralizing security for a number of system activities, such as the following situation:

- MVS Oracle administrators want RACF groups to identify a database user's roles.
- UNIX Oracle administrators want UNIX groups to identify a database user's roles.
- VMS Oracle administrators want to use rights identifiers to identify a database user's roles.

The main disadvantage of using the operating system to identify a user's database roles is that privilege management can only be performed at the role level. Individual privileges cannot be granted using the operating system, but can still be granted inside the database using `GRANT` statements.

A secondary disadvantage of using this feature is that by default users cannot connect to the database through the shared server, or any other network connection, if the operating system is managing roles. However, you can change this default; see ["Using Network Connections with Operating System Role Management"](#) on page 25-25.

Note: The features described in this section are available only on some operating systems. See your operating system specific Oracle documentation to determine if you can use these features.

Using Operating System Role Identification

To operate a database so that it uses the operating system to identify each user's database roles when a session is created, set the initialization parameter `OS_ROLES` to `TRUE` (and restart the instance, if it is currently running). When a user attempts to create a session with the database, Oracle initializes the user's security domain using the database roles identified by the operating system.

To identify database roles for a user, each Oracle user's operating system account must have operating system identifiers (these may be called groups, rights identifiers, or other similar names) that indicate which database roles are to be available for the user. Role specification can also indicate which roles are the default roles of a user and which roles are available with the `ADMIN OPTION`. No matter which operating system is used, the role specification at the operating system level follows the format:

```
ora_ID_ROLE[_[d][a]]
```

where:

- `ID` has a definition that varies on different operating systems. For example, on VMS, `ID` is the instance identifier of the database; on MVS, it is the machine type; on UNIX, it is the system `ID`.

Note: `ID` is case sensitive to match your `ORACLE_SID`. `ROLE` is not case sensitive.

- `ROLE` is the name of the database role.
- `d` is an optional character that indicates this role is to be a default role of the database user.
- `a` is an optional character that indicates this role is to be granted to the user with the `ADMIN OPTION`. This allows the user to grant the role to other roles only. Roles cannot be granted to users if the operating system is used to manage roles.

Note: If either the `d` or `a` characters are specified, they must be preceded by an underscore.

For example, an operating system account might have the following roles identified in its profile:

```
ora_PAYROLL_ROLE1  
ora_PAYROLL_ROLE2_a  
ora_PAYROLL_ROLE3_d  
ora_PAYROLL_ROLE4_da
```

When the corresponding user connects to the payroll instance of Oracle, `role3` and `role4` are defaults, while `role2` and `role4` are available with the `ADMIN OPTION`.

Using Operating System Role Management

When you use operating system managed roles, it is important to note that database roles are being granted to an operating system user. Any database user to which the OS user is able to connect will have the authorized database roles enabled. For this reason, you should consider defining all Oracle users as `IDENTIFIED EXTERNALLY` if you are using `OS_ROLES = TRUE`, so that the database accounts are tied to the OS account that was granted privileges.

Granting and Revoking Roles When `OS_ROLES=TRUE`

If `OS_ROLES` is set to `TRUE`, the operating system completely manages the grants and revokes of roles *to users*. Any previous grants of roles to users using `GRANT` statements do not apply; however, they are still listed in the data dictionary. Only the role grants made at the operating system level to users apply. Users can still grant privileges to roles and users.

Note: If the operating system grants a role to a user with the `ADMIN OPTION`, the user can grant the role only to other roles.

Enabling and Disabling Roles When `OS_ROLES=TRUE`

If `OS_ROLES` is set to `TRUE`, any role granted by the operating system can be dynamically enabled using the `SET ROLE` statement. This still applies, even if the role was defined to require a password or operating system authorization. However, any role not identified in a user's operating system account cannot be specified in a `SET ROLE` statement, even if a role has been granted using a `GRANT` statement when `OS_ROLES = FALSE`. (If you specify such a role, Oracle ignores it.)

When `OS_ROLES = TRUE`, a user can enable as many roles as specified by the initialization parameter `MAX_ENABLED_ROLES`.

Using Network Connections with Operating System Role Management

If you choose to have the operating system to manage roles, by default users cannot connect to the database through the shared server. This restriction is the default

because a remote user could impersonate another operating system user over a non-secure connection.

If you are not concerned with this security risk and want to use operating system role management with the shared server, or any other network connection, set the initialization parameter `REMOTE_OS_ROLES` in the database's initialization parameter file to `TRUE`. The change will take effect the next time you start the instance and mount the database. The default setting of this parameter is `FALSE`.

Viewing Privilege and Role Information

To access information about grants of privileges and roles, you can query the following data dictionary views:

View	Description
DBA_COL_PRIVS ALL_COL_PRIVS USER_COL_PRIVS	DBA view describes all column object grants in the database. ALL view describes all column object grants for which the current user or PUBLIC is the object owner, grantor, or grantee. USER view describes column object grants for which the current user is the object owner, grantor, or grantee.
ALL_COL_PRIVS_MADE USER_COL_PRIVS_MADE	ALL view lists column object grants for which the current user is object owner or grantor. USER view describes column object grants for which the current user is the grantor.
ALL_COL_PRIVS_RECD USER_COL_PRIVS_RECD	ALL view describes column object grants for which the current user or PUBLIC is the grantee. USER view describes column object grants for which the current user is the grantee.
DBA_TAB_PRIVS ALL_TAB_PRIVS USER_TAB_PRIVS	DBA view lists all grants on all objects in the database. ALL view lists the grants on objects where the user or PUBLIC is the grantee. USER view lists grants on all objects where the current user is the grantee.
ALL_TAB_PRIVS_MADE USER_TAB_PRIVS_MADE	ALL view lists the all object grants made by the current user or made on the objects owned by the current user. USER view lists grants on all objects owned by the current user.
ALL_TAB_PRIVS_RECD USER_TAB_PRIVS_RECD	ALL view lists object grants for which the user or PUBLIC is the grantee. USER view lists object grants for which the current user is the grantee.
DBA_ROLES	This view lists all roles that exist in the database.
DBA_ROLE_PRIVS USER_ROLE_PRIVS	DBA view lists roles granted to users and roles. USER view lists roles granted to the current user.

View	Description
DBA_SYS_PRIVS USER_SYS_PRIVS	DBA view lists system privileges granted to users and roles. USER view lists system privileges granted to the current user.
ROLE_ROLE_PRIVS	This view describes roles granted to other roles. Information is provided only about roles to which the user has access.
ROLE_SYS_PRIVS	This view contains information about system privileges granted to roles. Information is provided only about roles to which the user has access.
ROLE_TAB_PRIVS	This view contains information about object privileges granted to roles. Information is provided only about roles to which the user has access.
SESSION_PRIVS	This view lists the privileges that are currently enabled for the user.
SESSION_ROLES	This view lists the roles that are currently enabled to the user.

Some examples of using these views follow. For these examples, assume the following statements have been issued:

```
CREATE ROLE security_admin IDENTIFIED BY honcho;

GRANT CREATE PROFILE, ALTER PROFILE, DROP PROFILE,
      CREATE ROLE, DROP ANY ROLE, GRANT ANY ROLE, AUDIT ANY,
      AUDIT SYSTEM, CREATE USER, BECOME USER, ALTER USER, DROP USER
      TO security_admin WITH ADMIN OPTION;

GRANT SELECT, DELETE ON SYS.AUD$ TO security_admin;

GRANT security_admin, CREATE SESSION TO swilliams;

GRANT security_admin TO system_administrator;

GRANT CREATE SESSION TO jward;

GRANT SELECT, DELETE ON emp TO jward;

GRANT INSERT (ename, job) ON emp TO swilliams, jward;
```

See Also: *Oracle9i Database Reference* for a detailed description of these data dictionary views

Listing All System Privilege Grants

The following query returns all system privilege grants made to roles and users:

```
SELECT * FROM DBA_SYS_PRIVS;
```

GRANTEE	PRIVILEGE	ADM
-----	-----	---
SECURITY_ADMIN	ALTER PROFILE	YES
SECURITY_ADMIN	ALTER USER	YES
SECURITY_ADMIN	AUDIT ANY	YES
SECURITY_ADMIN	AUDIT SYSTEM	YES
SECURITY_ADMIN	BECOME USER	YES
SECURITY_ADMIN	CREATE PROFILE	YES
SECURITY_ADMIN	CREATE ROLE	YES
SECURITY_ADMIN	CREATE USER	YES
SECURITY_ADMIN	DROP ANY ROLE	YES
SECURITY_ADMIN	DROP PROFILE	YES
SECURITY_ADMIN	DROP USER	YES
SECURITY_ADMIN	GRANT ANY ROLE	YES
SWILLIAMS	CREATE SESSION	NO
JWARD	CREATE SESSION	NO

Listing All Role Grants

The following query returns all the roles granted to users and other roles:

```
SELECT * FROM DBA_ROLE_PRIVS;
```

GRANTEE	GRANTED_ROLE	ADM
-----	-----	---
SWILLIAMS	SECURITY_ADMIN	NO

Listing Object Privileges Granted to a User

The following query returns all object privileges (not including column-specific privileges) granted to the specified user:

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
WHERE GRANTEE = 'JWARD';
```

TABLE_NAME	PRIVILEGE	GRANTABLE
-----	-----	-----
EMP	SELECT	NO
EMP	DELETE	NO

To list all the column-specific privileges that have been granted, use the following query:

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
       FROM DBA_COL_PRIVS;
```

GRANTEE	TABLE_NAME	COLUMN_NAME	PRIVILEGE
SWILLIAMS	EMP	ENAME	INSERT
SWILLIAMS	EMP	JOB	INSERT
JWARD	EMP	NAME	INSERT
JWARD	EMP	JOB	INSERT

Listing the Current Privilege Domain of Your Session

The following query lists all roles currently enabled for the issuer:

```
SELECT * FROM SESSION_ROLES;
```

If `swilliams` has enabled the `security_admin` role and issues this query, Oracle returns the following information:

```
ROLE
-----
SECURITY_ADMIN
```

The following query lists all system privileges currently available in the issuer's security domain, both from explicit privilege grants and from enabled roles:

```
SELECT * FROM SESSION_PRIVS;
```

If `swilliams` has the `security_admin` role enabled and issues this query, Oracle returns the following results:

```
PRIVILEGE
-----
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

If the `security_admin` role is disabled for `swilliams`, the first query would have returned no rows, while the second query would only return a row for the `CREATE SESSION` privilege grant.

Listing Roles of the Database

The `DBA_ROLES` data dictionary view can be used to list all roles of a database and the authentication used for each role. For example, the following query lists all the roles in the database:

```
SELECT * FROM DBA_ROLES;
```

ROLE	PASSWORD
CONNECT	NO
RESOURCE	NO
DBA	NO
SECURITY_ADMIN	YES

Listing Information About the Privilege Domains of Roles

The `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, and `ROLE_TAB_PRIVS` data dictionary views contain information on the privilege domains of roles.

For example, the following query lists all the roles granted to the `system_admin` role:

```
SELECT GRANTED_ROLE, ADMIN_OPTION
       FROM ROLE_ROLE_PRIVS
       WHERE ROLE = 'SYSTEM_ADMIN';
```

GRANTED_ROLE	ADM
SECURITY_ADMIN	NO

The following query lists all the system privileges granted to the `security_admin` role:

```
SELECT * FROM ROLE_SYS_PRIVS WHERE ROLE = 'SECURITY_ADMIN';
```

ROLE	PRIVILEGE	ADM
SECURITY_ADMIN	ALTER PROFILE	YES
SECURITY_ADMIN	ALTER USER	YES

SECURITY_ADMIN	AUDIT ANY	YES
SECURITY_ADMIN	AUDIT SYSTEM	YES
SECURITY_ADMIN	BECOME USER	YES
SECURITY_ADMIN	CREATE PROFILE	YES
SECURITY_ADMIN	CREATE ROLE	YES
SECURITY_ADMIN	CREATE USER	YES
SECURITY_ADMIN	DROP ANY ROLE	YES
SECURITY_ADMIN	DROP PROFILE	YES
SECURITY_ADMIN	DROP USER	YES
SECURITY_ADMIN	GRANT ANY ROLE	YES

The following query lists all the object privileges granted to the `security_admin` role:

```
SELECT TABLE_NAME, PRIVILEGE FROM ROLE_TAB_PRIVS
       WHERE ROLE = 'SECURITY_ADMIN';
```

TABLE_NAME	PRIVILEGE
-----	-----
AUD\$	DELETE
AUD\$	SELECT

Auditing Database Use

This chapter describes how to use the Oracle database server's auditing facilities, and contains these topics:

- [Guidelines for Auditing](#)
- [What Information is Contained in the Audit Trail?](#)
- [Actions Audited by Default](#)
- [Auditing Administrative Users](#)
- [Managing the Audit Trail](#)
- [Fine-Grained Auditing](#)
- [Viewing Database Audit Trail Information](#)

Guidelines for Auditing

This section describes guidelines for auditing and contains the following topics:

- [Decide Whether to Use the Database or Operating System Audit Trail](#)
- [Keep Audited Information Manageable](#)
- [Guidelines for Auditing Suspicious Database Activity](#)
- [Guidelines for Auditing Normal Database Activity](#)

Decide Whether to Use the Database or Operating System Audit Trail

The data dictionary of every database has a table named `SYS.AUD$`, commonly referred to as the database **audit trail**, that is designed to store entries auditing database statements, privileges, or schema objects.

You can optionally choose to store the database audit information to an operating system file. If your operating system has an audit trail that stores audit records generated by the operating system auditing facility, and Oracle is allowed to write to it, you can choose to direct the database audit entries to this file. For example, the Windows operating system allows Oracle to write audit records as events to the application event log.

Consider the advantages and disadvantages of using either the database or operating system audit trail to store database audit records.

Using the database audit trail offers the following advantages:

- You can view selected portions of the audit trail with the predefined audit trail views of the data dictionary.
- You can use Oracle tools (such as Oracle Reports) to generate audit reports.

Alternatively, your operating system audit trail may allow you to consolidate audit records from multiple sources including Oracle and other applications. Therefore, examining system activity might be more efficient because all audit records are in one place.

See Also: Your operating system specific documentation for information about its auditing capabilities

Keep Audited Information Manageable

Although auditing is relatively inexpensive, limit the number of audited events as much as possible. This minimizes the performance impact on the execution of statements that are audited, and minimizes the size of the audit trail.

Use the following general guidelines when devising an auditing strategy:

- Evaluate your purpose for auditing.

After you have a clear understanding of the reasons for auditing, you can devise an appropriate auditing strategy and avoid unnecessary auditing.

For example, suppose you are auditing to investigate suspicious database activity. This information by itself is not specific enough. What types of suspicious database activity do you suspect or have you noticed? A more focused auditing purpose might be to audit unauthorized deletions from arbitrary tables in the database. This purpose narrows the type of action being audited and the type of object being affected by the suspicious activity.

- Audit knowledgeably.

Audit the minimum number of statements, users, or objects required to get the targeted information. This prevents unnecessary audit information from cluttering the meaningful information and consuming valuable space in the `SYSTEM` tablespace. Balance your need to gather sufficient security information with your ability to store and process it.

For example, if you are auditing to gather information about database activity, determine exactly what types of activities you are tracking, audit only the activities of interest, and audit only for the amount of time necessary to gather the information you desire. Do not audit objects if you are only interested in each session's logical I/O information.

Guidelines for Auditing Suspicious Database Activity

When you audit to monitor suspicious database activity, use the following guidelines:

- Audit generally, then specifically.

When starting to audit for suspicious database activity, it is common that not much information is available to target specific users or schema objects. Therefore, audit options must be set more generally at first. Once preliminary audit information is recorded and analyzed, the general audit options should be turned off and more specific audit options enabled. This process should

continue until enough evidence is gathered to make concrete conclusions about the origin of the suspicious database activity.

- Protect the audit trail.

When auditing for suspicious database activity, protect the audit trail so that audit information cannot be added, changed, or deleted without being audited.

See Also: ["Protecting the Audit Trail"](#) on page 26-18

Guidelines for Auditing Normal Database Activity

When your purpose for auditing is to gather historical information about particular database activities, use the following guidelines:

- Audit only pertinent actions.

To avoid cluttering meaningful information with useless audit records and reduce the amount of audit trail administration, only audit the targeted database activities.

- Archive audit records and purge the audit trail.

After you have collected the required information, archive the audit records of interest and purge the audit trail of this information.

What Information is Contained in the Audit Trail?

Oracle can write records to either the database audit trail, an operating system file, or both. This section describes the makeup of this audit trail information.

Information Stored in the Database Audit Trail

The database audit trail, stored in the `SYS.AUD$` table, contains different types of information, depending on the events audited and the auditing options set. The following information is always included in each audit trail record:

- Operating system login user name
- User name
- Session identifier
- Terminal identifier
- Name of the schema object accessed

- Operation performed or attempted
- Completion code of the operation
- Date and time stamp

The audit trail does not store information about any data values that might be involved in the audited statement. For example, old and new data values of updated rows are not stored when an `UPDATE` statement is audited. However, this specialized type of auditing can be performed using fine-grained auditing methods.

See Also: ["Fine-Grained Auditing"](#) on page 26-18 for more information about methods of fine-grained auditing

Information Stored in an Operating System File

The operating system file that contains the audit trail can contain any of the following:

- Audit records generated by the operating system
- Database audit trail records
- Database actions that are always audited
- Audit records for administrative users (`SYS`)

Audit trail records written to an operating system audit trail may contain encoded information, but this information can be decoded using data dictionary tables and error messages as follows:

Encoded Information	How to Decode
Action code	This describes the operation performed or attempted. The <code>AUDIT_ACTIONS</code> data dictionary table contains a list of these codes and their descriptions.
Privileges used	This describes any system privileges used to perform the operation. The <code>SYSTEM_PRIVILEGE_MAP</code> table lists all of these codes and their descriptions.
Completion code	This describes the result of the attempted operation. Successful operations return a value of zero; unsuccessful operations return the Oracle error code describing why the operation was unsuccessful. These codes are listed in <i>Oracle9i Database Error Messages</i> .

Actions Audited by Default

Regardless of whether database auditing is enabled, Oracle *always* audits certain database-related operations and writes them to the operating system audit file. These operations include the following:

- Connections to the instance with administrator privileges

An audit record is generated that lists the operating system user connecting to Oracle as `SYSOPER` or `SYSDBA`. This provides for accountability of users with administrative privileges. Full auditing for these users can be enabled as explained in "[Auditing Administrative Users](#)" on page 26-6.

- Database startup

An audit record is generated that lists the operating system user starting the instance, the user's terminal identifier, the date and time stamp, and whether database auditing was enabled or disabled. This is stored in the operating system audit trail because the database audit trail is not available until after startup has successfully completed. Recording the state of database auditing at startup helps detect when an administrator has restarted a database with database auditing disabled (thus enabling the administrator to perform unaudited actions).

- Database shutdown

An audit record is generated that lists the operating system user shutting down the instance, the user's terminal identifier, and the date and time stamp.

Auditing Administrative Users

Sessions for users who connect as `SYS`, this includes all users connecting as `SYSDBA` or `SYSOPER`, can be fully audited. Use the `AUDIT_SYS_OPERATIONS` initialization parameter to specify if user `SYS` is audited. For example, the following setting specifies that `SYS` is to be audited:

```
AUDIT_SYS_OPERATIONS = TRUE
```

A value of `FALSE`, which is the default, disables `SYS` auditing.

All audit records for `SYS` are written to the operating system file that contains the audit trail, and not to `SYS.AUD$`. All `SYS` issued SQL statements are audited indiscriminately and regardless of the setting of the `AUDIT_TRAIL` initialization parameter.

Consider the following `SYS` session:

```
CONNECT / AS SYSDBA;  
ALTER SYSTEM FLUSH SHARED_POOL;  
UPDATE salary SET base=1000 WHERE name='myname';
```

When SYS auditing is enabled, both the ALTER SYSTEM and UPDATE statements are displayed in the operating system audit file as follows:

```
Thu Jan 24 12:58:00 2002  
ACTION: 'CONNECT'  
DATABASE USER: '/'  
OSPRIV: SYSDBA  
CLIENT USER: scott  
CLIENT TERMINAL: pts/2  
STATUS: 0
```

```
Thu Jan 24 12:58:00 2002  
ACTION: 'alter system flush shared_pool'  
DATABASE USER: ''  
OSPRIV: SYSDBA  
CLIENT USER: scott  
CLIENT TERMINAL: pts/2  
STATUS: 0
```

```
Thu Jan 24 12:58:00 2002  
ACTION: 'update salary set base=1000 where name='myname''  
DATABASE USER: ''  
OSPRIV: SYSDBA  
CLIENT USER: scott  
CLIENT TERMINAL: pts/2  
STATUS: 0
```

Because of the superuser privileges available to users who connect as SYSDBA, Oracle recommends that DBAs rarely use this connection and only when necessary. Normal day to day maintenance activity can usually be done by DBAs assigned the DBA role.

Managing the Audit Trail

This section describes various aspects of managing audit trail information, and contains the following topics:

- [Enabling and Disabling Auditing](#)
- [Setting Auditing Options](#)

- [Auditing in a Multi-Tier Environment](#)
- [Turning Off Audit Options](#)
- [Controlling the Growth and Size of the Audit Trail](#)
- [Protecting the Audit Trail](#)

Enabling and Disabling Auditing

Any authorized database user can set statement, privilege, and object auditing options at any time, but Oracle does not generate audit information for the database audit trail unless database auditing is enabled. The security administrator is normally responsible for controlling auditing.

This section discusses the initialization parameters that enable and disable auditing.

Note: All of the initialization parameters affecting auditing are static. This means that if you change the values for the `AUDIT_SYS_OPERATIONS`, `AUDIT_TRAIL`, and `AUDIT_FILE_DEST` initialization parameters, you must shut down and restart your database for the new values to take effect.

Setting the `AUDIT_TRAIL` Initialization Parameter

Database auditing is enabled and disabled by the `AUDIT_TRAIL` initialization parameter in the database's initialization parameter file. The parameter can be set to the following values:

Parameter Value	Meaning
DB	Enables database auditing and directs all audit records to the database audit trail, except for records that are always written to the operating system audit trail
OS	Enables database auditing and directs all audit records to an operating system file
NONE	Disables auditing (This value is the default.)

Setting the `AUDIT_FILE_DEST` Initialization Parameter

The `AUDIT_FILE_DEST` initialization parameter specifies an operating system directory into which the audit trail is written when `AUDIT_TRAIL=OS` is specified. It is also the location to which mandatory auditing information is written and, if so

specified by the `AUDIT_SYS_OPERATIONS` initialization parameter, audit records for user `SYS`.

If the `AUDIT_FILE_DEST` parameter is not specified, the default location is `$ORACLE_HOME/rdbms/audit`.

Notes:

- If your operating system supports an audit trail, then its location is operating system specific. For example, the Windows operating systems writes audit records as events to the application event log. You can view and manage these events using Event Viewer. You are not allowed to specify the `AUDIT_FILE_DEST` initialization parameter for Windows platforms. For more information, see *Oracle9i Database Administrator's Guide for Windows*.
 - For some operating systems, an audit record for instance connection and database startup is always logged to the default location `$ORACLE_HOME/rdbms/audit` regardless of the setting for `AUDIT_FILE_DEST`. This is because until the database is mounted, the parameter setting is not known.
-
-

Setting Auditing Options

You specify auditing options using the `AUDIT` statement. The `AUDIT` statement allows you to set audit options at three levels:

Level	Effect
Statement	Causes auditing of specific SQL statements or groups of statements that affect a particular type of database object. For example, <code>AUDIT TABLE</code> audits the <code>CREATE TABLE</code> , <code>TRUNCATE TABLE</code> , <code>COMMENT ON TABLE</code> , and <code>DELETE [FROM] TABLE</code> statements.
Privilege	Audits SQL statements that are authorized by the specified system privilege. For Example, <code>AUDIT CREATE ANY TRIGGER</code> audits statements issued using the <code>CREATE ANY TRIGGER</code> system privilege.
Object	Audits specific statements on specific objects, such as <code>ALTER TABLE</code> on the <code>emp</code> table

To use the `AUDIT` statement to set statement and privilege options, you must have the `AUDIT SYSTEM` privilege. To use it to set object audit options, you must own the object to be audited or have the `AUDIT ANY` privilege.

Audit statements that set statement and privilege audit options can include a `BY` clause to specify a list of users or application proxies to limit the scope of the statement and privilege audit options.

When setting auditing options, you can also specify the following conditions for auditing:

- `BY SESSION/BY ACCESS`

`BY SESSION` causes Oracle to write a single record for all SQL statements of the same type issued in the same session. `BY ACCESS` causes Oracle to write one record for each access.

Note: If you are using an operating system file for the audit trail (`AUDIT_FILE_DEST=OS`), multiple records may still be written to the audit trail when `BY SESSION` is specified. This is because while Oracle can write to the operating system file, it is unable to read it to detect that it has already written an audit entry for the action.

- `WHENEVER SUCCESSFUL/WHENEVER NOT SUCCESSFUL`

`WHENEVER SUCCESSFUL` chooses auditing only for statements that succeed. `WHENEVER NOT SUCCESSFUL` chooses auditing only for statements that fail or result in errors.

The implications of your choice of auditing option and specification of `AUDIT` statement clauses is discussed in subsequent sections.

A new database session picks up auditing options from the data dictionary when the session is created. These auditing options remain in force for the duration of the database connection. Setting new system or object auditing options causes all subsequent database sessions to use these options; existing sessions continue using the audit options in place at session creation.

Caution: The `AUDIT` statement only specifies auditing options; it does not enable auditing as a whole. To turn auditing on and control whether Oracle generates audit records based on the audit options currently set, set the initialization parameter `AUDIT_TRAIL` as described in "Enabling and Disabling Auditing" on page 26-8.

See Also: *Oracle9i SQL Reference* for a complete description of the `AUDIT` statement

Specifying Statement Auditing

Valid statement audit options that can be included in `AUDIT` and `NOAUDIT` statements are listed in the *Oracle9i SQL Reference*.

Two special cases of statement auditing are discussed in the following sections.

Auditing Connections and Disconnections The `SESSION` statement option is unique because it does not generate an audit record when a particular type of statement is issued; this option generates a single audit record for each session created by connections to an instance. An audit record is inserted into the audit trail at connect time and updated at disconnect time. Cumulative information about a session such as connection time, disconnection time, logical and physical I/Os processed, and more is stored in a single audit record that corresponds to the session.

To audit all successful and unsuccessful connections to and disconnections from the database, regardless of user, `BY SESSION` (the default and only value for this option), enter the following statement:

```
AUDIT SESSION;
```

You can set this option selectively for individual users also, as in the next example:

```
AUDIT SESSION  
BY scott, lori;
```

Auditing Statements That Fail Because an Object Does Not Exist The `NOT EXISTS` statement option specifies auditing of all SQL statements that fail because the target object does not exist.

Specifying Privilege Auditing

Privilege audit options exactly match the corresponding system privileges. For example, the option to audit use of the `DELETE ANY TABLE` privilege is `DELETE`

ANY TABLE. To turn this option on, you use a statement similar to the following example:

```
AUDIT DELETE ANY TABLE
  BY ACCESS
  WHENEVER NOT SUCCESSFUL;
```

Oracle's system privileges are listed in the *Oracle9i SQL Reference*.

To audit all successful and unsuccessful uses of the DELETE ANY TABLE system privilege, enter the following statement:

```
AUDIT DELETE ANY TABLE;
```

To audit all unsuccessful SELECT, INSERT, and DELETE statements on all tables and unsuccessful uses of the EXECUTE PROCEDURE system privilege, by all database users, and by individual audited statement, issue the following statement:

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE, EXECUTE PROCEDURE
  BY ACCESS
  WHENEVER NOT SUCCESSFUL;
```

The AUDIT SYSTEM system privilege is required to set any statement or privilege audit option. Normally, the security administrator is the only user granted this system privilege.

Specifying Object Auditing

The *Oracle9i SQL Reference* lists valid object audit options and the schema object types for which each option is available.

A user can set any object audit option for the objects contained in the user's schema. The AUDIT ANY system privilege is required to set an object audit option for an object contained in another user's schema or to set the default object auditing option. Normally, the security administrator is the only user granted the AUDIT ANY privilege.

To audit all successful and unsuccessful DELETE statements on the scott.emp table, BY SESSION (the default value), enter the following statement:

```
AUDIT DELETE ON scott.emp;
```

To audit all successful SELECT, INSERT, and DELETE statements on the dept table owned by user jward, BY ACCESS, enter the following statement:

```
AUDIT SELECT, INSERT, DELETE
  ON jward.dept
```

```
BY ACCESS
WHENEVER SUCCESSFUL;
```

To set the default object auditing options to audit all unsuccessful `SELECT` statements, `BY SESSION` (the default), enter the following statement:

```
AUDIT SELECT
ON DEFAULT
WHENEVER NOT SUCCESSFUL;
```

Auditing in a Multi-Tier Environment

In a multi-tier environment, Oracle preserves the identity of the client through all tiers. This enables auditing of actions taken on behalf of the client. To do so, you use the `BY proxy` clause in your `AUDIT` statement.

This clause allows you a few options. You can:

- Audit SQL statements issued by the specified proxy on its own behalf
- Audit statements executed on behalf of a specified user or users
- Audit all statements executed on behalf of any user

The following example audits `SELECT TABLE` statements issued on behalf of client `jackson` by the proxy application server `appserve`.

```
AUDIT SELECT TABLE
BY appserve ON BEHALF OF jackson;
```

See Also: *Oracle9i Database Concepts* and *Oracle9i Application Developer's Guide - Fundamentals* for more information on proxies and multi-tier applications

Turning Off Audit Options

The `NOAUDIT` statement turns off the various audit options of Oracle. Use it to reset statement and privilege audit options, and object audit options. A `NOAUDIT` statement that sets statement and privilege audit options can include the `BY user` or `BY proxy` option to specify a list of users to limit the scope of the statement and privilege audit options.

You can use a `NOAUDIT` statement to disable an audit option selectively using the `WHENEVER` clause. If the clause is not specified, the auditing option is disabled entirely, for both successful and unsuccessful cases.

The BY SESSION/BY ACCESS option pair is *not* supported by the NOAUDIT statement; audit options, no matter how they were turned on, are turned off by an appropriate NOAUDIT statement.

Caution: The NOAUDIT statement only specifies auditing options; it does not disable auditing as a whole. To turn auditing off and stop Oracle from generating audit records, set the initialization parameter AUDIT_TRAIL in the database's initialization parameter file as described in "[Enabling and Disabling Auditing](#)" on page 26-8.

See Also: *Oracle9i SQL Reference* for a complete syntax listing of the NOAUDIT statement

Turning Off Statement and Privilege Auditing

The following statements turn off the corresponding audit options:

```
NOAUDIT session;
NOAUDIT session BY scott, lori;
NOAUDIT DELETE ANY TABLE;
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
EXECUTE PROCEDURE;
```

The following statement turns off all statement audit options:

```
NOAUDIT ALL;
```

The following statement turns off all privilege audit options:

```
NOAUDIT ALL PRIVILEGES;
```

To disable statement or privilege auditing options, you must have the AUDIT SYSTEM system privilege.

Turning Off Object Auditing

The following statements turn off the corresponding auditing options:

```
NOAUDIT DELETE
ON emp;
NOAUDIT SELECT, INSERT, DELETE
ON jward.dept;
```

Furthermore, to turn off all object audit options on the `emp` table, enter the following statement:

```
NOAUDIT ALL  
ON emp;
```

To turn off all default object audit options, enter the following statement:

```
NOAUDIT ALL  
ON DEFAULT;
```

All schema objects created before this `NOAUDIT` statement is issued continue to use the default object audit options in effect at the time of their creation, unless overridden by an explicit `NOAUDIT` statement after their creation.

To disable object audit options for a specific object, you must be the owner of the schema object. To disable the object audit options of an object in another user's schema or to disable default object audit options, you must have the `AUDIT ANY` system privilege. A user with privileges to disable object audit options of an object can override the options set by any user.

Controlling the Growth and Size of the Audit Trail

If the audit trail becomes completely full and no more audit records can be inserted, audited statements cannot be successfully executed until the audit trail is purged. Warnings are returned to all users that issue audited statements. Therefore, the security administrator must control the growth and size of the audit trail.

When auditing is enabled and audit records are being generated, the audit trail grows according to two factors:

- The number of audit options turned on
- The frequency of execution of audited statements

To control the growth of the audit trail, you can use the following methods:

- Enable and disable database auditing. If it is enabled, audit records are generated and stored in the audit trail; if it is disabled, audit records are not generated.
- Be very selective about the audit options that are turned on. If more selective auditing is performed, useless or unnecessary audit information is not generated and stored in the audit trail.
- Tightly control the ability to perform object auditing. This can be done two different ways:

- A security administrator owns all objects and the `AUDIT ANY` system privilege is never granted to any other user. Alternatively, all schema objects can belong to a schema for which the corresponding user does not have `CREATE SESSION` privilege.
- All objects are contained in schemas that do not correspond to real database users (that is, the `CREATE SESSION` privilege is not granted to the corresponding user) and the security administrator is the only user granted the `AUDIT ANY` system privilege.

In both scenarios, object auditing is controlled entirely by the security administrator.

The maximum size of the database audit trail (`SYS.AUD$` table) is determined by the default storage parameters of the `SYSTEM` tablespace, in which it is stored. You should not move `SYS.AUD$` to another tablespace as a means of controlling the growth and size of the audit trail. However, you can modify the storage parameters for `SYS.AUD$`.

Note: Moving the `SYS.AUD$` table out of the `SYSTEM` tablespace is not supported because the Oracle code makes implicit assumptions about the data dictionary tables, such as `SYS.AUD$`, which could cause problems with upgrades and backup/recovery scenarios.

See Also: Your operating system specific Oracle documentation for more information about managing the operating system audit trail when you are directing audit records to that location

Purging Audit Records from the Audit Trail

After auditing is enabled for some time, the security administrator may want to delete records from the database audit trail both to free audit trail space and to facilitate audit trail management.

For example, to delete *all* audit records from the audit trail, enter the following statement:

```
DELETE FROM SYS.AUD$;
```

Alternatively, to delete all audit records from the audit trail generated as a result of auditing the table `emp`, enter the following statement:

```
DELETE FROM SYS.AUD$
```

```
WHERE obj$name='EMP';
```

If audit trail information must be archived for historical purposes, the security administrator can copy the relevant records to a normal database table (for example, using `INSERT INTO table SELECT ... FROM SYS.AUD$...`) or export the audit trail table to an operating system file.

Only the user `SYS`, a user who has the `DELETE ANY TABLE` privilege, or a user to whom `SYS` has granted `DELETE` privilege on `SYS.AUD$` can delete records from the database audit trail.

Note: If the audit trail is completely full and connections are being audited (that is, if the `SESSION` option is set), typical users cannot connect to the database because the associated audit record for the connection cannot be inserted into the audit trail. In this case, the security administrator must connect as `SYS` (operations by `SYS` are not audited) and make space available in the audit trail.

See Also: *Oracle9i Database Utilities* for information about exporting tables

Reducing the Size of the Audit Trail

As with any database table, after records are deleted from the database audit trail, the extents allocated for this table still exist.

If the database audit trail has many extents allocated for it, but many of them are not being used, the space allocated to the database audit trail can be reduced by following these steps:

1. If you want to save information currently in the audit trail, copy it to another database table or export it using the `EXPORT` utility.
2. Connect as a user with administrator privileges.
3. Truncate `SYS.AUD$` using the `TRUNCATE` statement.
4. Reload archived audit trail records generated from Step 1.

The new version of `SYS.AUD$` is allocated only as many extents as are necessary to contain current audit trail records.

Note: `SYS.AUD$` is the only `SYS` object that should ever be directly modified.

Protecting the Audit Trail

When auditing for suspicious database activity, protect the integrity of the audit trail's records to guarantee the accuracy and completeness of the auditing information.

To protect the database audit trail from unauthorized deletions, grant the `DELETE ANY TABLE` system privilege to security administrators only.

To audit changes made to the database audit trail, use the following statement:

```
AUDIT INSERT, UPDATE, DELETE
  ON sys.aud$
  BY ACCESS;
```

Audit records generated as a result of object audit options set for the `SYS.AUD$` table can only be deleted from the audit trail by someone connected with administrator privileges, which itself has protection against unauthorized use.

Fine-Grained Auditing

In the auditing methods discussed so far, a fixed set of facts is recorded in the audit trail. Additionally, audit options can only be set to monitor access of objects or privileges. No support has been discussed for obtaining more specific information about the environment or query results, nor any mechanism to specify audit conditions in order to minimize false audits. For these purposes, Oracle offers fine-grained auditing.

Fine-grained auditing allows the monitoring of data access based on content. For example, a central tax authority needs to track access to tax returns to guard against employee snooping. Enough detail is wanted to be able to determine what data was accessed, not just that `SELECT` privilege was used by a specific user on a particular table. Fine-grained auditing provides this functionality.

In general, fine-grained auditing policy is based on simple user-defined SQL predicates on table objects as conditions for selective auditing. During fetching, whenever policy conditions are met for a returning row, the query is audited. Later, Oracle executes user-defined audit event handlers using autonomous transactions to process the event.

Fine-grained auditing can be implemented in user applications using the `DBMS_FGA` package or by using database triggers.

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for information about using fine-grained auditing

Viewing Database Audit Trail Information

The database audit trail (`SYS.AUD$`) is a single table in each Oracle database's data dictionary. To help you meaningfully view auditing information in this table, several predefined views are available. They must be created by you. You can later delete them if you decide not to use auditing.

Creating the Audit Trail Views

The following views (except `STMT_AUDIT_OPTION_MAP`) are created by the `CATALOG.SQL` and `CATAUDIT.SQL` scripts:

View	Description
<code>STMT_AUDIT_OPTION_MAP</code>	Contains information about auditing option type codes. Created by the <code>SQL.BSQ</code> script at <code>CREATE DATABASE</code> time.
<code>AUDIT_ACTIONS</code>	Contains descriptions for audit trail action type codes
<code>ALL_DEF_AUDIT_OPTS</code>	Contains default object-auditing options that will be applied when objects are created
<code>DBA_STMT_AUDIT_OPTS</code>	Describes current system auditing options across the system and by user
<code>DBA_PRIV_AUDIT_OPTS</code>	Describes current system privileges being audited across the system and by user
<code>DBA_OBJ_AUDIT_OPTS</code> <code>USER_OBJ_AUDIT_OPTS</code>	Describes auditing options on all objects. <code>USER</code> view describes auditing options on all objects owned by the current user.
<code>DBA_AUDIT_TRAIL</code> <code>USER_AUDIT_TRAIL</code>	Lists all audit trail entries. <code>USER</code> view shows audit trail entries relating to current user.
<code>DBA_AUDIT_OBJECT</code> <code>USER_AUDIT_OBJECT</code>	Contains audit trail records for all objects in the system. <code>USER</code> view lists audit trail records for statements concerning objects that are accessible to the current user.

View	Description
DBA_AUDIT_SESSION USER_AUDIT_SESSION	Lists all audit trail records concerning CONNECT and DISCONNECT. USER view lists all audit trail records concerning connections and disconnections for the current user.
DBA_AUDIT_STATEMENT USER_AUDIT_STATEMENT	Lists audit trail records concerning GRANT, REVOKE, AUDIT, NOAUDIT, and ALTER SYSTEM statements throughout the database, or for the USER view, issued by the user
DBA_AUDIT_EXISTS	Lists audit trail entries produced BY AUDIT NOT EXISTS
The following views are used for fine-grained auditing:	
DBA_AUDIT_POLICIES	Shows all the auditing policies on the system.
DBA_FGA_AUDIT_TRAIL	Lists audit trail records for value-based auditing.

See Also: *Oracle9i Database Reference* for more detailed descriptions of the Oracle provided predefined views

Deleting the Audit Trail Views

If you disable auditing and no longer need the audit trail views, delete them by connecting to the database as SYS and running the script file CATNOAUD.SQL. The name and location of the CATNOAUD.SQL script are operating system dependent.

Using Audit Trail Views to Investigate Suspicious Activities

This section offers examples that demonstrate how to examine and interpret the information in the audit trail. Consider the following situation.

You would like to audit the database for the following suspicious activities:

- Passwords, tablespace settings, and quotas for some database users are being altered without authorization.
- A high number of deadlocks are occurring, most likely because of users acquiring exclusive table locks.
- Rows are arbitrarily being deleted from the emp table in scott's schema.

You suspect the users jward and swilliams of several of these detrimental actions.

To enable your investigation, you issue the following statements (in order):

```
AUDIT ALTER, INDEX, RENAME ON DEFAULT
  BY SESSION;
CREATE VIEW scott.employee AS SELECT * FROM scott.emp;
AUDIT SESSION BY jward, swilliams;
AUDIT ALTER USER;
AUDIT LOCK TABLE
  BY ACCESS
  WHENEVER SUCCESSFUL;
AUDIT DELETE ON scott.emp
  BY ACCESS
  WHENEVER SUCCESSFUL;
```

The following statements are subsequently issued by the user jward:

```
ALTER USER tsmith QUOTA 0 ON users;
DROP USER djones;
```

The following statements are subsequently issued by the user swilliams:

```
LOCK TABLE scott.emp IN EXCLUSIVE MODE;
DELETE FROM scott.emp WHERE mgr = 7698;
ALTER TABLE scott.emp ALLOCATE EXTENT (SIZE 100K);
CREATE INDEX scott.ename_index ON scott.emp (ename);
CREATE PROCEDURE scott.fire_employee (empid NUMBER) AS
  BEGIN
    DELETE FROM scott.emp WHERE empno = empid;
  END;
/

EXECUTE scott.fire_employee(7902);
```

The following sections display the information relevant to your investigation that can be viewed using the audit trail views in the data dictionary:

- [Listing Active Statement Audit Options](#)
- [Listing Active Privilege Audit Options](#)
- [Listing Active Object Audit Options for Specific Objects](#)
- [Listing Default Object Audit Options](#)
- [Listing Audit Records](#)
- [Listing Audit Records for the AUDIT SESSION Option](#)

Listing Active Statement Audit Options

The following query returns all the statement audit options that are set:

```
SELECT * FROM DBA_STMT_AUDIT_OPTS;
```

USER_NAME	AUDIT_OPTION	SUCCESS	FAILURE
JWARD	SESSION	BY SESSION	BY SESSION
SWILLIAMS	SESSION	BY SESSION	BY SESSION
	LOCK TABLE	BY ACCESS	NOT SET

Notice that the view reveals the statement audit options set, whether they are set for success or failure (or both), and whether they are set for BY SESSION or BY ACCESS.

Listing Active Privilege Audit Options

The following query returns all the privilege audit options that are set:

```
SELECT * FROM DBA_PRIV_AUDIT_OPTS;
```

USER_NAME	PRIVILEGE	SUCCESS	FAILURE
ALTER USER	BY SESSION	BY SESSION	

Listing Active Object Audit Options for Specific Objects

The following query returns all audit options set for any objects whose name starts with the characters emp and which are contained in scott's schema:

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS
WHERE OWNER = 'SCOTT' AND OBJECT_NAME LIKE 'EMP%';
```

OWNER	OBJECT_NAME	OBJECT_TY	ALT	AUD	COM	DEL	GRA	IND	INS	LOC	...
SCOTT	EMP	TABLE	S/S	-/-	-/-	A/-	-/-	S/S	-/-	-/-	...
SCOTT	EMPLOYEE	VIEW	-/-	-/-	-/-	A/-	-/-	S/S	-/-	-/-	...

Notice that the view returns information about all the audit options for the specified object. The information in the view is interpreted as follows:

- The character "-" indicates that the audit option is not set.
- The character "S" indicates that the audit option is set, BY SESSION.
- The character "A" indicates that the audit option is set, BY ACCESS.

- Each audit option has two possible settings, `WHENEVER SUCCESSFUL` and `WHENEVER NOT SUCCESSFUL`, separated by `"/"`. For example, the `DELETE` audit option for `scott.emp` is set `BY ACCESS` for successful delete statements and not set at all for unsuccessful delete statements.

Listing Default Object Audit Options

The following query returns all default object audit options:

```
SELECT * FROM ALL_DEF_AUDIT_OPTS;

ALT AUD COM DEL GRA IND INS LOC REN SEL UPD REF EXE
--- --- --- --- --- --- --- --- --- --- --- ---
S/S -/- -/- -/- -/- S/S -/- -/- S/S -/- -/- -/- -/-
```

Notice that the view returns information similar to the `USER_OBJ_AUDIT_OPTS` and `DBA_OBJ_AUDIT_OPTS` views (see previous example).

Listing Audit Records

The following query lists audit records generated by statement and object audit options:

```
SELECT * FROM DBA_AUDIT_OBJECT;
```

Listing Audit Records for the AUDIT SESSION Option

The following query lists audit information corresponding to the `AUDIT SESSION` statement audit option:

```
SELECT USERNAME, LOGOFF_TIME, LOGOFF_LREAD, LOGOFF_PREAD,
       LOGOFF_LWRITE, LOGOFF_DLOCK
       FROM DBA_AUDIT_SESSION;
```

USERNAME	LOGOFF_TI	LOGOFF_LRE	LOGOFF_PRE	LOGOFF_LWR	LOGOFF_DLO
JWARD	02-AUG-91	53	2	24	0
SWILLIAMS	02-AUG-91	3337	256	630	0

Part V

Database Resource Management

Part V discusses database resource management. It includes the following chapter:

- [Chapter 27, "Using the Database Resource Manager"](#)

Using the Database Resource Manager

Oracle provides database resource management capability through its Database Resource Manager. This chapter introduces you to its use.

The following topics are discussed:

- [What Is the Database Resource Manager?](#)
- [Administering the Database Resource Manager](#)
- [Creating a Simple Resource Plan](#)
- [Creating Complex Resource Plans](#)
- [Managing Resource Consumer Groups](#)
- [Enabling the Database Resource Manager](#)
- [Putting It All Together: Database Resource Manager Examples](#)
- [Monitoring and Tuning the Database Resource Manager](#)
- [Viewing Database Resource Manager Information](#)

Note: This chapter discusses the use of the Oracle supplied `DBMS_RESOURCE_MANAGER` and `DBMS_RESOURCE_MANAGER_PRIVS` packages to administer the Database Resource Manager. You can more easily administer the Database Resource Manager through the Oracle Enterprise Manager (OEM). It provides an easy to use graphical interface for administering the Database Resource Manager.

See the Oracle Enterprise Manager documentation set for more information.

What Is the Database Resource Manager?

The main goal of the Database Resource Manager is to give the Oracle database server more control over resource management decisions, thus circumventing problems resulting from inefficient operating system management.

This section contains the following topics:

- [What Problems Does the Database Resource Manager Address?](#)
- [How Does the Database Resource Manager Address These Problems?](#)
- [What are the Elements of the Database Resource Manager?](#)
- [Understanding Resource Plans](#)

What Problems Does the Database Resource Manager Address?

When database resource allocation decisions are left to the operating system, you may encounter the following problems:

- Excessive overhead
Excessive overhead results from operating system context switching between Oracle server processes when the number of server processes is high.
- Inefficient scheduling
The operating system deschedules Oracle database servers while they hold latches, which is inefficient.
- Inappropriate allocation of resources
The operating system distributes resources equally among all active processes and is unable to prioritize one task over another.
- Inability to manage database-specific resources, such as parallel execution servers and active sessions

How Does the Database Resource Manager Address These Problems?

Oracle's Database Resource Manager helps to overcome these problems by allowing the database more control over how machine resources are allocated.

Specifically, using the Database Resource Manager, you can:

- Guarantee certain users a minimum amount of processing resources regardless of the load on the system and the number of users

- Distribute available processing resources by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage may be given to ROLAP (relational on-line analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users
- Create an **active session pool**. This pool consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will abort.
- Allow **automatic switching** of users from one group to another group based on administrator defined criteria. If a member of a particular group of users creates a session that executes for longer than a specified amount of time, that session can be automatically switched to another group of users with different resource requirements.
- Prevent the execution of operations that are estimated to run for a longer time than a predefined limit
- Create an **undo pool**. This pool consists of the amount of undo space that can be consumed in by a group of users.
- Configure an instance to use a particular method of allocating resources. You can dynamically change the method, for example, from a daytime setup to a nighttime setup, without having to shut down and restart the instance.

What are the Elements of the Database Resource Manager?

The elements of Oracle's database resource management, which you define through the Database Resource Manager packages, are described below.

Element	Description
Resource consumer group	User sessions grouped together based on resource processing requirements.
Resource plan	Contains directives that specify how resources are allocated to resource consumer groups.

Element	Description
Resource allocation method	The method/policy used by the Database Resource Manager when allocating for a particular resource; used by resource consumer groups and resource plans. Oracle provides the resource allocation methods that are available, but you determine which method to use.
Resource plan directive	Used by administrators to associate resource consumer groups with particular plans and allocate resources among resource consumer groups.

You will learn how to create and use these elements in later sections of this chapter.

Understanding Resource Plans

This section briefly introduces the concept of resource plans. Included are some illustrations of simple resource plans. More complex plans are included in the examples presented later ("[Putting It All Together: Database Resource Manager Examples](#)" on page 27-25), after it has been explained how to build and maintain the elements of the Database Resource Manager.

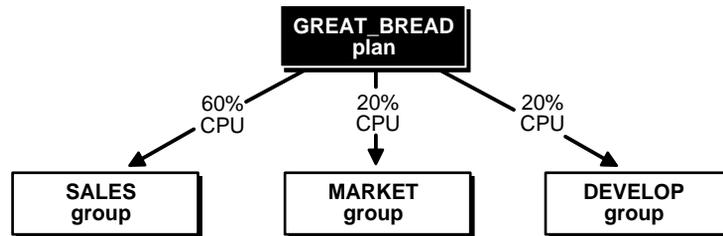
Resource plans specify the resource consumer groups belonging to the plan and contain directives for how resources are to be allocated among these groups. You use the `DBMS_RESOURCE_MANAGER` package to create and maintain these elements of the Database Resource Manager: resource plans, resource consumer groups, and resource plan directives. Plan information is stored in tables in the data dictionary. Several views are available for viewing plan data.

See Also: *Oracle9i Database Concepts* for more information about the Database Resource Manager

["Viewing Database Resource Manager Information"](#) on page 27-31

A Single-Level Resource Plan

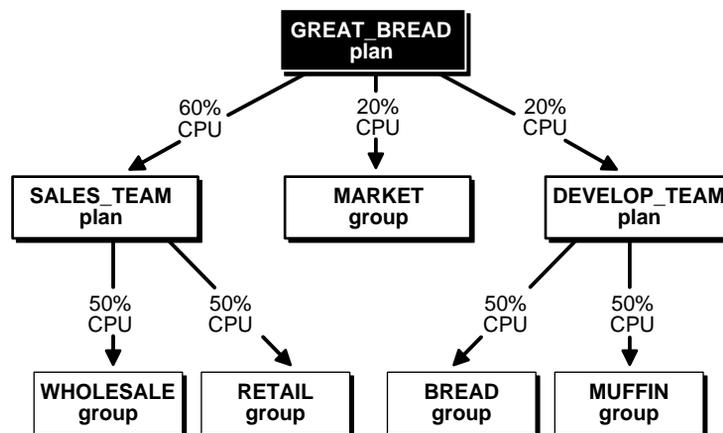
The first illustration, shown in [Figure 27-1](#), is of a single-level plan, where the plan allocates resources among resource consumer groups. The Great Bread Company has a plan called `great_bread` that allocates CPU resources among three resource consumer groups. Specifically, `sales` is allotted 60% of the CPU time, `market` is allotted 20%, and `develop` receives the remaining 20%.

Figure 27-1 A Simple Resource Management Plan

Oracle provides a procedure (`CREATE_SIMPLE_PLAN`) that enables you to quickly create a simple resource plan. This procedure is discussed in ["Creating a Simple Resource Plan"](#) on page 27-10.

A Multilevel Resource Plan

But a plan cannot only contain resource consumer groups, it can also contain other plans, called **subplans**. Maybe the Great Bread Company chooses to divide their CPU resource as shown in [Figure 27-2](#).

Figure 27-2 A Multilevel Plan With Subplans

In this case, the `great_bread` plan still allocates CPU resources to the consumer group `market`, but now it allocates CPU resources to subplans `sales_team` and `develop_team`, who in turn allocate resources to consumer groups. [Figure 27-2](#)

illustrates a **plan schema**, which contains a **top plan** (`great_bread`) and all of its descendents.

It is possible for a subplan or consumer group to have more than one parent (owning plan), but there cannot be any loops in a plan schema. An example of a subplan having more than one parent would be if the Great Bread Company had a night plan and a day plan. Both the night plan and the day plan contain the `sales` subplan as a member, but perhaps with a different CPU resource allocation in each instance.

Note: As explained later, the above plans should also contain a plan directive for `OTHER_GROUPS`. To present a simplified view, however, this plan directive is not shown.

Resource Consumer Groups

Resource consumer groups are groups of users, or sessions, that are grouped together based on their processing needs. Resource plan directives, discussed next, specify how resources are allocated among consumer groups and subplans in a plan schema.

Resource Plan Directives

How resources are allocated to resource consumer groups is specified in resource allocation directives. The Database Resource Manager provides several means of allocating resources.

CPU Method This method enables you to specify how CPU resources are to be allocated among consumer groups or subplans. The multiple levels of CPU resource allocation (up to eight levels) provide a means of prioritizing CPU usage within a plan schema. Level 2 gets resources only after level 1 is unable to use all of its resources. Multiple levels not only provide a way of prioritizing, but they provide a way of explicitly specifying how all primary and leftover resources are to be used.

Active Session Pool with Queuing You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum designates the active session pool. When a session cannot be initiated because the pool is full, the session is placed into a queue. When an active session completes, the first session in the queue can then be scheduled for execution. You can also specify a timeout period after which a job in the execution queue (waiting for execution) will timeout, causing it to terminate with an error.

An entire parallel execution session is counted as one active session.

Degree of Parallelism Limit Specifying a parallel degree limit enables you to control the maximum degree of parallelism for any operation within a consumer group.

Automatic Consumer Group Switching This method enables you to control resources by specifying criteria that, if met, causes the automatic switching of sessions to another consumer group. The criteria used to determine switching are:

- **Switch group**—specifies the consumer group to which this session is switched if the other (following) criteria are met
- **Switch time**—specifies the length of time that a session can execute before it is switched to another consumer group
- **Use estimate**—specifies whether Oracle is to use its own estimate of how long an operation will execute

The Database Resource Manager switches a running session to *switch group* if the session is active for more than *switch time* seconds. Active means that the session is running and consuming resources, not waiting idly for user input or waiting for CPU cycles. The session is allowed to continue running, even if the active session pool for the new group is full. Under these conditions a consumer group can have more sessions running than specified by its active session pool. Once the session finishes its operation and becomes idle, it is switched back to its original group.

If *use estimate* is set to `TRUE`, the Database Resource Manager uses a predicted estimate of how long the operation will take to complete. If Oracle's predicted estimate is longer than the value specified as the switch time, then Oracle switches the session before execution starts. If this parameter is not set, the operation starts normally and only switches groups when other switch criteria are met.

Execution Time Limit You can specify a maximum execution time allowed for an operation. If Oracle estimates that an operation will run longer than the specified maximum execution time, the operation is terminated with an error. This error can be trapped and the operation rescheduled.

Undo Pool You can specify an undo pool for each consumer group. An undo pool controls the amount of total undo that can be generated by a consumer group. When the total undo generated by a consumer group exceeds its undo limit, the current DML statement generating the redo is terminated. No other members of the consumer group can perform further data manipulation until undo space is freed from the pool.

See Also: *Oracle9i Database Concepts* for additional conceptual information about the Database Resource Manager.

Administering the Database Resource Manager

You must have the system privilege `ADMINISTER_RESOURCE_MANAGER` to administer the Database Resource Manager. Typically, database administrators have this privilege with the `ADMIN` option as part of the `DBA` (or equivalent) role.

Being an administrator for the Database Resource Manager allows you to execute all of the procedures in the `DBMS_RESOURCE_MANAGER` package. These are listed in the following table, and their use is explained in succeeding sections of this chapter.

Procedure	Description
<code>CREATE_SIMPLE_PLAN</code>	Creates a simple resource plan, containing up to eight consumer groups, in one step. This is the quickest way to get started when you use this package.
<code>CREATE_PLAN</code>	Creates a resource plan and specifies its allocation methods.
<code>UPDATE_PLAN</code>	Updates a resource plan's comment information.
<code>DELETE_PLAN</code>	Deletes a resource plan and its directives.
<code>DELETE_PLAN_CASCADE</code>	Deletes a resource plan and all of its descendents.
<code>CREATE_CONSUMER_GROUP</code>	Creates a resource consumer group.
<code>UPDATE_CONSUMER_GROUP</code>	Updates a consumer group's comment information.
<code>DELETE_CONSUMER_GROUP</code>	Deletes a consumer group.
<code>CREATE_PLAN_DIRECTIVE</code>	Specifies the resource plan directives that allocate resources to resource consumer groups within a plan or among subplans in a multilevel plan schema.
<code>UPDATE_PLAN_DIRECTIVE</code>	Updates plan directives.
<code>DELETE_PLAN_DIRECTIVE</code>	Deletes plan directives.
<code>CREATE_PENDING_AREA</code>	Creates a pending area (scratch area) within which changes can be made to a plan schema.
<code>VALIDATE_PENDING_AREA</code>	Validates the pending changes to a plan schema.
<code>CLEAR_PENDING_AREA</code>	Clears all pending changes from the pending area.
<code>SUBMIT_PENDING_AREA</code>	Submits all changes to a plan schema.
<code>SET_INITIAL_CONSUMER_GROUP</code>	Sets the initial consumer group for a user.

Procedure	Description
SWITCH_CONSUMER_GROUP_FOR_SESS	Switches the consumer group of a specific session.
SWITCH_CONSUMER_GROUP_FOR_USER	Switches the consumer group of all sessions belonging to a specific user.

You may, as an administrator with the `ADMIN` option, choose to grant the administrative privilege to other users or roles. This is possible using the `DBMS_RESOURCE_MANAGER_PRIVS` package. This package contains the procedures listed in the table below.

Procedure	Description
GRANT_SYSTEM_PRIVILEGE	Grants <code>ADMINISTER_RESOURCE_MANAGER</code> system privilege to a user or role.
REVOKE_SYSTEM_PRIVILEGE	Revokes <code>ADMINISTER_RESOURCE_MANAGER</code> system privilege from a user or role.
GRANT_SWITCH_CONSUMER_GROUP	Grants permission to a user, role, or <code>PUBLIC</code> to switch to a specified resource consumer group.
REVOKE_SWITCH_CONSUMER_GROUP	Revokes permission for a user, role, or <code>PUBLIC</code> to switch to a specified resource consumer group.

The following example grants the administrative privilege to user `scott`, but does not grant `scott` the `ADMIN` option. Therefore, `scott` can execute all of the procedures in the `DBMS_RESOURCE_MANAGER` package, but `scott` cannot use the `GRANT_SYSTEM_PRIVILEGE` procedure to grant the administrative privilege to others.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE -
  (GRANTEE_NAME => 'scott', PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER', -
   ADMIN_OPTION => FALSE);
```

You can revoke this privilege using the `REVOKE_SYSTEM_PRIVILEGE` procedure.

Note: The `ADMINISTER_RESOURCE_MANAGER` system privilege can only be granted or revoked by using the `DBMS_RESOURCE_MANAGER_PRIVS` package. It cannot be granted or revoked through the SQL `GRANT` or `REVOKE` statements.

The other procedures in the `DBMS_RESOURCE_MANAGER_PRIVS` package are discussed in ["Managing the Switch Privilege"](#) on page 27-22.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference*. contains detailed information about the Database Resource Manager packages:

- `DBMS_RESOURCE_MANAGER`
- `DBMS_RESOURCE_MANAGER_PRIVS`

Creating a Simple Resource Plan

You can quickly create a simple resource plan that will be adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure. This procedure enables you to create consumer groups and allocate resources to them by executing a single statement. Using this procedure, you are not required to invoke the procedures that are described in succeeding sections for creating a pending area, creating each consumer group individually, and specifying resource plan directives.

You can specify the following parameters for the `CREATE_SIMPLE_PLAN` procedure:

Parameter	Description
<code>SIMPLE_PLAN</code>	Name of the plan
<code>CONSUMER_GROUP1</code>	Consumer group name for first group
<code>GROUP1_CPU</code>	CPU resource allocated to this group
<code>CONSUMER_GROUP2</code>	Consumer group name for second group
<code>GROUP2_CPU</code>	CPU resource allocated to this group
<code>CONSUMER_GROUP3</code>	Consumer group name for third group
<code>GROUP3_CPU</code>	CPU resource allocated to this group
<code>CONSUMER_GROUP4</code>	Consumer group name for fourth group
<code>GROUP4_CPU</code>	CPU resource allocated to this group
<code>CONSUMER_GROUP5</code>	Consumer group name for fifth group
<code>GROUP5_CPU</code>	CPU resource allocated to this group
<code>CONSUMER_GROUP6</code>	Consumer group name for sixth group
<code>GROUP6_CPU</code>	CPU resource allocated to this group

Parameter	Description
CONSUMER_GROUP7	Consumer group name for seventh group
GROUP7_CPU	CPU resource allocated to this group
CONSUMER_GROUP8	Consumer group name for eighth group
GROUP8_CPU	CPU resource allocated to this group

Up to eight consumer groups can be specified using this procedure and the only plan directive that can be specified is for CPU. Each consumer group specified in the plan is allocated its CPU percentage at level 2. Also included in the plan are SYS_GROUP (an Oracle defined groups that is the initial consumer group for the users SYS and SYSTEM) and OTHER_GROUPS.

Example: Using the CREATE_SIMPLE_PLAN Procedure

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'simple_plan1',
CONSUMER_GROUP1 => 'mygroup1', GROUP1_CPU => 80,
CONSUMER_GROUP2 => 'mygroup2', GROUP2_CPU => 20);
END;
```

Executing the above statements creates the following plan:

Consumer Group	Level 1	Level 2	Level 3
SYS_GROUP	100%	-	-
mygroup1	-	80%	-
mygroup2	-	20%	-
OTHER_GROUPS	-	-	100%

Creating Complex Resource Plans

This section describes the actions and DBMS_RESOURCE_MANAGER procedures that you can use when your situation requires that you create more complex resource plans. It contains the following sections:

- [Using the Pending Area for Creating Plan Schemas](#)
- [Creating Resource Plans](#)
- [Creating Resource Consumer Groups](#)

- [Specifying Resource Plan Directives](#)

Using the Pending Area for Creating Plan Schemas

The first thing you must do to create or modify plan schemas is to create a **pending area**. This is a scratch area allowing you to stage your changes and to validate them before they are made active.

Creating a Pending Area

To create a pending area, you use the following statement:

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

In effect, what is really happening here is that you are making the pending area active and "loading" all existing, or active, plan schemas into the pending area so that they can be updated or new plans added. Active plan schemas are those schemas already stored in the data dictionary for use by the Database Resource Manager. If you attempt to update a plan or add a new plan without first activating (creating) the pending area, you will receive an error message notifying you that the pending area is not active.

Views are available for inspecting all active resource plan schemas as well as the pending ones. These views are listed in [Viewing Database Resource Manager Information](#) on page 27-31.

Validating Changes

At any time when you are making changes in the pending area you can call the validate procedure as shown here.

```
EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

This procedure checks whether changes that have been made are valid. The following rules must be adhered to, and are checked by the validate procedure:

1. No plan schema can contain any loops.
2. All plans and resource consumer groups referred to by plan directives must exist.
3. All plans must have plan directives that point to either plans or resource consumer groups.
4. All percentages in any given level must not add up to greater than 100.

5. A plan that is currently being used as a top plan by an active instance cannot be deleted.
6. The following plan directive parameters can appear only in plan directives that refer to resource consumer groups (not other resource plans):
 - `PARALLEL_DEGREE_LIMIT_P1`
 - `ACTIVE_SESS_POOL_P1`
 - `QUEUEING_P1`
 - `SWITCH_GROUP`
 - `SWITCH_TIME`
 - `SWITCH_ESTIMATE`
 - `MAX_EST_EXEC_TIME`
 - `UNDO_POOL`
7. There can be no more than 32 resource consumer groups in any active plan schema. Also, at most, a plan can have 32 children. All leaves of a top plan must be resource consumer groups; at the lowest level in a plan schema the plan directives must refer to consumer groups.
8. Plans and resource consumer groups cannot have the same name.
9. There must be a plan directive for `OTHER_GROUPS` somewhere in any active plan schema. This ensures that a session which is not part of any of the consumer groups included in the currently active plan is allocated resources (as specified by the `OTHER_GROUPS` directive).

You will receive an error message if any of the above rules are violated. You can then make changes to fix any problems and call the `validate` procedure again.

It is possible to create "orphan" consumer groups that have no plan directives referring to them. This allows the creation of consumer groups that will not currently be used, but may be part of some plan to be implemented in the future.

Submitting Changes

After you have validated your changes, call the `submit` procedure to make your changes active.

```
EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

The submit procedure also performs validation, so you do not necessarily need to make separate calls to the validate procedure. However, if you are making major changes to plan schemas, debugging problems is often easier if you incrementally validate your changes. No changes are submitted (made active) until validation is successful on all of the changes in the pending area.

The `SUBMIT_PENDING_AREA` procedure clears (deactivates) the pending area after successfully validating and committing the changes.

Note: A call to `SUBMIT_PENDING_AREA` may fail even if `VALIDATE_PENDING_AREA` succeeds. This can happen if, for example, a plan being deleted is loaded by an instance after a call to `VALIDATE_PENDING_AREA`, but before a call to `SUBMIT_PENDING_AREA`.

Clearing the Pending Area

There is also a procedure for clearing the pending area at any time. This statement causes all of your changes to be cleared from the pending area:

```
EXEC DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;
```

You must call the `CREATE_PENDING_AREA` procedure before you can again attempt to make changes.

Creating Resource Plans

When you create a resource plan, you can specify the following parameters:

Parameter	Description
PLAN	Name of the plan.
COMMENT	Any comment. This field is optional.
The following parameters are not required to be specified. The defaults are appropriate and the only values allowed at this time.	
CPU_MTH	CPU resource allocation method. <code>EMPHASIS</code> is the default and the only CPU method allowed at the resource plan level.

Parameter	Description
ACTIVE_SESS_POOL_MTH	Active session pool resource allocation method. Controls maximum concurrent users. ACTIVE_SESS_POOL_ABSOLUTE is the default and only method available.
PARALLEL_DEGREE_LIMIT_MTH	Resource allocation method for specifying a limit on the degree of parallelism of any operation. PARALLEL_DEGREE_LIMIT_ABSOLUTE is the default and only method available.
QUEUEING_MTH	Queuing resource allocation method. Controls order in which queued sessions will execute. FIFO_TIMEOUT is the default and only method available.

Oracle provides one resource plan, SYSTEM_PLAN, that contains a simple structure that may be adequate for some environments. It is illustrated later in "[An Oracle Supplied Plan](#)" on page 27-28.

See Also: *Oracle9i Database Concepts* contains detailed descriptions of the resource allocation methods

Creating a Plan

You create a plan using the CREATE_PLAN procedure. The following creates a plan called great_bread. You choose to use the default resource allocation methods.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'great_bread', -
    COMMENT => 'great plan');
```

Updating a Plan

Use the UPDATE_PLAN procedure to update plan information. If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary. The following statement updates the COMMENT parameter.

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN(PLAN => 'great_bread', -
    NEW_COMMENT => 'great plan for great bread');
```

Deleting a Plan

The DELETE_PLAN procedure deletes the specified plan as well as all the plan directives associated with it. The following statement deletes the great_bread plan and its directives.

```
EXEC DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
```

The resource consumer groups themselves are not deleted, but they are no longer associated with the `great_bread` plan.

The `DELETE_PLAN_CASCADE` procedure deletes the specified plan as well as all its descendants (plan directives, subplans, resource consumer groups). If `DELETE_PLAN_CASCADE` encounters an error, it will roll back, leaving the plan schema unchanged.

Creating Resource Consumer Groups

When you create a resource consumer group, you can specify the following parameters:

Parameter	Description
<code>CONSUMER_GROUP</code>	Name of the consumer group.
<code>COMMENT</code>	Any comment.
<code>CPU_MTH</code>	The CPU resource allocation method for consumer groups. The default is <code>ROUND-ROBIN</code> . This is the only method currently available for resource consumer groups.

There are two special consumer groups that are always present in the data dictionary, and they cannot be modified or deleted. These are:

- `DEFAULT_CONSUMER_GROUP`

This is the initial consumer group for all users/sessions that have not been explicitly assigned an initial consumer group. `DEFAULT_CONSUMER_GROUP` has switch privileges granted to `PUBLIC`; therefore, all users are automatically granted switch privilege for this consumer group (see ["Managing the Switch Privilege"](#) on page 27-22).

- `OTHER_GROUPS`

This consumer group cannot be explicitly assigned to a user. `OTHER_GROUPS` must have a resource directive specified in the schema of any active plan. This group applies collectively to all sessions that belong to a consumer group that is not part of the currently active plan schema, including `DEFAULT_CONSUMER_GROUP`.

Additionally, two other groups, `SYS_GROUP` and `LOW_GROUP`, are provided as part of the Oracle supplied `SYSTEM_PLAN` that is described in "[An Oracle Supplied Plan](#)" on page 27-28.

Creating a Consumer Group

You create a consumer group using the `CREATE_CONSUMER_GROUP` procedure. The following creates a consumer group called `sales`. Remember, the pending area must be active to execute this statement successfully.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (CONSUMER_GROUP => 'sales', -
          COMMENT => 'retail and wholesale sales');
```

Updating a Consumer Group

Use the `UPDATE_CONSUMER_GROUP` procedure to update consumer group information. If you do not specify the arguments for the `UPDATE_CONSUMER_GROUP` procedure, they remain unchanged in the data dictionary.

Deleting a Consumer Group

The `DELETE_CONSUMER_GROUP` procedure deletes the specified consumer group. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group will have the `DEFAULT_CONSUMER_GROUP` set as their initial consumer group. All currently running sessions belonging to a deleted consumer group will be switched to `DEFAULT_CONSUMER_GROUP`.

Specifying Resource Plan Directives

Resource plan directives assign consumer groups to resource plans and provide the parameters for each resource allocation method. When you create a resource plan directive, you can specify the following parameters

Parameter	Description
<code>PLAN</code>	Name of the resource plan.
<code>GROUP_OR_SUBPLAN</code>	Name of the consumer group or subplan.
<code>COMMENT</code>	Any comment.
<code>CPU_P1</code>	Specifies CPU percentage at the first level. Default is <code>NULL</code> for all CPU parameters.
<code>CPU_P2</code>	Specifies CPU percentage at the second level.

Parameter	Description
CPU_P3	Specifies CPU percentage at the third level.
CPU_P4	Specifies CPU percentage at the fourth level.
CPU_P5	Specifies CPU percentage at the fifth level.
CPU_P6	Specifies CPU percentage at the sixth level.
CPU_P7	Specifies CPU percentage at the seventh level.
CPU_P8	Specifies CPU percentage at the eighth level.
ACTIVE_SESS_POOL_P1	Specifies maximum number of concurrently active sessions for a consumer group. Default is UNLIMITED.
QUEUEING_P1	Specified time (in seconds) after which a job in the execution queue (waiting for execution) will timeout. Default is UNLIMITED.
PARALLEL_DEGREE_LIMIT_P1	Specifies a limit on the degree of parallelism for any operation. Default is UNLIMITED.
SWITCH_GROUP	Specifies consumer group to which this session is switched if other switch criteria is met. Default is NULL.
SWITCH_TIME	Specifies time (in seconds) that a session can execute before it is switched to another consumer group. Default in UNLIMITED.
SWITCH_ESTIMATE	If TRUE, tells Oracle to use its execution time estimate to automatically switch the consumer group of an operation prior to beginning its execution. Default is FALSE.
MAX_EST_EXEC_TIME	Specifies the maximum execution time (in seconds) allowed for a session. Default is UNLIMITED.
UNDO_POOL	Sets a maximum in kilobytes (K) on the total amount of undo generated by a consumer group. Default is UNLIMITED.

Creating a Resource Plan Directive

You use the `CREATE_PLAN_DIRECTIVE` to create a resource plan directive. The following statement creates a resource plan directive for plan `great_bread`.

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
```

```
GROUP_OR_SUBPLAN => 'sales', COMMENT => 'sales group', -
CPU_P1 => 60, PARALLEL_DEGREE_LIMIT_P1 => 4);
```

To complete the plan, similar to that shown in [Figure 27-1](#), execute the following statements:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
      GROUP_OR_SUBPLAN => 'market', COMMENT => 'marketing group',
      CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
      GROUP_OR_SUBPLAN => 'develop', COMMENT => 'development group',
      CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
      GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'this one is required',
      CPU_P1 => 0, CPU_P2 => 100);
END;
```

In this plan, consumer group `sales` has a maximum degree of parallelism of 4 for any operation, while none of the other consumer groups are limited in their degree of parallelism. Also, whenever there are leftover level 1 CPU resources, they are allocated (100%) to `OTHER_GROUPS`.

Updating Resource Plan Directives

Use the `UPDATE_PLAN_DIRECTIVE` procedure to update plan directives. This example changes CPU allocation for resource consumer group `develop`.

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
      GROUP_OR_SUBPLAN => 'develop', NEW_CPU_P1 => 15);
```

If you do not specify the arguments for the `UPDATE_PLAN_DIRECTIVE` procedure, they remain unchanged in the data dictionary.

Deleting Resource Plan Directives

To delete a resource plan directive, use the `DELETE_PLAN_DIRECTIVE` procedure

How Resource Plan Directives Interact

If there are multiple resource plan directives that refer to the same consumer group, then the following rules apply for specific cases:

1. The parallel degree limit for the consumer group will be the *minimum* of all the incoming values.

2. The active session pool for the consumer group will be the *sum* of all the incoming values and the queue timeout will be the *minimum* of all incoming timeout values.
3. If there is more than one switch group and more than one switch time, the Database Resource Manager will choose the *most restrictive* of all incoming values. Specifically:
 - `SWITCH_TIME = min` (all incoming `over_switch_time` values)
 - `SWITCH_ESTIMATE = TRUE` overrides `SWITCH_ESTIMATE = FALSE`

Note: If both plan directives specify the same switch time, but different switch group's, then the choice as to which group to switch to will be statically, yet arbitrarily, decided by the Database Resource Manager.

4. If a session is switched to another consumer group because it exceeds its switch time, that session will execute even if the active session pool for the new consumer group is full.
5. The maximum estimated execution time will be the most restrictive of all incoming values. Specifically:
 - `max_estimated_exec_time = min` (all incoming `max_estimated_exec_time` values)

Managing Resource Consumer Groups

Before you enable the Database Resource Manager, you must assign resource consumer groups to users. In addition to providing procedures to create, update, or delete the elements used by the Database Resource Manager, the `DBMS_RESOURCE_MANAGER` package contains the procedure to assign resource consumer groups to users. It also provides procedures that allow you to temporarily switch a user session to another consumer group.

The `DBMS_RESOURCE_MANAGER_PRIVS` package, described earlier for granting the Database Resource Manager system privilege, can also be used to grant the switch privilege to another user, who can then alter their own consumer group.

You do not use a pending area for any of the procedures discussed below.

Assigning an Initial Resource Consumer Group

The initial consumer group of a user is the consumer group to which any session created by that user initially belongs. The user's initial consumer group is automatically set to `DEFAULT_CONSUMER_GROUP` when the user is created.

A user (or `PUBLIC`) must be granted permission to switch to a specific consumer group before that consumer group can become the user's initial consumer group. This permission is called the switch privilege, and is explained in ["Managing the Switch Privilege"](#) on page 27-22. The switch privilege to an initial consumer group cannot come from a role granted to that user.

The following statements illustrate setting a user's initial consumer group.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', 'sales', -
    TRUE);
EXEC DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP('scott', 'sales');
```

Changing Resource Consumer Groups

There are two procedures, which are part of the `DBMS_RESOURCE_MANAGER` package, that allow administrators to change the resource consumer group of running sessions. Both of these procedures can also change the consumer group of any parallel execution server sessions associated with the coordinator's session. The changes made by these procedures pertain to current sessions only; they are not persistent. They also do not change the initial consumer groups for users.

Instead of killing a session of a user who is using excessive CPU, an administrator can instead change that user's consumer group to one that is allowed less CPU. Or, this switching can be enforced automatically, using automatic consumer group switching resource plan directives.

Switching a Session

The `SWITCH_CONSUMER_GROUP_FOR_SESS` causes the specified session to immediately be moved into the specified resource consumer group. In effect, this statement can raise or lower priority. The following statement changes the resource consumer group of a specific session to a new consumer group. The session identifier (`SID`) is 17, the session serial number (`SERIAL#`) is 12345, and the session is to be changed to the `high_priority` consumer group.

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345', -
    'high_priority');
```

The `SID`, session serial number, and current resource consumer group for a session are viewable using the `V$SESSION` data dictionary view.

Switching Sessions for a User

The `SWITCH_CONSUMER_GROUP_FOR_USER` procedure changes the resource consumer group for all sessions with a given user name.

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('scott', -  
    'low_group');
```

Managing the Switch Privilege

Using the `DBMS_RESOURCE_MANAGER_PRIVS` package, you can grant or revoke the switch privilege to a user, role, or `PUBLIC`. The switch privilege gives users the privilege to switch their current resource consumer group to a specified resource consumer group. The package also enables you to revoke the switch privilege.

The actual switching is done by executing a procedure in the `DBMS_SESSION` package. A user who has been granted the switch privilege (or a procedure owned by that user) can use the `SWITCH_CURRENT_CONSUMER_GROUP` procedure to switch to another resource consumer group. The new group must be one to which the user has been specifically authorized to switch.

Granting the Switch Privilege

The following example grants the privilege to switch to a consumer group. User `scott` is granted the privilege to switch to consumer group `bug_batch_group`.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', -  
    'bug_batch_group', TRUE);
```

User `scott` is also granted permission to grant switch privileges for `bug_batch_group` to others.

If you grant a user permission to switch to a particular consumer group, then that user can switch their current consumer group to the new consumer group.

If you grant a role permission to switch to a particular resource consumer group, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new consumer group.

If you grant `PUBLIC` the permission to switch to a particular consumer group, then any user can switch to that group.

If the `GRANT_OPTION` argument is `TRUE`, then users granted switch privilege for the consumer group can also grant switch privileges for that consumer group to others.

Revoking Switch Privileges

The following example revokes user `scott`'s privilege to switch to consumer group `bug_batch_group`.

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP ('scott', -
    'bug_batch_group');
```

If you revoke a user's switch privileges to a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail. If you revoke the initial consumer group from a user, then that user will automatically be part of the `DEFAULT_CONSUMER_GROUP` when logging in.

If you revoke a role's switch privileges to a consumer group, then any users who only had switch privilege for the consumer group through that role will not be able to subsequently switch to that consumer group.

If you revoke switch privileges to a consumer group from `PUBLIC`, then any users other than those who are explicitly assigned switch privileges either directly or through `PUBLIC`, will not be able to subsequently switch to that consumer group.

Using the `DBMS_SESSION` Package to Switch Consumer Group

If granted the switch privilege, users can switch their current consumer group using the `SWITCH_CURRENT_CONSUMER_GROUP` procedure in the `DBMS_SESSION` package.

This procedure enables users to switch to a consumer group for which they have the switch privilege. If the caller is another procedure, then this procedure enables users to switch to a consumer group for which the owner of that procedure has switch privileges.

The parameters for this procedure are:

Parameter	Description
<code>NEW_CONSUMER_GROUP</code>	The consumer group to which the user is switching.
<code>OLD_CONSUMER_GROUP</code>	An output parameter. Stores the name of the consumer group from which the user switched. Can be used to switch back later.

Parameter	Description
INITIAL_GROUP_ON_ERROR	Controls behavior if a switching error occurs. If TRUE, in the event of an error, the user is switched to the initial consumer group. If FALSE, raise an error.

The following example illustrates switching to a new consumer group. By printing the value of the output parameter `old_group`, we illustrate how the old consumer group name has been saved.

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
    DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('sales', old_group, FALSE);
    DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
END;
```

The following line is output:

```
OLD GROUP = DEFAULT_CONSUMER_GROUP
```

The `DBMS_SESSION` package can be used from within a PL/SQL application, thus allowing the application to change consumer groups, or effectively priority, dynamically.

Note: The Database Resource Manager also works in environments where a generic database user name is used to log on to an application. The `DBMS_SESSION` package can be called to switch a session's consumer group assignment at session startup, or as particular modules are called.

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

Enabling the Database Resource Manager

You enable the Database Resource Manager by setting the `RESOURCE_MANAGER_PLAN` initialization parameter. This parameter specifies the top plan, identifying the

plan schema to be used for this instance. If no plan is specified with this parameter, the Database Resource Manager is not activated. The following example activates the Database Resource Manager and assigns the top plan as `mydb_plan`.

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

You can also activate or deactivate the Database Resource Manager, or change the current top plan, using the `ALTER SYSTEM` statement. In this example, the top plan is specified as `mydb_plan`.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;
```

An error message is returned if the specified plan does not exist in the data dictionary.

To deactivate the Database Resource Manager, issue the following statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

Putting It All Together: Database Resource Manager Examples

This section provides some examples of resource plan schemas. The following examples are presented:

- [Multilevel Schema Example](#)
- [Example of Using Several Resource Allocation Methods](#)
- [An Oracle Supplied Plan](#)

Multilevel Schema Example

The following statements create a multilevel schema as illustrated in [Figure 27-3](#). They use default plan and resource consumer group methods.

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
  COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
  COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
  COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Online_group',
  COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Batch_group',
  COMMENT => 'Resource consumer group/method for batch job bug users sessions');
```

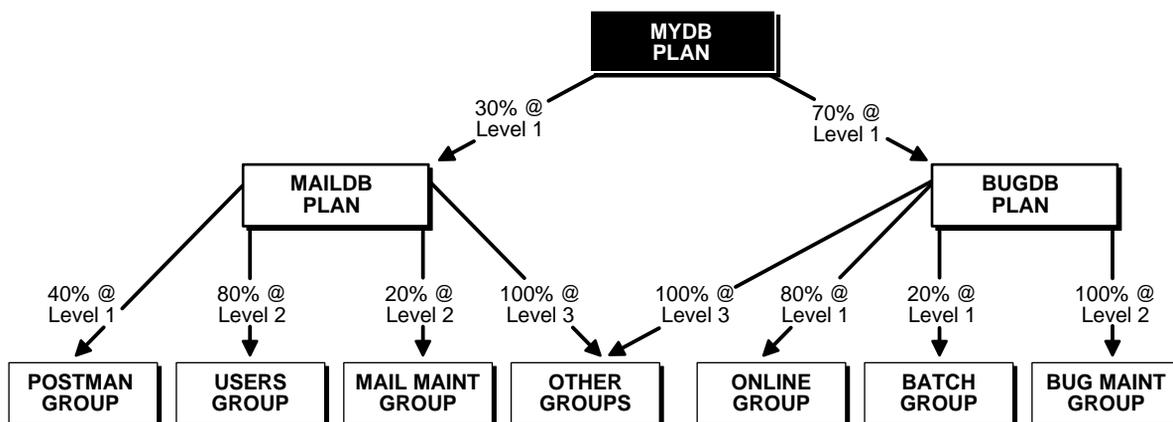
```

DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maintenance_group',
    COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_users_group',
    COMMENT => 'Resource consumer group/method for mail users sessions');
DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Postman_group',
    COMMENT => 'Resource consumer group/method for mail postman');
DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maintenance_group',
    COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
    GROUP_OR_SUBPLAN => 'Bug_Online_group',
    COMMENT => 'online bug users sessions at level 1', CPU_P1 => 80, CPU_P2=> 0,
    PARALLEL_DEGREE_LIMIT_P1 => 8);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
    GROUP_OR_SUBPLAN => 'Bug_Batch_group',
    COMMENT => 'batch bug users sessions at level 1', CPU_P1 => 20, CPU_P2 => 0,
    PARALLEL_DEGREE_LIMIT_P1 => 2);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
    GROUP_OR_SUBPLAN => 'Bug_Maintenance_group',
    COMMENT => 'bug maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 100,
    PARALLEL_DEGREE_LIMIT_P1 => 3);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
    CPU_P3 => 100);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
    GROUP_OR_SUBPLAN => 'Mail_Postman_group',
    COMMENT => 'mail postman at level 1', CPU_P1 => 40, CPU_P2 => 0,
    PARALLEL_DEGREE_LIMIT_P1 => 4);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
    GROUP_OR_SUBPLAN => 'Mail_users_group',
    COMMENT => 'mail users sessions at level 2', CPU_P1 => 0, CPU_P2 => 80,
    PARALLEL_DEGREE_LIMIT_P1 => 4);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
    GROUP_OR_SUBPLAN => 'Mail_Maintenance_group',
    COMMENT => 'mail maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 20,
    PARALLEL_DEGREE_LIMIT_P1 => 2);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0,
    CPU_P3 => 100);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
    GROUP_OR_SUBPLAN => 'maildb_plan',
    COMMENT=> 'all mail users sessions at level 1', CPU_P1 => 30);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
    GROUP_OR_SUBPLAN => 'bugdb_plan',
    COMMENT => 'all bug users sessions at level 1', CPU_P1 => 70);
DEMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DEMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;

```

The preceding call to `VALIDATE_PENDING_AREA` is optional because the validation is implicitly performed in `SUBMIT_PENDING_AREA`.

Figure 27-3 Multilevel Schema



Example of Using Several Resource Allocation Methods

The example presented here could represent a plan for a database supporting a packaged ERP (Enterprise Resource Planning) or CRM (Customer Relationship Management). The work in such an environment can be highly varied. There may be a mix of short transactions and quick queries, in combination with longer running batch jobs that include large parallel queries. The goal is to give good response time to OLTP (Online Transaction Processing), while allowing batch jobs to run in parallel.

The plan is summarized in the following table.

Group	CPU Resource Allocation %	Active Session Pool Parameters	Automatic Switching Parameters	Max Estimated Execution Time	Undo Pool
oltp	Level 1: 80%		Switch to group: batch Switch time: 3 Use estimate: TRUE		Size: 200K

Group	CPU Resource Allocation %	Active Session Pool Parameters	Automatic Switching Parameters	Max Estimated Execution Time	Undo Pool
batch	Level 2: 100%	Pool size: 5 Timeout: 600		Time: 3600	
OTHER_GROUPS	Level 3: 100%				

The following statements create the above plan, which is named `erp_plan`:

```

BEGIN
DEMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DEMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', CPU_P1 => 80,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3, SWITCH_ESTIMATE => TRUE,
  UNDO_POOL => 200);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', CPU_P2 => 100,
  ACTIVE_SESS_POOL_P1 => 5, QUEUEING_P1 => 600,
  MAX_EST_EXEC_TIME => 3600);
DEMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', CPU_P3 => 100);
DEMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DEMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;

```

An Oracle Supplied Plan

Oracle provides one default resource manager plan, `SYSTEM_PLAN`, which gives priority to system sessions. `SYSTEM_PLAN` is defined as follows:

Resource Consumer Group	CPU Resource Allocation		
	Level 1	Level 2	Level 3
SYS_GROUP	100%	0%	0%
OTHER_GROUPS	0%	100%	0%

Resource Consumer Group	CPU Resource Allocation		
	Level 1	Level 2	Level 3
LOW_GROUP	0%	0%	100%

The Oracle provided groups in this plan are:

- `SYS_GROUP` is the initial consumer group for the users `SYS` and `SYSTEM`.
- `OTHER_GROUPS` applies collectively to all sessions that belong to a consumer group that is not part of the currently active plan schema.
- `LOW_GROUP` provides a group having lower priority than `SYS_GROUP` and `OTHER_GROUPS` in this plan. It is up to you to decide which user sessions will be part of `LOW_GROUP`. Switch privilege is granted to `PUBLIC` for this group.

These groups can be used, or not used, and can be modified or deleted.

You can use this simple Oracle provided plan if it is appropriate for your environment.

Monitoring and Tuning the Database Resource Manager

To effectively monitor and tune the Database Resource Manager, you must design a representative environment. The Database Resource Manager works best in large production environments in which system utilization is high. If a test places insufficient load on the system, measured CPU allocations can be very different from the allocations specified in the active resource plan.

Creating the Environment

To create a representative environment, there must be sufficient load (demand for CPU resources) to make CPU resources scarce. If the following rules are followed, the test environment should generate actual (measured) resource allocations that match those specified in the active resource plan.

1. Create the minimum number of concurrently running processes required to generate sufficient load. This is the larger of:
 - Four processes for each consumer group
 - $1.5 * (\text{number of processors})$ for each consumer group. If the result is not an integer, round up.

2. Each and every process must be capable of consuming all of the CPU resources allocated to the consumer group in which it runs. Write resource intensive programs that continue to spin no matter what happens. This can be as simple as:

```
BEGIN
DECLARE
  m NUMBER;
BEGIN
  FOR i IN 1..100000 LOOP
    FOR j IN 1..100000 LOOP
      m := sqrt(4567);
    END LOOP;
  END LOOP;
END;
END;
/
```

Why Is This Necessary to Produce Expected Results?

When every group can secure as much CPU resources as it demands, the Database Resource Manager first seeks to maximize system throughput, not to enforce allocation percentages. For example, consider the following conditions:

- There is only one process available to run for each consumer group.
- Each process runs continuously.
- There are four CPUs.

In this case, the measured CPU allocation to each consumer group will be 25%, no matter what the allocations specified in the active resource plan.

Another factor determines the calculation in (1) above. Processor affinity scheduling at the operating system level can distort CPU allocation on underutilized systems. This is explained in the following paragraphs.

Until the number of concurrently running processes reaches a certain level, typical operating system scheduling algorithms will prevent full utilization. The Database Resource Manager controls CPU usage by restricting the number of running processes. By deciding which processes are allowed to run and for what duration, the Database Resource Manager controls CPU resource allocation. When a CPU has resources available, and other processors are fully utilized, the operating system migrates processes to the underutilized processor, but not immediately.

With processor affinity, the operating system waits (for a time) to migrate processes, "hoping" that another process will be dispatched to run instead of forcing process migration from one CPU to another. On a fully loaded system with enough processes waiting, this strategy will work. In large production environments, processor affinity increases performance significantly, because invalidating the current CPU cache and then loading the new one is quite expensive. Since processes have processor affinity on most platforms, more processes than CPUs for each consumer group must be run. Otherwise, full system utilization is not possible.

Monitoring Results

Use the `V$RSRC_CONSUMER_GROUP` view to monitor CPU usage. It provides the cumulative amount of CPU time consumed by all sessions in each consumer group. It also provides a number of other measures helpful for tuning.

```
SQL> SELECT NAME, CONSUMED_CPU_TIME FROM V$RSRC_CONSUMER_GROUP;
```

NAME	CONSUMED_CPU_TIME
-----	-----
OTHER_GROUPS	14301
TEST_GROUP	8802
TEST_GROUP2	0

3 rows selected.

Viewing Database Resource Manager Information

The following table lists views that are associated with Database Resource Manager:

View	Description
DBA_RSRC_CONSUMER_GROUP_PRIVS USER_RSRC_CONSUMER_GROUP_PRIVS	DBA view lists all resource consumer groups and the users and roles to which they have been granted. USER view lists all resource consumer groups granted to the user.
DBA_RSRC_CONSUMER_GROUPS	Lists all resource consumer groups that exist in the database.
DBA_RSRC_MANAGER_SYSTEM_PRIVS USER_RSRC_MANAGER_SYSTEM_PRIVS	DBA view lists all users and roles that have been granted Database Resource Manager system privileges. USER view lists all the users that are granted system privileges for the DBMS_RESOURCE_MANAGER package.
DBA_RSRC_PLAN_DIRECTIVES	Lists all resource plan directives that exist in the database.
DBA_RSRC_PLANS	List all resource plans that exist in the database.

View	Description
DBA_USERS USERS_USERS	DBA view contains information about all users of the database. Specifically, for the Database Resource Manager, it contains the initial resource consumer group for the user. USER view contains information about the current user, and specifically, for the Database Resource Manager, it contains the current user's initial resource consumer group.
V\$ACTIVE_SESS_POOL_MTH	Displays all available active session pool resource allocation methods.
V\$PARALLEL_DEGREE_LIMIT_MTH	Displays all available parallel degree limit resource allocation methods.
V\$QUEUEING	Displays all available queuing resource allocation methods.
V\$RSRC_CONSUMER_GROUP	Displays information about active resource consumer groups. This view can be used for tuning.
V\$RSRC_CONSUMER_GROUP_CPU_MTH	Displays all available CPU resource allocation methods for resource consumer groups.
V\$RSRC_PLAN	Displays the names of all currently active resource plans.
V\$RSRC_PLAN_CPU_MTH	Displays all available CPU resource allocation methods for resource plans.
V\$SESSION	Lists session information for each current session. Specifically, lists the name of each current session's resource consumer group.

You can use these views for viewing privileges, viewing plan schemas, or you can monitor them to gather information for tuning the Database Resource Manager. Some examples of their use follow.

See Also: *Oracle9i Database Reference* for detailed information about the contents of each of these views

Viewing Consumer Groups Granted to Users or Roles

The `DBA_RSRC_CONSUMER_GROUP_PRIVS` view displays the consumer groups granted to users or roles. Specifically, it displays the groups to which a user or role is allowed to belong or be switched. For example, in the view shown below, user `scott` can belong to the consumer groups `market` or `sales`, he has the ability to assign (grant) other users to the `sales` group but not the `market` group. Neither group is his initial consumer group.

```
SQL> SELECT * FROM DBA_RSRC_CONSUMER_GROUP_PRIVS;
```

GRANTEE	GRANTED_GROUP	GRA	INI
PUBLIC	DEFAULT_CONSUMER_GROUP	YES	YES
PUBLIC	LOW_GROUP	NO	NO
SCOTT	MARKET	NO	NO
SCOTT	SALES	YES	NO
SYSTEM	SYS_GROUP	NO	YES

Scott was granted the ability to switch to these groups using the `DBMS_RESOURCE_MANAGER_PRIVS` package.

Viewing Plan Schema Information

This example shows using the `DBA_RSRC_PLANS` view to display all of the resource plans defined in the database. All of the plans displayed are active, meaning they are not staged in the pending area

```
SQL> SELECT PLAN,COMMENTS,STATUS FROM DBA_RSRC_PLANS;
```

PLAN	COMMENTS	STATUS
SYSTEM_PLAN	Plan to give system sessions priority	ACTIVE
BUGDB_PLAN	Resource plan/method for bug users sessions	ACTIVE
MAILDB_PLAN	Resource plan/method for mail users sessions	ACTIVE
MYDB_PLAN	Resource plan/method for bug and mail users sessions	ACTIVE
GREAT_BREAD	Great plan for great bread	ACTIVE
ERP_PLAN	Resource plan/method for ERP Database	ACTIVE

6 rows selected.

Viewing Current Consumer Groups for Sessions

You can use the `V$SESSION` view to display the consumer groups that are currently assigned to sessions.

```
SQL> SELECT SID,SERIAL#,USERNAME,RESOURCE_CONSUMER_GROUP FROM V$SESSION;
```

SID	SERIAL#	USERNAME	RESOURCE_CONSUMER_GROUP
.	.	.	.
11	136	SYS	SYS_GROUP

```
13      16570 SCOTT      SALES
```

```
10 rows selected.
```

Viewing the Currently Active Plans

This example sets `mydb_plan`, as created by the statements shown earlier in ["Multilevel Schema Example"](#) on page 27-25, as the top level plan. The `V$RSRC_PLAN` view is queried to display the currently active plans.

```
SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;
```

```
System altered.
```

```
SQL> SELECT * FROM V$RSRC_PLAN;
```

```
NAME
```

```
-----
```

```
MYDB_PLAN
```

```
MAILDB_PLAN
```

```
BUGDB_PLAN
```

Part VI

Distributed Database Management

Part VI discusses the management of a distributed database environment. It contains the following sections:

- [Chapter 28, "Distributed Database Concepts"](#)
- [Chapter 29, "Managing a Distributed Database"](#)
- [Chapter 30, "Developing Applications for a Distributed Database System"](#)
- [Chapter 31, "Distributed Transactions Concepts"](#)
- [Chapter 32, "Managing Distributed Transactions"](#)

Distributed Database Concepts

This chapter describes the basic concepts and terminology of Oracle's distributed database architecture. It contains the following topics:

- [Distributed Database Architecture](#)
- [Database Links](#)
- [Distributed Database Administration](#)
- [Transaction Processing in a Distributed System](#)
- [Distributed Database Application Development](#)
- [Character Set Support for Distributed Environments](#)

Distributed Database Architecture

A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle database. In a **heterogeneous distributed database system**, at least one of the databases is a non-Oracle database. Distributed databases use a **client/server** architecture to process information requests.

This section contains the following topics:

- [Homogenous Distributed Database Systems](#)
- [Heterogeneous Distributed Database Systems](#)
- [Client/Server Database Architecture](#)

Homogenous Distributed Database Systems

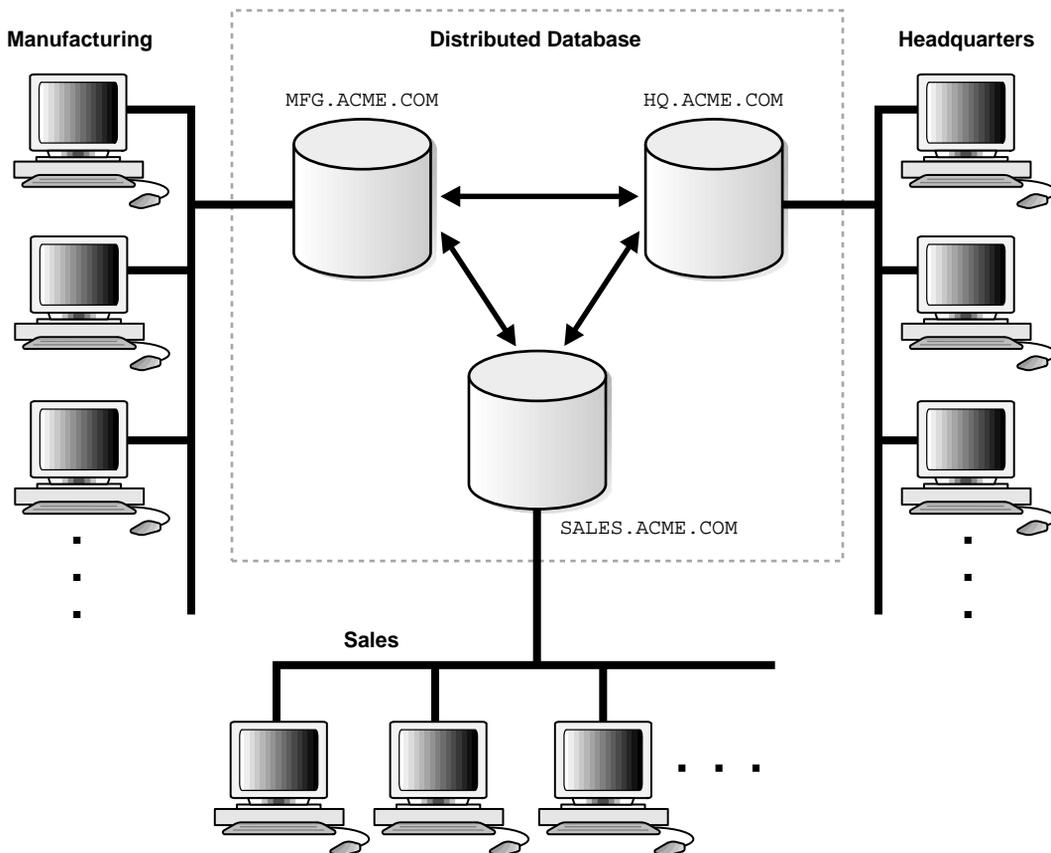
A homogenous distributed database system is a network of two or more Oracle databases that reside on one or more machines. [Figure 28-1](#) illustrates a distributed system that connects three databases: `hq`, `mfg`, and `sales`. An application can simultaneously access or modify the data in several databases in a single distributed environment. For example, a single query from a Manufacturing client on local database `mfg` can retrieve joined data from the `products` table on the local database and the `dept` table on the remote `hq` database.

For a client application, the location and platform of the databases are transparent. You can also create **synonyms** for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, if you are connected to database `mfg` but want to access data on database `hq`, creating a synonym on `mfg` for the remote `dept` table enables you to issue this query:

```
SELECT * FROM dept;
```

In this way, a distributed system gives the appearance of native data access. Users on `mfg` do not have to know that the data they access resides on remote databases.

Figure 28-1 Homogeneous Distributed Database



An Oracle distributed database system can incorporate Oracle databases of different versions. All supported releases of Oracle can participate in a distributed database system. Nevertheless, the applications that work with the distributed database must understand the functionality that is available at each node in the system. A distributed database application cannot expect an Oracle7 database to understand the SQL extensions that are only available with Oracle9*i*.

Distributed Databases Versus Distributed Processing

The terms **distributed database** and **distributed processing** are closely related, yet have distinct meanings. Their definitions are as follows:

- Distributed database

A set of databases in a distributed system that can appear to applications as a single data source.

- Distributed processing

The operations that occurs when an application distributes its tasks among different computers in a network. For example, a database application typically distributes front-end presentation tasks to client computers and allows a back-end database server to manage shared access to a database. Consequently, a distributed database application processing system is more commonly referred to as a client/server database application system.

Oracle distributed database systems employ a distributed processing architecture. For example, an Oracle database server acts as a client when it requests data that another Oracle database server manages.

Distributed Databases Versus Replicated Databases

The terms distributed database system and **database replication** are related, yet distinct. In a **pure** (that is, not replicated) distributed database, the system manages a single copy of all data and supporting database objects. Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real-time.

Note: This book discusses only pure distributed databases.

The term **replication** refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment.

Most commonly, replication is used to improve local database performance and protect the availability of applications because alternate data access options exist. For example, an application may normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible.

See Also: *Oracle9i Replication* for more information about Oracle's replication features

Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle system. To the application, the heterogeneous distributed database system appears as a single, local, Oracle database. The local Oracle database server hides the distribution and heterogeneity of the data.

The Oracle database server accesses the non-Oracle system using Oracle Heterogeneous Services in conjunction with an **agent**. If you access the non-Oracle data store using an Oracle Transparent Gateway, then the agent is a system-specific application. For example, if you include a Sybase database in an Oracle distributed system, then you need to obtain a Sybase-specific transparent gateway so that the Oracle databases in the system can communicate with it.

Alternatively, you can use **generic connectivity** to access non-Oracle data stores so long as the non-Oracle system supports the ODBC or OLE DB protocols.

Note: Other than the introductory material presented in this chapter, this book does not discuss Oracle Heterogeneous Services.

See *Oracle9i Heterogeneous Connectivity Administrator's Guide* for more detailed information about Heterogeneous Services.

Heterogeneous Services

Heterogeneous Services (HS) is an integrated component within the Oracle database server and the enabling technology for the current suite of Oracle Transparent Gateway products. HS provides the common architecture and administration mechanisms for Oracle gateway products and other heterogeneous access facilities. Also, it provides upwardly compatible functionality for users of most of the earlier Oracle Transparent Gateway releases.

Transparent Gateway Agents

For each non-Oracle system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle system. The agent is specific to the non-Oracle system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle and non-Oracle databases and uses the Heterogeneous Services component in the Oracle database server. The agent executes SQL and transactional requests at the non-Oracle system on behalf of the Oracle database server.

See Also: Your Oracle supplied gateway-specific documentation for information about transparent gateways

Generic Connectivity

Generic connectivity enables you to connect to non-Oracle data stores by using either a Heterogeneous Services ODBC agent or a Heterogeneous Services OLE DB agent—both are included with your Oracle product as a standard feature. Any data source compatible with the ODBC or OLE DB standards can be accessed using a generic connectivity agent.

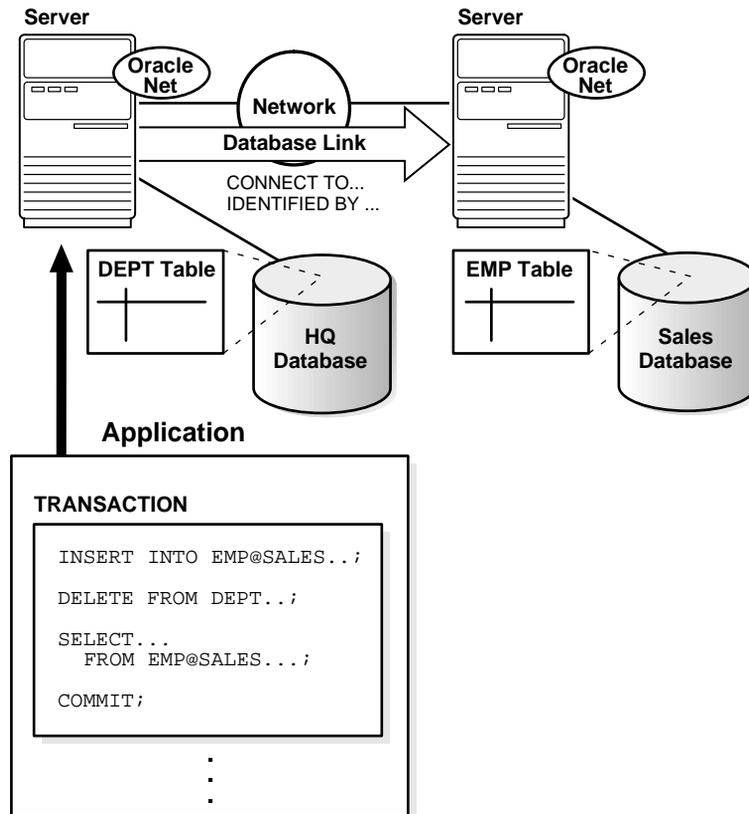
The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent. However, some data access features are only available with transparent gateway agents.

Client/Server Database Architecture

A database server is the Oracle software managing a database, and a client is an application that requests information from a server. Each computer in a network is a node that can host one or more databases. Each node in a distributed database system can act as a client, a server, or both, depending on the situation.

In [Figure 28-2](#), the host for the `hq` database is acting as a database server when a statement is issued against its local data (for example, the second statement in each transaction issues a statement against the local `dept` table), but is acting as a client when it issues a statement against remote data (for example, the first statement in each transaction is issued against the remote table `emp` in the `sales` database).

Figure 28-2 An Oracle Distributed Database System



A client can connect **directly** or **indirectly** to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. For example, if you connect to the `hq` database and access the `dept` table on this database as in Figure 28-2, you can issue the following:

```
SELECT * FROM dept;
```

This query is direct because you are not accessing an object on a remote database.

In contrast, an indirect connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, if you connect to the `hq` database but access the `emp` table on the remote `sales` database as in Figure 28-2, you can issue the following:

```
SELECT * FROM emp@sales;
```

This query is indirect because the object you are accessing is not on the database to which you are directly connected.

Database Links

The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database.

This section contains the following topics:

- [What Are Database Links?](#)
- [Why Use Database Links?](#)
- [Global Database Names in Database Links](#)
- [Names for Database Links](#)
- [Types of Database Links](#)
- [Users of Database Links](#)
- [Creation of Database Links: Examples](#)
- [Schema Objects and Database Links](#)
- [Database Link Restrictions](#)

What Are Database Links?

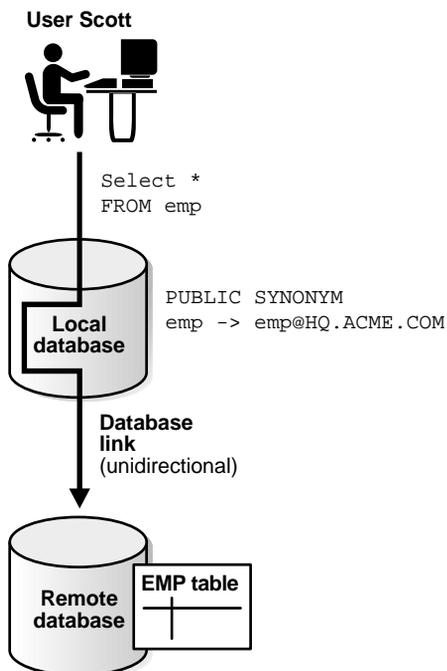
A database link is a pointer that defines a one-way communication path from an Oracle database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique **global database name** in the network domain. The global database name uniquely identifies a database server in a distributed system.

Figure 28-3 shows an example of user `scott` accessing the `emp` table on the remote database with the global name `hq.acme.com`:

Figure 28-3 Database Link



Database links are either private or public. If they are private, then only the user who created the link has access; if they are public, then all database users have access.

One principal difference among database links is the way that connections to a remote database occur. Users access a remote database through the following types of links:

Type of Link	Description
Connected user link	Users connect as themselves, which means that they must have an account on the remote database with the same username as their account on the local database.

Type of Link	Description
Fixed user link	Users connect using the username and password referenced in the link. For example, if Jane uses a fixed user link that connects to the hq database with the username and password scott/tiger, then she connects as scott, Jane has all the privileges in hq granted to scott directly, and all the default roles that scott has been granted in the hq database.
Current user link	A user connects as a global user. A local user can connect as a global user in the context of a stored procedure—without storing the global user's password in a link definition. For example, Jane can access a procedure that Scott wrote, accessing Scott's account and Scott's schema on the hq database. Current user links are an aspect of Oracle Advanced Security.

Create database links using the `CREATE DATABASE LINK` statement. After a link is created, you can use it to specify schema objects in SQL statements.

See Also:

- *Oracle9i SQL Reference* for syntax of the `CREATE DATABASE` statement
- *Oracle Advanced Security Administrator's Guide* for information about Oracle Advanced Security

What Are Shared Database Links?

A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.

When a local database is connected to a remote database through a database link, either database can run in dedicated or shared server mode. The following table illustrates the possibilities:

Local Database Mode	Remote Database Mode
Dedicated	Dedicated
Dedicated	Shared server
Shared server	Dedicated
Shared server	Shared server

A shared database link can exist in any of these four configurations. Shared links differ from standard database links in the following ways:

- Different users accessing the same schema object through a database link can share a network connection.
- When a user needs to establish a connection to a remote server from a particular server process, the process can reuse connections already established to the remote server. The reuse of the connection can occur if the connection was established on the same server process with the same database link—possibly in a different session. In a non-shared database link, a connection is not shared across multiple sessions.
- When you use a shared database link in a shared server configuration, a network connection is established directly out of the shared server process in the local server. For a non-shared database link on a local shared server, this connection would have been established through the local dispatcher, requiring context switches for the local dispatcher, and requiring data to go through the dispatcher.

See Also: *Oracle9i Net Services Administrator's Guide* for information about shared server

Why Use Database Links?

The great advantage of database links is that they allow users to access another user's objects in a remote database so that they are bounded by the privilege set of the object's owner. In other words, a local user can access a link to a remote database without having to be a user on the remote database.

For example, assume that employees submit expense reports to Accounts Payable (A/P), and further suppose that a user using an A/P application needs to retrieve information about employees from the `hq` database. The A/P users should be able to connect to the `hq` database and execute a stored procedure in the remote `hq` database that retrieves the desired information. The A/P users should not need to be `hq` database users to do their jobs; they should only be able to access `hq` information in a controlled way as limited by the procedure.

Database links allow you to grant limited access on remote databases to local users. By using current user links, you can create centrally managed global users whose password information is hidden from both administrators *and* non-administrators. For example, A/P users can access the `hq` database as `scott`, but unlike fixed user links, `scott`'s credentials are not stored where database users can see them.

By using fixed user links, you can create non-global users whose password information is stored in unencrypted form in the `LINK$` data dictionary table. Fixed user links are easy to create and require low overhead because there are no SSL or directory requirements, but a security risk results from the storage of password information in the data dictionary.

See Also:

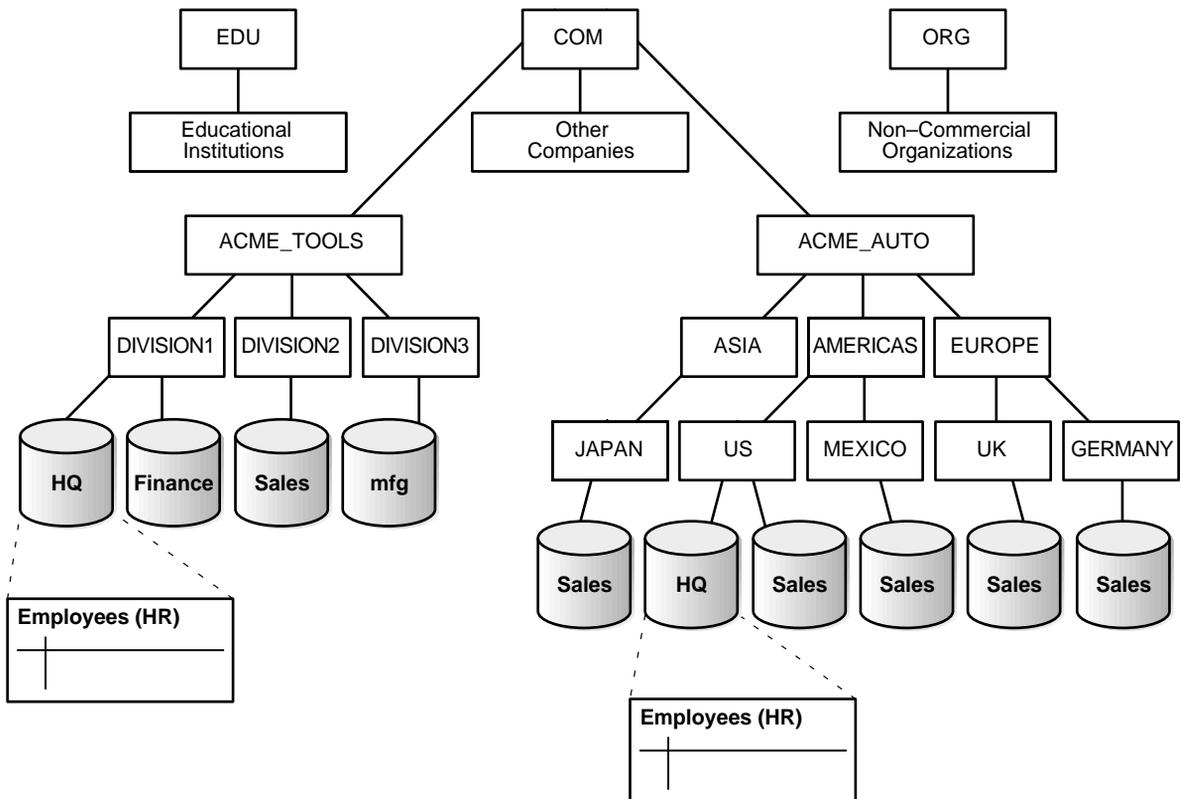
- ["Users of Database Links"](#) on page 28-16 for an explanation of database link users
- ["Viewing Information About Database Links"](#) for an explanation of how to hide passwords from non-administrators

Global Database Names in Database Links

To understand how a database link works, you must first understand what a global database name is. Each database in a distributed database is uniquely identified by its global database name. Oracle forms a database's global database name by prefixing the database's network domain, specified by the `DB_DOMAIN` initialization parameter at database creation, with the individual database name, specified by the `DB_NAME` initialization parameter.

For example, [Figure 28-4](#) illustrates a representative hierarchical arrangement of databases throughout a network.

Figure 28-4 Hierarchical Arrangement of Networked Databases



The name of a database is formed by starting at the leaf of the tree and following a path to the root. For example, the `mfg` database is in `division3` of the `acme_tools` branch of the `com` domain. The global database name for `mfg` is created by concatenating the nodes in the tree as follows:

- `mfg.division3.acme_tools.com`

While several databases can share an individual name, each database must have a unique global database name. For example, the network domains `us.americas.acme_auto.com` and `uk.europe.acme_auto.com` each contain a `sales` database. The global database naming system distinguishes the `sales` database in the `americas` division from the `sales` database in the `europa` division as follows:

- `sales.us.americas.acme_auto.com`
- `sales.uk.europe.acme_auto.com`

See Also: ["Managing Global Names in a Distributed System"](#) on page 29-2 to learn how to specify and change global database names

Names for Database Links

Typically, a database link has the same name as the global database name of the remote database that it references. For example, if the global database name of a database is `sales.us.oracle.com`, then the database link is also called `sales.us.oracle.com`.

When you set the initialization parameter `GLOBAL_NAMES` to `TRUE`, Oracle ensures that the name of the database link is the same as the global database name of the remote database. For example, if the global database name for `hq` is `hq.acme.com`, and `GLOBAL_NAMES` is `TRUE`, then the link name must be called `hq.acme.com`. Note that Oracle checks the domain part of the global database name as stored in the data dictionary, *not* the `DB_DOMAIN` setting in the initialization parameter file (see ["Changing the Domain in a Global Database Name"](#) on page 29-4).

If you set the initialization parameter `GLOBAL_NAMES` to `FALSE`, then you are not required to use global naming. You can then name the database link whatever you want. For example, you can name a database link to `hq.acme.com` as `foo`.

Note: Oracle Corporation recommends that you use global naming because many useful features, including Replication, require global naming.

After you have enabled global naming, database links are essentially transparent to users of a distributed database because the name of a database link is the same as the global name of the database to which the link points. For example, the following statement creates a database link in the local database to remote database `sales`:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';
```

See Also: *Oracle9i Database Reference* for more information about specifying the initialization parameter `GLOBAL_NAMES`

Types of Database Links

Oracle lets you create **private**, **public**, and **global** database links. These basic link types differ according to which users are allowed access to the remote database:

Type	Owner	Description
Private	User who created the link. View ownership data through: <ul style="list-style-type: none"> ▪ DBA_DB_LINKS ▪ ALL_DB_LINKS ▪ USER_DB_LINKS 	Creates link in a specific schema of the local database. Only the owner of a private database link or PL/SQL subprograms in the schema can use this link to access database objects in the corresponding remote database.
Public	User called PUBLIC. View ownership data through views shown above.	Creates a database-wide link. All users and PL/SQL subprograms in the database can use the link to access database objects in the corresponding remote database.
Global	User called PUBLIC. View ownership data through views shown above.	Creates a network-wide link. When an Oracle network uses Oracle Names, the names servers in the system automatically create and manage global database links for every Oracle database in the network. Users and PL/SQL subprograms in any database can use a global link to access objects in the corresponding remote database.

Determining the type of database links to employ in a distributed database depends on the specific requirements of the applications using the system. Consider these features when making your choice:

Type of Link	Features
Private database link	This link is more secure than a public or global link, because only the owner of the private link, or subprograms within the same schema, can use the link to access the remote database.
Public database link	When many users require an access path to a remote Oracle database, you can create a single public database link for all users in a database.
Global database link	When an Oracle network uses Oracle Names, an administrator can conveniently manage global database links for all databases in the system. Database link management is centralized and simple.

See Also:

- ["Specifying Link Types"](#) on page 29-9 to learn how to create different types of database links
- ["Viewing Information About Database Links"](#) on page 29-21 to learn how to access information about links

Users of Database Links

When creating the link, you determine which user should connect to the remote database to access the data. The following table explains the differences among the categories of users involved in database links:

User Type	Meaning	Sample Link Creation Syntax
Connected user	<p>A local user accessing a database link in which no fixed username and password have been specified. If <code>SYSTEM</code> accesses a public link in a query, then the connected user is <code>SYSTEM</code>, and Oracle connects to the <code>SYSTEM</code> schema in the remote database.</p> <p>Note: A connected user does not have to be the user who created the link, but is any user who is accessing the link.</p>	<pre>CREATE PUBLIC DATABASE LINK hq USING 'hq';</pre>
Current user	<p>A global user in a <code>CURRENT_USER</code> database link. The global user must be authenticated by an X.509 certificate (an SSL-authenticated enterprise user) or a password (a password-authenticated enterprise user), and be a user on both databases involved in the link. Current user links are an aspect of the Oracle Advanced Security option.</p> <p>See <i>Oracle Advanced Security Administrator's Guide</i> for information about global security</p>	<pre>CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';</pre>
Fixed user	<p>A user whose username/password is part of the link definition. If a link includes a fixed user, then the fixed user's username and password are used to connect to the remote database.</p>	<pre>CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY doe USING 'hq';</pre>

See Also: ["Specifying Link Users"](#) on page 29-11 to learn how to specify users where creating links

Connected User Database Links

Connected user links have no connect string associated with them. The advantage of a connected user link is that a user referencing the link connects to the remote

database as the same user. Furthermore, because no connect string is associated with the link, no password is stored in clear text in the data dictionary.

Connected user links have some disadvantages. Because these links require users to have accounts and privileges on the remote databases to which they are attempting to connect, they require more privilege administration for administrators. Also, giving users more privileges than they need violates the fundamental security concept of least privilege: users should only be given the privileges they need to perform their jobs.

The ability to use a connected user database link depends on several factors, chief among them whether the user is authenticated by Oracle using a password, or externally authenticated by the operating system or a network authentication service. If the user is externally authenticated, then the ability to use a connected user link also depends on whether the remote database accepts remote authentication of users, which is set by the `REMOTE_OS_AUTHENT` initialization parameter.

The `REMOTE_OS_AUTHENT` parameter operates as follows:

If <code>REMOTE_OS_AUTHENT</code> is...	Then...
TRUE for the remote database	An externally-authenticated user can connect to the remote database using a connected user database link.
FALSE for the remote database	An externally-authenticated user cannot connect to the remote database using a connected user database link unless a secure protocol or a network authentication service supported by the Oracle Advanced Security option is used.

Fixed User Database Links

A benefit of a fixed user link is that it connects a user in a primary database to a remote database with the security context of the user specified in the connect string. For example, local user `joe` can create a public database link in `joe`'s schema that specifies the fixed user `scott` with password `tiger`. If `jane` uses the fixed user link in a query, then `jane` is the user on the local database, but she connects to the remote database as `scott/tiger`.

Fixed user links have a username and password associated with the connect string. The username and password are stored in unencrypted form in the data dictionary in the `LINK$` table.

Caution: The fact that the username and password are stored in unencrypted form in the data dictionary creates a potential security weakness of fixed user database links.

If the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `TRUE`, a user with the `SELECT ANY TABLE` system privilege has access to the data dictionary, and thus the authentication associated with a fixed user is compromised.

The default for the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter is `FALSE`.

For an example of this security problem, assume that `jane` does not have privileges to use a private link that connects to the `hq` database as `scott/tiger`, but has `SELECT ANY TABLE` privilege on a database in which the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter is set to `TRUE`. She can select from `LINK$` and read that the connect string to `hq` is `scott/tiger`. If `jane` has an account on the host on which `hq` resides, then she can connect to the host and then connect to `hq` as `scott` using the password `tiger`. She will have all `scott`'s privileges if she connects locally and any audit records will be recorded as if she were `scott`.

See Also: ["System Privileges"](#) on page 25-2 for more information about system privileges and the `O7_DICTIONARY_ACCESSIBILITY` initialization parameter

Current User Database Links

Current user database links make use of a global user. A global user must be authenticated by an X.509 certificate or a password, and be a user on both databases involved in the link.

The user invoking the `CURRENT_USER` link does not have to be a global user. For example, if `jane` is authenticated (not as a global user) by password to the Accounts Payable database, she can access a stored procedure to retrieve data from the `hq` database. The procedure uses a current user database link, which connects her to `hq` as global user `scott`. User `scott` is a global user and authenticated through a certificate over SSL, but `jane` is not.

Note that current user database links have these consequences:

- If the current user database link is *not* accessed from within a stored object, then the current user is the same as the connected user accessing the link. For

example, if `scott` issues a `SELECT` statement through a current user link, then the current user is `scott`.

- When executing a stored object such as a procedure, view, or trigger that accesses a database link, the current user is the user that *owns* the stored object, and not the user that *calls* the object. For example, if `jane` calls procedure `scott.p` (created by `scott`), and a current user link appears *within* the called procedure, then `scott` is the current user of the link.
- If the stored object is an invoker-rights function, procedure, or package, then the invoker's authorization ID is used to connect as a remote user. For example, if user `jane` calls procedure `scott.p` (an invoker-rights procedure created by `scott`), and the link appears inside procedure `scott.p`, then `jane` is the current user.
- You cannot connect to a database as an enterprise user and then use a current user link in a stored procedure that exists in a shared, global schema. For example, if user `jane` accesses a stored procedure in the shared schema `guest` on database `hq`, she cannot use a current user link in this schema to log on to a remote database.

See Also:

- ["Distributed Database Security"](#) on page 28-24 for more information about security issues relating to database links
- *Oracle Advanced Security Administrator's Guide*

Creation of Database Links: Examples

Create database links using the `CREATE DATABASE LINK` statement. The table gives examples of SQL statements that create database links in a local database to the remote `sales.us.americas.acme_auto.com` database:

SQL Statement	Connects To Database	Connects As	Link Type
<code>CREATE DATABASE LINK sales.us.americas.acme_auto.com USING 'sales_us';</code>	<code>sales using net service name sales_us</code>	Connected user	Private connected user
<code>CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';</code>	<code>sales using service name am_sls</code>	Current global user	Private current user

SQL Statement	Connects To Database	Connects As	Link Type
CREATE DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger USING 'sales_us';	sales using net service name sales_us	scott using password tiger	Private fixed user
CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'rev';	sales using net service name rev	scott using password tiger	Public fixed user
CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING 'sales';	sales using net service name sales	scott using password tiger, authenticated as anupam using password bhide	Shared public fixed user

See Also:

- ["Creating Database Links"](#) on page 29-8 to learn how to create link
- *Oracle9i SQL Reference* for information about the CREATE DATABASE LINK statement syntax

Schema Objects and Database Links

After you have created a database link, you can execute SQL statements that access objects on the remote database. For example, to access remote object `emp` using database link `foo`, you can issue:

```
SELECT * FROM emp@foo;
```

You must also be authorized in the remote database to access specific remote objects.

Constructing properly formed object names using database links is an essential aspect of data manipulation in distributed systems.

Naming of Schema Objects Using Database Links

Oracle uses the global database name to name the schema objects globally using the following scheme:

schema.schema_object@global_database_name

where:

- *schema* is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.
- *schema_object* is a logical data structure like a table, index, view, synonym, procedure, package, or a database link.
- *global_database_name* is the name that uniquely identifies a remote database. This name must be the same as the concatenation of the remote database's initialization parameters `DB_NAME` and `DB_DOMAIN`, unless the parameter `GLOBAL_NAMES` is set to `FALSE`, in which case any name is acceptable.

For example, using a database link to database `sales.division3.acme.com`, a user or application can reference remote data as follows:

```
SELECT * FROM scott.emp@sales.division3.acme.com; # emp table in scott's schema
SELECT loc FROM scott.dept@sales.division3.acme.com;
```

If `GLOBAL_NAMES` is set to `FALSE`, then you can use any name for the link to `sales.division3.acme.com`. For example, you can call the link `foo`. Then, you can access the remote database as follows:

```
SELECT name FROM scott.emp@foo; # link name different from global name
```

Authorization for Accessing Remote Schema Objects

To access a remote schema object, you must be granted access to the remote object in the remote database. Further, to perform any updates, inserts, or deletes on the remote object, you must be granted the `SELECT` privilege on the object, along with the `UPDATE`, `INSERT`, or `DELETE` privilege. Unlike when accessing a local object, the `SELECT` privilege is necessary for accessing a remote object because Oracle has no remote describe capability. Oracle must do a `SELECT *` on the remote object in order to determine its structure.

Synonyms for Schema Objects

Oracle lets you create synonyms so that you can hide the database link name from the user. A synonym allows access to a table on a remote database using the same syntax that you would use to access a table on a local database. For example, assume you issue the following query against a table in a remote database:

```
SELECT * FROM emp@hq.acme.com;
```

You can create the synonym `emp` for `emp@hq.acme.com` so that you can issue the following query instead to access the same data:

```
SELECT * FROM emp;
```

See Also: ["Using Synonyms to Create Location Transparency"](#) on page 29-28 to learn how to create synonyms for objects specified using database links

Schema Object Name Resolution

To resolve application references to schema objects (a process called **name resolution**), Oracle forms object names hierarchically. For example, Oracle guarantees that each schema within a database has a unique name, and that within a schema each object has a unique name. As a result, a schema object's name is always unique within the database. Furthermore, Oracle resolves application references to an object's local name.

In a distributed database, a schema object such as a table is accessible to all applications in the system. Oracle extends the hierarchical naming model with global database names to effectively create **global object names** and resolve references to the schema objects in a distributed database system. For example, a query can reference a remote table by specifying its fully qualified name, including the database in which it resides.

For example, assume that you connect to the local database as user `SYSTEM`:

```
CONNECT SYSTEM/password@sales1
```

You then issue the following statements using database link `hq.acme.com` to access objects in the `scott` and `jane` schemas on remote database `hq`:

```
SELECT * FROM scott.emp@hq.acme.com;
INSERT INTO jane.accounts@hq.acme.com (acc_no, acc_name, balance)
VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.acme.com
SET balance = balance + 500;
DELETE FROM jane.accounts@hq.acme.com
WHERE acc_name = 'BOWER';
```

Database Link Restrictions

You *cannot* perform the following operations using database links:

- Grant privileges on remote objects

- Execute `DESCRIBE` operations on some remote objects. The following remote objects, however, do support `DESCRIBE` operations:
 - Tables
 - Views
 - Procedures
 - Functions
- Analyze remote objects
- Define or enforce referential integrity
- Grant roles to users in a remote database
- Obtain nondefault roles on a remote database. For example, if `jane` connects to the local database and executes a stored procedure that uses a fixed user link connecting as `scott`, `jane` receives `scott`'s default roles on the remote database. `Jane` cannot issue `SET ROLE` to obtain a nondefault role.
- Execute hash query joins that use shared server connections
- Use a current user link without authentication through SSL, password, or NT native authentication

Distributed Database Administration

The following sections explain some of the topics relating to database management in an Oracle distributed database system:

- [Site Autonomy](#)
- [Distributed Database Security](#)
- [Auditing Database Links](#)
- [Administration Tools](#)

See Also:

- [Chapter 29, "Managing a Distributed Database"](#) to learn how to administer homogenous systems
- *Oracle9i Heterogeneous Connectivity Administrator's Guide* to learn about heterogeneous services concepts

Site Autonomy

Site autonomy means that each server participating in a distributed database is administered independently from all other databases. Although several databases can work together, each database is a separate repository of data that is managed individually. Some of the benefits of site autonomy in an Oracle distributed database include:

- Nodes of the system can mirror the logical organization of companies or groups that need to maintain independence.
- Local administrators control corresponding local data. Therefore, each database administrator's domain of responsibility is smaller and more manageable.
- Independent failures are less likely to disrupt other nodes of the distributed database. No single database failure need halt all distributed operations or be a performance bottleneck.
- Administrators can recover from isolated system failures independently from other nodes in the system.
- A data dictionary exists for each local database—a global catalog is not necessary to access local data.
- Nodes can upgrade software independently.

Although Oracle permits you to manage each database in a distributed database system independently, you should not ignore the global requirements of the system. For example, you may need to:

- Create additional user accounts in each database to support the links that you create to facilitate server-to-server connections.
- Set additional initialization parameters such as `COMMIT_POINT_STRENGTH`, and `OPEN_LINKS`.

Distributed Database Security

Oracle supports all of the security features that are available with a nondistributed database environment for distributed database systems, including:

- Password authentication for users and roles
- Some types of external authentication for users and roles including:
 - Kerberos version 5 for connected user links
 - DCE for connected user links

- Login packet encryption for client-to-server and server-to-server connections

The following sections explain some additional topics to consider when configuring an Oracle distributed database system:

- [Authentication Through Database Links](#)
- [Authentication Without Passwords](#)
- [Supporting User Accounts and Roles](#)
- [Centralized User and Privilege Management](#)
- [Data Encryption](#)

See Also: *Oracle Advanced Security Administrator's Guide* for more information about external authentication

Authentication Through Database Links

Database links are either private or public, **authenticated** or **non-authenticated**. You create public links by specifying the `PUBLIC` keyword in the link creation statement. For example, you can issue:

```
CREATE PUBLIC DATABASE LINK foo USING 'sales' ;
```

You create authenticated links by specifying the `CONNECT TO` clause, `AUTHENTICATED BY` clause, or both clauses together in the database link creation statement. For example, you can issue:

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'sales' ;
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO mick IDENTIFIED BY jagger
    AUTHENTICATED BY david IDENTIFIED BY bowie USING 'sales' ;
```

This table describes how users access the remote database through the link:

Link Type	Authenticated?	Security Access
Private	No	When connecting to the remote database, Oracle uses security information (userid/password) taken from the local session. Hence, the link is a connected user database link. Passwords must be synchronized between the two databases.

Link Type	Authenticated?	Security Access
Private	Yes	The userid/password is taken from the link definition rather than from the local session context. Hence, the link is a fixed user database link. This configuration allows passwords to be different on the two databases, but the local database link password must match the remote database password. The password is stored in clear text on the local system catalog, adding a security risk.
Public	No	Works the same as a private non-authenticated link, except that all users can reference this pointer to the remote database.
Public	Yes	All users on the local database can access the remote database and all use the same userid/password to make the connection. Also, the password is stored in clear text in the local catalog, so you can see the password if you have sufficient privileges in the local database.

Authentication Without Passwords

When using a connected user or current user database link, you can use an external authentication source such as Kerberos to obtain **end-to-end security**. In end-to-end authentication, credentials are passed from server to server and can be authenticated by a database server belonging to the same domain. For example, if `jane` is authenticated externally on a local database, and wants to use a connected user link to connect as herself to a remote database, the local server passes the security ticket to the remote database.

Supporting User Accounts and Roles

In a distributed database system, you must carefully plan the user accounts and roles that are necessary to support applications using the system. Note that:

- The user accounts necessary to establish server-to-server connections must be available in all databases of the distributed database system.
- The roles necessary to make available application privileges to distributed database application users must be present in all databases of the distributed database system.

As you create the database links for the nodes in a distributed database system, determine which user accounts and roles each site needs to support server-to-server connections that use the links.

In a distributed environment, users typically require access to many network services. When you must configure separate authentications for each user to access each network service, security administration can become unwieldy, especially for large systems.

See Also: ["Creating Database Links"](#) on page 29-8 for more information about the user accounts that must be available to support different types of database links in the system

Centralized User and Privilege Management

Oracle provides different ways for you to manage the users and privileges involved in a distributed system. For example, you have these options:

- Enterprise user management. You can create global users who are authenticated through SSL or by using passwords, then manage these users and their privileges in a directory through an independent enterprise directory service.
- Network authentication service. This common technique simplifies security management for distributed environments. You can use the Oracle Advanced Security option to enhance Oracle Net and the security of an Oracle distributed database system. Windows NT native authentication is an example of a non-Oracle authentication solution.

See Also: *Oracle Advanced Security Administrator's Guide* for more information about global user security

Schema-Dependent Global Users One option for centralizing user and privilege management is to create the following:

- A global user in a centralized directory
- A user in every database that the global user must connect to

For example, you can create a global user called `fred` with the following SQL statement:

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England' ;
```

This solution allows a single global user to be authenticated by a centralized directory.

The schema-dependent global user solution has the consequence that you must create a user called `fred` on every database that this user must access. Because most users need permission to access an application schema but do not need their own

schemas, the creation of a separate account in each database for every global user creates significant overhead. Because of this problem, Oracle also supports schema-independent users, which are global users that access a single, generic schema in every database.

Schema-Independent Global Users Oracle supports functionality that allows a global user to be centrally managed by an enterprise directory service. Users who are managed in the directory are called **enterprise users**. This directory contains information about:

- Which databases in a distributed system an enterprise user can access
- Which role on each database an enterprise user can use
- Which schema on each database an enterprise user can connect to

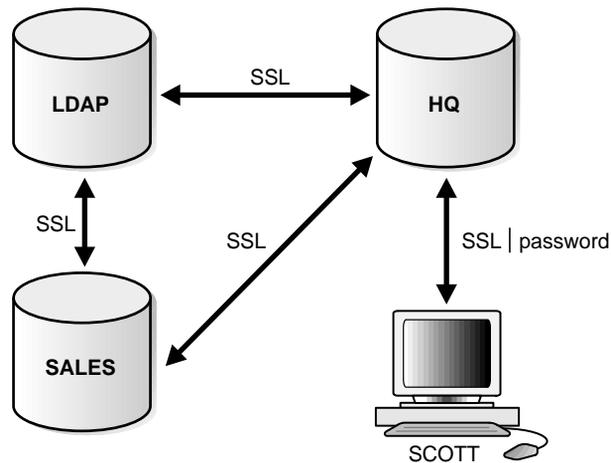
The administrator of each database is not required to create a global user account for each enterprise user on each database to which the enterprise user needs to connect. Instead, multiple enterprise users can connect to the same database schema, called a **shared schema**.

Note: You cannot access a current user database link in a shared schema.

For example, suppose `jane`, `bill`, and `scott` all use a human resources application. The `hr` application objects are all contained in the `guest` schema on the `hq` database. In this case, you can create a local global user account to be used as a shared schema. This global username, that is, shared schema name, is `guest`. `jane`, `bill`, and `scott` are all created as enterprise users in the directory service. They are also mapped to the `guest` schema in the directory, and can be assigned different authorizations in the `hr` application.

[Figure 28-5](#) illustrates an example of global user security using the enterprise directory service:

Figure 28–5 Global User Security



Assume that the enterprise directory service contains the following information on enterprise users for `hq` and `sales`:

Database	Role	Schema	Enterprise Users
hq	clerk1	guest	bill scott
sales	clerk2	guest	jane scott

Also, assume that the local administrators for `hq` and `sales` have issued statements as follows:

Database	CREATE Statements
hq	<pre>CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk1 GRANT select ON emp; CREATE PUBLIC DATABASE LINK sales_link CONNECT AS CURRENT_USER USING 'sales';</pre>
sales	<pre>CREATE USER guest IDENTIFIED GLOBALLY AS ''; CREATE ROLE clerk2 GRANT select ON dept;</pre>

Assume that enterprise user `scott` requests a connection to local database `hq` in order to execute a distributed transaction involving `sales`. The following steps occur (not necessarily in this exact order):

1. Enterprise user `scott` is authenticated using SSL or a password.
2. User `scott` issues the following statement:

```
SELECT e.ename, d.loc
FROM emp e, dept@sales_link d
WHERE e.deptno=d.deptno;
```
3. Databases `hq` and `sales` mutually authenticate one another using SSL.
4. Database `hq` queries the enterprise directory service to determine whether enterprise user `scott` has access to `hq`, and discovers `scott` can access local schema `guest` using role `clerk1`.
5. Database `sales` queries the enterprise directory service to determine whether enterprise user `scott` has access to `sales`, and discovers `scott` can access local schema `guest` using role `clerk2`.
6. Enterprise user `scott` logs into `sales` to schema `guest` with role `clerk2` and issues a `SELECT` to obtain the required information and transfer it to `hq`.
7. Database `hq` receives the requested data from `sales` and returns it to the client `scott`.

See Also: *Oracle Advanced Security Administrator's Guide* for more information about enterprise user security

Data Encryption

The Oracle Advanced Security option also enables Oracle Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4 or the Data Encryption Standard (DES) encryption algorithm.

To ensure that data has not been modified, deleted, or replayed during transmission, the security services of the Oracle Advanced Security option can generate a cryptographically secure message digest and include it with each packet sent across the network.

See Also: *Oracle Advanced Security Administrator's Guide* for more information about these and other features of the Oracle Advanced Security option

Auditing Database Links

You must always perform auditing operations locally. That is, if a user acts in a local database and accesses a remote database through a database link, the local actions are audited in the local database, and the remote actions are audited in the remote database—provided appropriate audit options are set in the respective databases.

The remote database cannot determine whether a successful connect request and subsequent SQL statements come from another server or from a locally connected client. For example, assume the following:

- Fixed user link `hq.acme.com` connects local user `jane` to the remote `hq` database as remote user `scott`.
- User `scott` is audited on the remote database.

Actions performed during the remote database session are audited as if `scott` were connected locally to `hq` and performing the same actions there. You must set audit options in the remote database to capture the actions of the username—in this case, `scott` on the `hq` database—embedded in the link if the desired effect is to audit what `jane` is doing in the remote database.

Note: You can audit the global username for global users.

You cannot set local auditing options on remote objects. Therefore, you cannot audit use of a database link, although access to remote objects can be audited on the remote database.

Administration Tools

The database administrator has several choices for tools to use when managing an Oracle distributed database system:

- [Enterprise Manager](#)
- [Third-Party Administration Tools](#)
- [SNMP Support](#)

Enterprise Manager

Enterprise Manager is Oracle's database administration tool that provides a graphical user interface (GUI). Enterprise Manager provides administrative

functionality for distributed databases through an easy-to-use interface. You can use Enterprise Manager to:

- Administer multiple databases. You can use Enterprise Manager to administer a single database or to simultaneously administer multiple databases.
- Centralize database administration tasks. You can administer both local and remote databases running on any Oracle platform in any location worldwide. In addition, these Oracle platforms can be connected by any network protocols supported by Oracle Net.
- Dynamically execute SQL, PL/SQL, and Enterprise Manager commands. You can use Enterprise Manager to enter, edit, and execute statements. Enterprise Manager also maintains a history of statements executed.

Thus, you can reexecute statements without retyping them, a particularly useful feature if you need to execute lengthy statements repeatedly in a distributed database system.

- Manage security features such as global users, global roles, and the enterprise directory service.

Third-Party Administration Tools

Currently more than 60 companies produce more than 150 products that help manage Oracle databases and networks, providing a truly open environment.

SNMP Support

Besides its network administration capabilities, Oracle **Simple Network Management Protocol (SNMP)** support allows an Oracle database server to be located and queried by any SNMP-based network management system. SNMP is the accepted standard underlying many popular network management systems such as:

- HP's OpenView
- Digital's POLYCENTER Manager on NetView
- IBM's NetView/6000
- Novell's NetWare Management System
- SunSoft's SunNet Manager

See Also: *Oracle SNMP Support Reference Guide* for more information about SNMP

Transaction Processing in a Distributed System

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user.

A **remote transaction** contains only statements that access a single remote node. A **distributed transaction** contains statements that access more than one node.

The following sections define important concepts in transaction processing and explain how transactions access data in a distributed database:

- [Remote SQL Statements](#)
- [Distributed SQL Statements](#)
- [Shared SQL for Remote and Distributed Statements](#)
- [Remote Transactions](#)
- [Distributed Transactions](#)
- [Two-Phase Commit Mechanism](#)
- [Database Link Name Resolution](#)
- [Schema Object Name Resolution](#)

Remote SQL Statements

A **remote query** statement is a query that selects information from one or more remote tables, all of which reside at the same remote node. For example, the following query accesses data from the `dept` table in the `scott` schema of the remote `sales` database:

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

A **remote update** statement is an update that modifies data in one or more tables, all of which are located at the same remote node. For example, the following query updates the `dept` table in the `scott` schema of the remote `sales` database:

```
UPDATE scott.dept@mktng.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

Note: A remote update can include a subquery that retrieves data from one or more remote nodes, but because the update happens at only a single remote node, the statement is classified as a remote update.

Distributed SQL Statements

A **distributed query** statement retrieves information from two or more nodes. For example, the following query accesses data from the local database as well as the remote `sales` database:

```
SELECT ename, dname
FROM scott.emp e, scott.dept@sales.us.americas.acme_auto.com d
WHERE e.deptno = d.deptno;
```

A **distributed update** statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes. For example, the following PL/SQL program unit updates tables on the local database and the remote `sales` database:

```
BEGIN
UPDATE scott.dept@sales.us.americas.acme_auto.com
SET loc = 'NEW YORK'
WHERE deptno = 10;
UPDATE scott.emp
SET deptno = 11
WHERE deptno = 10;
END;
COMMIT;
```

Oracle sends statements in the program to the remote nodes, and their execution succeeds or fails as a unit.

Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

See Also: *Oracle9i Database Concepts* for more information about shared SQL

Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node. For example, the following transaction contains two statements, each of which accesses the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.acme_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, this transaction updates the local database and the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

Note: If all statements of a transaction reference only a single remote node, the transaction is remote, not distributed.

Two-Phase Commit Mechanism

A database must guarantee that all statements in a transaction, distributed or nondistributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a nondistributed database are discussed in the *Oracle9i Database Concepts*. In a distributed database, Oracle must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

Oracle's **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

See Also: [Chapter 31, "Distributed Transactions Concepts"](#) for more information about Oracle's two-phase commit mechanism

Database Link Name Resolution

A **global object name** is an object specified using a database link. The essential components of a global object name are:

- Object name
- Database name
- Domain

The following table shows the components of an explicitly specified global database object name:

Statement	Object	Database	Domain
<code>SELECT * FROM joan.dept@sales.acme.com</code>	dept	sales	acme.com
<code>SELECT * FROM emp@mktg.us.acme.com</code>	emp	mktg	us.acme.com

Whenever a SQL statement includes a reference to a global object name, Oracle searches for a database link with a name that matches the database name specified in the global object name. For example, if you issue the following statement:

```
SELECT * FROM scott.emp@orders.us.acme.com;
```

Oracle searches for a database link called `orders.us.acme.com`. Oracle performs this operation to determine the path to the specified remote database.

Oracle always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.
2. Public database links in the local database.
3. Global database links (only if an Oracle Names Server is available).

Name Resolution When the Global Database Name Is Complete

Assume that you issue the following SQL statement, which specifies a complete global database name:

```
SELECT * FROM emp@prod1.us.oracle.com;
```

In this case, both the database name (`prod1`) and domain components (`us.oracle.com`) are specified, so Oracle searches for private, public, and global database links. Oracle searches only for links that match the specified global database name.

Name Resolution When the Global Database Name Is Partial

If any part of the domain is specified, Oracle assumes that a complete global database name is specified. If a SQL statement specifies a partial global database name (that is, only the database component is specified), Oracle appends the value in the `DB_DOMAIN` initialization parameter to the value in the `DB_NAME` initialization parameter to construct a complete name. For example, assume you issue the following statements:

```
CONNECT scott/tiger@locdb
SELECT * FROM scott.emp@orders;
```

If the network domain for `locdb` is `us.acme.com`, then Oracle appends this domain to `orders` to construct the complete global database name of `orders.us.acme.com`. Oracle searches for database links that match only the constructed global name. If a matching link is not found, Oracle returns an error and the SQL statement cannot execute.

Name Resolution When No Global Database Name Is Specified

If a global object name references an object in the local database and a database link name is *not* specified using the `@` symbol, then Oracle automatically detects that the object is local and does not search for or use database links to resolve the object reference. For example, assume that you issue the following statements:

```
CONNECT scott/tiger@locdb
SELECT * from scott.emp;
```

Because the second statement does not specify a global database name using a database link connect string, Oracle does not search for database links.

Terminating the Search for Name Resolution

Oracle does not necessarily stop searching for matching database links when it finds the first match. Oracle must search for matching private, public, and network database links until it determines a complete path to the remote database (both a remote account and service name).

The first match determines the remote schema as illustrated in the following table:

If you...	Then Oracle...	As in the example...
Do <i>not</i> specify the CONNECT clause	Uses a connected user database link	CREATE DATABASE LINK k1 USING 'prod'
Do specify the CONNECT TO ... IDENTIFIED BY clause	Uses a fixed user database link	CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY tiger USING 'prod'
Specify the CONNECT TO CURRENT_USER clause	Uses a current user database link	CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'
Do <i>not</i> specify the USING clause	Searches until it finds a link specifying a database string. If matching database links are found and a string is never identified, Oracle returns an error.	CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER

After Oracle determines a complete path, it creates a remote session—assuming that an identical connection is not already open on behalf of the same local session. If a session already exists, Oracle reuses it.

Schema Object Name Resolution

After the local Oracle database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement. The first match determines the remote schema according to the following rules:

If you use...	Then object resolution proceeds in the...
A fixed user database link	Schema specified in the link creation statement
A connected user database link	Connected user's remote schema
A current user database link	Current user's schema

If Oracle cannot find the object, then it checks public objects of the remote database. If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

The following are examples of global object name resolution in a distributed database system. For all the following examples, assume that:

Example of Global Object Name Resolution: Complete Object Name

This example illustrates how Oracle resolves a complete global object name and determines the appropriate path to the remote database using both a private and public database link. For this example, assume the following:

- The remote database is named `sales.division3.acme.com`.
- The local database is named `hq.division3.acme.com`.
- An Oracle Names Server (and therefore, global database links) is not available.
- A remote table `emp` is contained in the schema `tsmith`.

Consider the following statements issued by `scott` at the local database:

```
CONNECT scott/tiger@hq
```

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

Later, `JWARD` connects and issues the following statements:

```
CONNECT jward/bronco@hq
```

```
CREATE DATABASE LINK sales.division3.acme.com
CONNECT TO tsmith IDENTIFIED BY radio;
```

```
UPDATE tsmith.emp@sales.division3.acme.com
SET deptno = 40
WHERE deptno = 10;
```

Oracle processes the final statement as follows:

1. Oracle determines that a complete global object name is referenced in `jward`'s update statement. Therefore, the system begins searching in the local database for a database link with a matching name.
2. Oracle finds a matching private database link in the schema `jward`. Nevertheless, the private database link `jward.sales.division3.acme.com` does not indicate a complete path to the remote `sales` database, only a remote account. Therefore, Oracle now searches for a matching public database link.
3. Oracle finds the public database link in `scott`'s schema. From this public database link, Oracle takes the service name `dbstring`.
4. Combined with the remote account taken from the matching private fixed user database link, Oracle determines a complete path and proceeds to establish a connection to the remote `sales` database as user `tsmith/radio`.
5. The remote database can now resolve the object reference to the `emp` table. Oracle searches in the `tsmith` schema and finds the referenced `emp` table.
6. The remote database completes the execution of the statement and returns the results to the local database.

Example of Global Object Name Resolution: Partial Object Name

This example illustrates how Oracle resolves a partial global object name and determines the appropriate path to the remote database using both a private and public database link.

For this example, assume that:

- The remote database is named `sales.division3.acme.com`.
- The local database is named `hq.division3.acme.com`.
- An Oracle Names Server (and therefore, global database links) is not available.
- A table `emp` on the remote database `sales` is contained in the schema `tsmith`, but not in schema `scott`.
- A public synonym named `emp` resides at remote database `sales` and points to `tsmith.emp` in the remote database `sales`.
- The public database link in "[Example of Global Object Name Resolution: Complete Object Name](#)" on page 28-39 is already created on local database `hq`:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

Consider the following statements issued at local database `hq`:

```
CONNECT scott/tiger@hq

CREATE DATABASE LINK sales.division3.acme.com;

DELETE FROM emp@sales
WHERE empno = 4299;
```

Oracle processes the final `DELETE` statement as follows:

1. Oracle notices that a partial global object name is referenced in `scott`'s `DELETE` statement. It expands it to a complete global object name using the domain of the local database as follows:

```
DELETE FROM emp@sales.division3.acme.com
WHERE empno = 4299;
```

2. Oracle searches the local database for a database link with a matching name.
3. Oracle finds a matching *private* connected user link in the schema `scott`, but the private database link indicates no path at all. Oracle uses the connected username/password as the remote account portion of the path and then searches for and finds a matching *public* database link:

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

4. Oracle takes the database net service name `dbstring` from the public database link. At this point, Oracle has determined a complete path.
5. Oracle connects to the remote database as `scott/tiger` and searches for and does not find an object named `emp` in the schema `scott`.
6. The remote database searches for a public synonym named `emp` and finds it.
7. The remote database executes the statement and returns the results to the local database.

Global Name Resolution in Views, Synonyms, and Procedures

A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name. If the global object name is complete, then Oracle stores the definition of the object without expanding the global object name. If the name is partial, however, Oracle expands the name using the domain of the local database name.

The following table explains when Oracle completes the expansion of a partial global object name for views, synonyms, and program units:

If you...	Then Oracle...
Create a view	Does <i>not</i> expand partial global names—the data dictionary stores the exact text of the defining query. Instead, Oracle expands a partial global object name each time a statement that uses the view is parsed.
Create a synonym	Expands partial global names. The definition of the synonym stored in the data dictionary includes the expanded global object name.
Compile a program unit	Expands partial global names.

What Happens When Global Names Change

Global name changes can affect views, synonyms, and procedures that reference remote data using partial global object names. If the global name of the referenced database changes, views and procedures may try to reference a nonexistent or incorrect database. On the other hand, synonyms do not expand database link names at runtime, so they do not change.

Scenarios for Global Name Changes

For example, consider two databases named `sales.uk.acme.com` and `hq.uk.acme.com`. Also, assume that the `sales` database contains the following view and synonym:

```
CREATE VIEW employee_names AS
    SELECT ename FROM scott.emp@hr;
```

```
CREATE SYNONYM employee FOR scott.emp@hr;
```

Oracle expands the `employee` synonym definition and stores it as:

```
scott.emp@hr.uk.acme.com
```

Scenario 1: Both Databases Change Names First, consider the situation where both the Sales and Human Resources departments are relocated to the United States. Consequently, the corresponding global database names are both changed as follows:

Old Global Name	New Global Name
sales.uk.acme.com	sales.us.oracle.com
hq.uk.acme.com	hq.us.acme.com

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

Scenario 2: One Database Changes Names Now consider that only the Sales department is moved to the United States; Human Resources remains in the UK. Consequently, the corresponding global database names are both changed as follows:

Old Global Name	New Global Name
sales.uk.acme.com	sales.us.oracle.com
hq.uk.acme.com	no change

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.uk.acme.com

In this case, the defining query of the `employee_names` view expands to a non-existent global database name. On the other hand, the `employee` synonym continues to reference the correct database, `hq.uk.acme.com`.

Distributed Database Application Development

Application development in a distributed system raises issues that are not applicable in a nondistributed system. This section contains the following topics relevant for distributed application development:

- [Transparency in a Distributed Database System](#)
- [Remote Procedure Calls \(RPCs\)](#)
- [Distributed Query Optimization](#)

See Also: [Chapter 30, "Developing Applications for a Distributed Database System"](#) to learn how to develop applications for distributed systems

Transparency in a Distributed Database System

With minimal effort, you can develop applications that make an Oracle distributed database system transparent to users that work with the system. The goal of transparency is to make a distributed database system appear as though it is a single Oracle database. Consequently, the system does not burden developers and users of the system with complexities that would otherwise make distributed database application development challenging and detract from user productivity.

The following sections explain more about transparency in a distributed database system.

Location Transparency

An Oracle distributed database system has features that allow application developers and administrators to hide the physical location of database objects from applications and users. **Location transparency** exists when a user can universally refer to a database object such as a table, regardless of the node to which an application connects. Location transparency has several benefits, including:

- Access to remote data is simple, because database users do not need to know the physical location of database objects.
- Administrators can move database objects with no impact on end-users or existing database applications.

Typically, administrators and developers use synonyms to establish location transparency for the tables and supporting objects in an application schema. For example, the following statements create synonyms in a database for tables in another, remote database.

```
CREATE PUBLIC SYNONYM emp
  FOR scott.emp@sales.us.americas.acme_auto.com;
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.acme_auto.com;
```

Now, rather than access the remote tables with a query such as:

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.acme_auto.com e,
       scott.dept@sales.us.americas.acme_auto.com d
 WHERE e.deptno = d.deptno;
```

An application can issue a much simpler query that does not have to account for the location of the remote tables.

```
SELECT ename, dname
  FROM emp e, dept d
 WHERE e.deptno = d.deptno;
```

In addition to synonyms, developers can also use views and stored procedures to establish location transparency for applications that work in a distributed database system.

SQL and COMMIT Transparency

Oracle's distributed database architecture also provides query, update, and transaction transparency. For example, standard SQL statements such as SELECT, INSERT, UPDATE, and DELETE work just as they do in a nondistributed database environment. Additionally, applications control transactions using the standard SQL statements COMMIT, SAVEPOINT, and ROLLBACK—there is no requirement for complex programming or other special operations to provide distributed transaction control.

- The statements in a single transaction can reference any number of local or remote tables.
- Oracle guarantees that all nodes involved in a distributed transaction take the same action: they either all commit or all roll back the transaction.
- If a network or system failure occurs during the commit of a distributed transaction, the transaction is automatically and transparently resolved globally.

Specifically, when the network or system is restored, the nodes either all commit or all roll back the transaction.

Internal to Oracle, each committed transaction has an associated **system change number (SCN)** to uniquely identify the changes made by the statements within that transaction. In a distributed database, the SCNs of communicating nodes are coordinated when:

- A connection is established using the path described by one or more database links.
- A distributed SQL statement is executed.
- A distributed transaction is committed.

Among other benefits, the coordination of SCNs among the nodes of a distributed database system allows global distributed read-consistency at both the statement and transaction level. If necessary, global distributed time-based recovery can also be completed.

Replication Transparency

Oracle also provide many features to transparently replicate data among the nodes of the system. For more information about Oracle's replication features, see *Oracle9i Replication*.

Remote Procedure Calls (RPCs)

Developers can code PL/SQL packages and procedures to support applications that work with a distributed database. Applications can make local procedure calls to perform work at the local database and **remote procedure calls (RPCs)** to perform work at a remote database.

When a program calls a remote procedure, the local server passes all procedure parameters to the remote server in the call. For example, the following PL/SQL program unit calls the packaged procedure `del_emp` located at the remote `sales` database and passes it the parameter `1257`:

```
BEGIN
  emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;
```

In order for the RPC to succeed, the called procedure must exist at the remote site, and the user being connected to must have the proper privileges to execute the procedure.

When developing packages and procedures for distributed database systems, developers must code with an understanding of what program units should do at remote locations, and how to return the results to a calling application.

Distributed Query Optimization

Distributed query optimization is an Oracle feature that reduces the amount of data transfer required between sites when a transaction retrieves data from remote tables referenced in a distributed SQL statement.

Distributed query optimization uses Oracle's cost-based optimization to find or generate SQL expressions that extract only the necessary data from remote tables, process that data at a remote site or sometimes at the local site, and send the results to the local site for final processing. This operation reduces the amount of required data transfer when compared to the time it takes to transfer all the table data to the local site for processing.

Using various cost-based optimizer hints such as `DRIVING_SITE`, `NO_MERGE`, and `INDEX`, you can control where Oracle processes the data and how it accesses the data.

See Also: ["Using Cost-Based Optimization"](#) on page 30-5 for more information about cost-based optimization

Character Set Support for Distributed Environments

Oracle supports environments in which clients, Oracle database servers, and non-Oracle servers use different character sets. In Oracle, `NCHAR` support is provided for heterogeneous environments. You can set a variety of National Language Support (NLS) and Heterogeneous Services (HS) environment variables and initialization parameters to control data conversion between different character sets.

Character settings are defined by the following NLS and HS parameters:

Parameters	Environment	Defined For
<code>NLS_LANG</code> (environment variable)	Client-Server	Client
<code>NLS_LANGUAGE</code> <code>NLS_CHARACTERSET</code> <code>NLS_TERRITORY</code>	Client-Server Non-Heterogeneous Distributed Heterogeneous Distributed	Oracle database server

Parameters	Environment	Defined For
HS_LANGUAGE	Heterogeneous Distributed	Non-Oracle server Transparent gateway
NLS_NCHAR (environment variable) HS_NLS_NCHAR	Heterogeneous Distributed	Oracle database server Transparent gateway

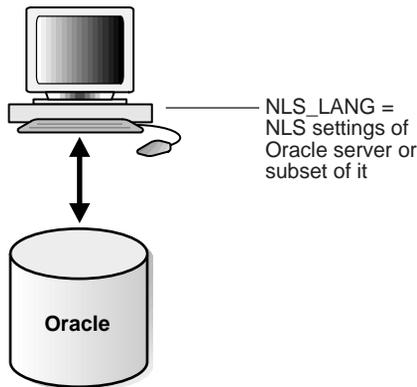
See Also:

- *Oracle9i Database Globalization Support Guide* for information about NLS parameters
- *Oracle9i Heterogeneous Connectivity Administrator's Guide* for information about HS parameters

Client/Server Environment

In a client/server environment, set the client character set to be the same as or a subset of the Oracle database server character set, as illustrated in [Figure 28-6](#):

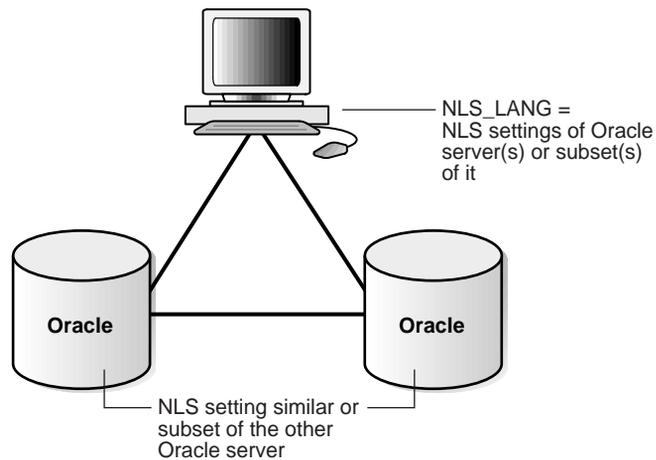
Figure 28-6 NLS Parameter Settings in a Client-Server Environment



Homogeneous Distributed Environment

In a non-heterogeneous environment, the client and server character sets should be either the same as or subsets of the main server character set, as illustrated in [Figure 28-7](#):

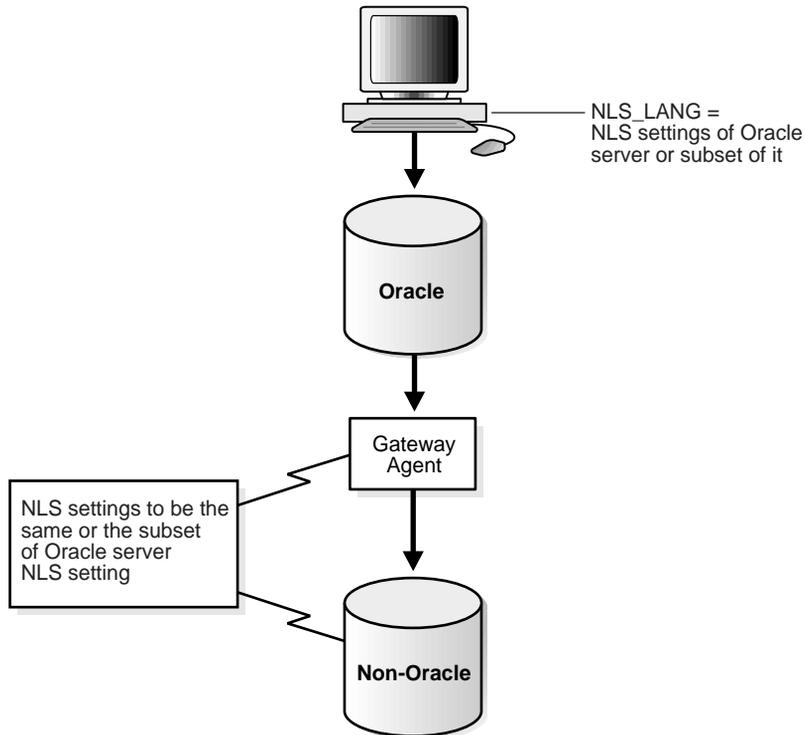
Figure 28–7 NLS Parameter Settings in a Homogeneous Environment



Heterogeneous Distributed Environment

In a heterogeneous environment, the NLS settings of the client, the transparent gateway, and the non-Oracle data source should be either the same or a subset of the Oracle database server character set as illustrated in [Figure 28–8](#). Transparent gateways have full globalization support.

Figure 28–8 NLS Parameter Settings in a Heterogeneous Environment



In a heterogeneous environment, only transparent gateways built with HS technology support complete NCHAR capabilities. Whether a specific transparent gateway supports NCHAR depends on the non-Oracle data source it is targeting. For information on how a particular transparent gateway handles NCHAR support, consult the system-specific transparent gateway documentation.

See Also: *Oracle9i Heterogeneous Connectivity Administrator's Guide* for more detailed information about Heterogeneous Services

Managing a Distributed Database

This chapter describes how to manage and maintain a distributed database system and contains the following topics:

- [Managing Global Names in a Distributed System](#)
- [Creating Database Links](#)
- [Creating Shared Database Links](#)
- [Managing Database Links](#)
- [Viewing Information About Database Links](#)
- [Creating Location Transparency](#)
- [Managing Statement Transparency](#)
- [Managing a Distributed Database: Scenarios](#)

Managing Global Names in a Distributed System

In a distributed database system, each database should have a unique **global database name**. Global database names uniquely identify a database in the system. A primary administration task in a distributed system is managing the creation and alteration of global database names.

This section contains the following topics:

- [Understanding How Global Database Names Are Formed](#)
- [Determining Whether Global Naming Is Enforced](#)
- [Viewing a Global Database Name](#)
- [Changing the Domain in a Global Database Name](#)
- [Changing a Global Database Name: Scenario](#)

Understanding How Global Database Names Are Formed

A global database name is formed from two components: a database name and a domain. The database name and the domain name are determined by the following initialization parameters at database creation:

Component	Parameter	Requirements	Example
Database name	DB_NAME	Must be eight characters or less.	sales
Domain containing the database	DB_DOMAIN	Must follow standard Internet conventions. Levels in domain names must be separated by dots and the order of domain names is from leaf to root, left to right.	us.acme.com

These are examples of valid global database names:

DB_NAME	DB_DOMAIN	Global Database Name
sales	au.oracle.com	sales.au.oracle.com
sales	us.oracle.com	sales.us.oracle.com
mktg	us.oracle.com	mktg.us.oracle.com
payroll	nonprofit.org	payroll.nonprofit.org

The `DB_DOMAIN` initialization parameter is only important at database creation time when it is used, together with the `DB_NAME` parameter, to form the database's global name. At this point, the database's global name is stored in the data dictionary. You must change the global name using an `ALTER DATABASE` statement, *not* by altering the `DB_DOMAIN` parameter in the initialization parameter file. It is good practice, however, to change the `DB_DOMAIN` parameter to reflect the change in the domain name before the next database startup.

Determining Whether Global Naming Is Enforced

The name that you give to a link on the local database depends on whether the remote database that you want to access enforces global naming. If the remote database enforces global naming, then you must use the remote database's global database name as the name of the link. For example, if you are connected to the local `hq` server and want to create a link to the remote `mfg` database, and `mfg` enforces global naming, then you must use `mfg`'s global database name as the link name.

You can also use service names as part of the database link name. For example, if you use the service names `sn1` and `sn2` to connect to database `hq.acme.com`, and `hq` enforces global naming, then you can create the following link names to `hq`:

- `HQ.ACME.COM@SN1`
- `HQ.ACME.COM@SN2`

See Also: ["Using Connection Qualifiers to Specify Service Names Within Link Names"](#) on page 29-13 for more information about using services names in link names

To determine whether global naming on a database is enforced on a database, either examine the database's initialization parameter file or query the `V$PARAMETER` view. For example, to see whether global naming is enforced on `mfg`, you could start a session on `mfg` and then create and execute the following `globalnames.sql` script (sample output included):

```
COL NAME FORMAT A12
COL VALUE FORMAT A6
SELECT NAME, VALUE FROM V$PARAMETER
WHERE NAME = 'global_names'
/
```

```
SQL> @globalnames
```

```
NAME          VALUE
-----
global_names  FALSE
```

Viewing a Global Database Name

Use the data dictionary view `GLOBAL_NAME` to view the database's global name. For example, issue the following:

```
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.AU.ORACLE.COM
```

Changing the Domain in a Global Database Name

Use the `ALTER DATABASE` statement to change the domain in a database's global name. Note that after the database is created, changing the initialization parameter `DB_DOMAIN` has no effect on the global database name or on the resolution of database link names.

The following example shows the syntax for the renaming statement, where *database* is a database name and *domain* is the network domain:

```
ALTER DATABASE RENAME GLOBAL_NAME TO database.domain;
```

Use the following procedure to change the domain in a global database name:

1. Determine the current global database name. For example, issue:

```
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.AU.ORACLE.COM
```

2. Rename the global database name using an `ALTER DATABASE` statement. For example, enter:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.oracle.com;
```

3. Query the `GLOBAL_NAME` table to check the new name. For example, enter:

```
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.US.ORACLE.COM
```

Changing a Global Database Name: Scenario

In this scenario, you change the domain part of the global database name of the local database. You also create database links using partially-specified global names to test how Oracle resolves the names. You discover that Oracle resolves the partial names using the domain part of the current global database name of the local database, not the value for the initialization parameter `DB_DOMAIN`.

1. You connect to `SALES.US.ACME.COM` and query the `GLOBAL_NAME` data dictionary view to determine the database's current global name:

```
CONNECT SYSTEM/password@sales.us.acme.com
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.US.ACME.COM
```

2. You query the `V$PARAMETER` view to determine the current setting for the `DB_DOMAIN` initialization parameter:

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'db_domain';
```

```
NAME          VALUE
-----
db_domain    US.ACME.COM
```

3. You then create a database link to a database called `hq`, using only a partially-specified global name:

```
CREATE DATABASE LINK hq USING 'sales';
```

Oracle expands the global database name for this link by appending the domain part of the global database name of the *local* database to the name of the database specified in the link.

4. You query `USER_DB_LINKS` to determine which domain name Oracle uses to resolve the partially specified global database name:

```
SELECT DB_LINK FROM USER_DB_LINKS;
```

```
DB_LINK
-----
HQ.US.ACME.COM
```

This result indicates that the domain part of the global database name of the local database is `us.acme.com`. Oracle uses this domain in resolving partial database link names when the database link is created.

5. Because you have received word that the `sales` database will move to Japan, you rename the `sales` database to `sales.jp.acme.com`:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.acme.com;
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.JP.ACME.COM
```

6. You query `V$PARAMETER` again and discover that the value of `DB_DOMAIN` is *not* changed, although you renamed the domain part of the global database name:

```
SELECT NAME, VALUE FROM V$PARAMETER
       WHERE NAME = 'db_domain';
```

```
NAME          VALUE
-----
db_domain    US.ACME.COM
```

This result indicates that the value of the `DB_DOMAIN` initialization parameter is independent of the `ALTER DATABASE RENAME GLOBAL_NAME` statement. The `ALTER DATABASE` statement determines the domain of the global database name, not the `DB_DOMAIN` initialization parameter (although it is good practice to alter `DB_DOMAIN` to reflect the new domain name).

7. You create another database link to database `supply`, and then query `USER_DB_LINKS` to see how Oracle resolves the domain part of `supply`'s global database name:

```
CREATE DATABASE LINK supply USING 'supply';
SELECT DB_LINK FROM USER_DB_LINKS;
```

```
DB_LINK
-----
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
```

This result indicates that Oracle resolves the partially specified link name by using the domain `jp.acme.com`. This domain is used when the link is created because it is the domain part of the global database name of the local database. Oracle does *not* use the `DB_DOMAIN` initialization parameter setting when resolving the partial link name.

8. You then receive word that your previous information was faulty: `sales` will be in the `ASIA.JP.ACME.COM` domain, not the `JP.ACME.COM` domain. Consequently, you rename the global database name as follows:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.acme.com;
SELECT * FROM GLOBAL_NAME;
```

```
GLOBAL_NAME
-----
SALES.ASIA.JP.ACME.COM
```

9. You query `V$PARAMETER` to again check the setting for the parameter `DB_DOMAIN`:

```
SELECT NAME, VALUE FROM V$PARAMETER
       WHERE NAME = 'db_domain';
```

```
NAME          VALUE
-----
db_domain     US.ACME.COM
```

The result indicates that the domain setting in the parameter file is exactly the same as it was before you issued *either* of the `ALTER DATABASE RENAME` statements.

10. Finally, you create a link to the warehouse database and again query `USER_DB_LINKS` to determine how Oracle resolves the partially-specified global name:

```
CREATE DATABASE LINK warehouse USING 'warehouse';
SELECT DB_LINK FROM USER_DB_LINKS;
```

```
DB_LINK
-----
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
WAREHOUSE.ASIA.JP.ACME.COM
```

Again, you see that Oracle uses the domain part of the global database name of the local database to expand the partial link name during link creation.

Note: In order to correct the `supply` database link, it must be dropped and re-created.

See Also: *Oracle9i Database Reference* for more information about specifying the `DB_NAME` and `DB_DOMAIN` initialization parameters

Creating Database Links

To support application access to the data and schema objects throughout a distributed database system, you must create all necessary database links. This section contains the following topics:

- [Obtaining Privileges Necessary for Creating Database Links](#)
- [Specifying Link Types](#)
- [Specifying Link Users](#)
- [Using Connection Qualifiers to Specify Service Names Within Link Names](#)

Obtaining Privileges Necessary for Creating Database Links

A database link is a pointer in the local database that allows you to access objects on a remote database. To create a private database link, you must have been granted the proper privileges. The following table illustrates which privileges are required on which database for which type of link:

Privilege	Database	Required For
CREATE DATABASE LINK	Local	Creation of a private database link.
CREATE PUBLIC DATABASE LINK	Local	Creation of a public database link.
CREATE SESSION	Remote	Creation of any type of database link.

To see which privileges you currently have available, query `ROLE_SYS_PRIVS`. For example, you could create and execute the following `privs.sql` script (sample output included):

```
SELECT DISTINCT PRIVILEGE AS "Database Link Privileges"
```

```

FROM ROLE_SYS_PRIVS
WHERE PRIVILEGE IN ( 'CREATE SESSION', 'CREATE DATABASE LINK',
                    'CREATE PUBLIC DATABASE LINK' )
/

SQL> @privs

Database Link Privileges
-----
CREATE DATABASE LINK
CREATE PUBLIC DATABASE LINK
CREATE SESSION

```

Specifying Link Types

When you create a database link, you must decide who will have access to it. The following sections describe how to create the three basic types of links:

- [Creating Private Database Links](#)
- [Creating Public Database Links](#)
- [Creating Global Database Links](#)

Creating Private Database Links

To create a private database link, specify the following (where *link_name* is the global database name or an arbitrary link name):

```
CREATE DATABASE LINK link_name ...;
```

Following are examples of private database links:

This SQL Statement...	Creates...
<pre>CREATE DATABASE LINK supply.us.acme.com;</pre>	<p>A private link using the global database name to the remote supply database.</p> <p>The link uses the userid/password of the connected user. So if <i>scott</i> (identified by <i>tiger</i>) uses the link in a query, the link establishes a connection to the remote database as <i>scott/tiger</i>.</p>
<pre>CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY doe USING 'us_supply';</pre>	<p>A private fixed user link called <i>link_2</i> to the database with service name <i>us_supply</i>. The link connects to the remote database with the userid/password of <i>jane/doe</i> regardless of the connected user.</p>

This SQL Statement...	Creates...
<pre>CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';</pre>	<p>A private link called <code>link_1</code> to the database with service name <code>us_supply</code>. The link uses the <code>userid/password</code> of the current user to log onto the remote database.</p> <p>Note: The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links" on page 28-16). Current user links are part of the Oracle Advanced Security option.</p>

See Also: *Oracle9i SQL Reference* for CREATE DATABASE LINK syntax

Creating Public Database Links

To create a public database link, use the keyword `PUBLIC` (where *link_name* is the global database name or an arbitrary link name):

```
CREATE PUBLIC DATABASE LINK link_name ...;
```

Following are examples of public database links:

This SQL Statement...	Creates...
<pre>CREATE PUBLIC DATABASE LINK supply.us.acme.com;</pre>	<p>A public link to the remote <code>supply</code> database. The link uses the <code>userid/password</code> of the connected user. So if <code>scott</code> (identified by <code>tiger</code>) uses the link in a query, the link establishes a connection to the remote database as <code>scott/tiger</code>.</p>
<pre>CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_USER USING 'supply';</pre>	<p>A public link called <code>pu_link</code> to the database with service name <code>supply</code>. The link uses the <code>userid/password</code> of the current user to log onto the remote database.</p> <p>Note: The current user may not be the same as the connected user, and must be a global user on both databases involved in the link (see "Users of Database Links" on page 28-16).</p>
<pre>CREATE PUBLIC DATABASE LINK sales.us.acme.com CONNECT TO jane IDENTIFIED BY doe;</pre>	<p>A public fixed user link to the remote <code>sales</code> database. The link connects to the remote database with the <code>userid/password</code> of <code>jane/doe</code>.</p>

See Also: *Oracle9i SQL Reference* for CREATE PUBLIC DATABASE LINK syntax

Creating Global Database Links

You must define **global database links** in the Oracle Names Server. See the *Oracle9i Net Services Administrator's Guide* to learn how to create global database links.

Specifying Link Users

A database link defines a communication path from one database to another. When an application uses a database link to access a remote database, Oracle establishes a database session in the remote database on behalf of the local application request.

When you create a private or public database link, you can determine which schema on the remote database the link will establish connections to by creating fixed user, current user, and connected user database links.

Creating Fixed User Database Links

To create a **fixed user database link**, you embed the credentials (in this case, a username and password) required to access the remote database in the definition of the link:

```
CREATE DATABASE LINK ... CONNECT TO username IDENTIFIED BY password ...;
```

Following are examples of fixed user database links:

This SQL Statement...	Creates...
CREATE PUBLIC DATABASE LINK supply.us.acme.com CONNECT TO scott AS tiger;	A public link using the global database name to the remote supply database. The link connects to the remote database with the userid/password scott/tiger.
CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY doe USING 'finance';	A private fixed user link called foo to the database with service name finance. The link connects to the remote database with the userid/password jane/doe.

When an application uses a fixed user database link, the local server always establishes a connection to a fixed remote schema in the remote database. The local server also sends the fixed user's credentials across the network when an application uses the link to access the remote database.

Creating Connected User and Current User Database Links

Connected user and current user database links do not include credentials in the definition of the link. The credentials used to connect to the remote database can change depending on the user that references the database link and the operation performed by the application.

Note: For many distributed applications, you do not want a user to have privileges in a remote database. One simple way to achieve this result is to create a procedure that contains a fixed user or current user database link within it. In this way, the user accessing the procedure temporarily assumes someone else's privileges.

For an extended conceptual discussion of the distinction between connected users and current users, see "[Users of Database Links](#)" on page 28-16.

Creating a Connected User Database Link To create a connected user database link, omit the `CONNECT TO` clause. The following syntax creates a connected user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink ... [USING 'net_service_name'];
```

For example, to create a connected user database link, use the following syntax:

```
CREATE DATABASE LINK sales.division3.acme.com USING 'sales';
```

Creating a Current User Database Link To create a current user database link, use the `CONNECT TO CURRENT_USER` clause in the link creation statement. Current user links are only available through the Oracle Advanced Security option.

The following syntax creates a current user database link, where *dblink* is the name of the link and *net_service_name* is an optional connect string:

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink CONNECT TO CURRENT_USER
[USING 'net_service_name'];
```

For example, to create a connected user database link to the `sales` database, you might use the following syntax:

```
CREATE DATABASE LINK sales CONNECT TO CURRENT_USER USING 'sales';
```

Note: To use a current user database link, the current user must be a global user on both databases involved in the link.

See Also: *Oracle9i SQL Reference* for more syntax information about creating database links

Using Connection Qualifiers to Specify Service Names Within Link Names

In some situations, you may want to have several database links of the same type (for example, public) that point to the same remote database, yet establish connections to the remote database using different communication pathways. Some cases in which this strategy is useful are:

- A remote database is part of an Oracle Real Application Clusters configuration, so you define several public database links at your local node so that connections can be established to specific instances of the remote database.
- Some clients connect to the Oracle server using TCP/IP while others use DECNET.

To facilitate such functionality, Oracle allows you to create a database link with an optional service name in the database link name. When creating a database link, a service name is specified as the trailing portion of the database link name, separated by an @ sign, as in @sales. This string is called a **connection qualifier**.

For example, assume that remote database `hq.acme.com` is managed in a Oracle Real Application Clusters environment. The `hq` database has two instances named `hq_1` and `hq_2`. The local database can contain the following public database links to define pathways to the remote instances of the `hq` database:

```
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_1
  USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_2
  USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.acme.com
  USING 'string_to_hq';
```

Notice in the first two examples that a service name is simply a part of the database link name. The text of the service name does not necessarily indicate how a connection is to be established; this information is specified in the service name of the `USING` clause. Also notice that in the third example, a service name is not specified as part of the link name. In this case, just as when a service name is specified as part of the link name, the instance is determined by the `USING` string.

To use a service name to specify a particular instance, include the service name at the end of the global object name:

```
SELECT * FROM scott.emp@hq.acme.com@hq_1
```

Note that in this example, there are two @ symbols.

Creating Shared Database Links

Every application that references a remote server using a standard database link establishes a connection between the local database and the remote database. Many users running applications simultaneously can cause a high number of connections between the local and remote databases.

Shared database links enable you to limit the number of network connections required between the local server and the remote server.

This section contains the following topics:

- [Determining Whether to Use Shared Database Links](#)
- [Creating Shared Database Links](#)
- [Configuring Shared Database Links](#)

See Also: ["What Are Shared Database Links?"](#) on page 28-10 for a conceptual overview of shared database links

Determining Whether to Use Shared Database Links

Look carefully at your application and shared server configuration to determine whether to use shared links. A simple guideline is to use shared database links when the number of users accessing a database link is expected to be much larger than the number of server processes in the local database.

The following table illustrates three possible configurations involving database links:

Link Type	Server Mode	Consequences
Non-Shared	Dedicated/shared server	If your application uses a standard public database link, and 100 users simultaneously require a connection, then 100 direct network connections to the remote database are required.
Shared	shared server	If 10 shared server processes exist in the local shared server mode database, then 100 users that use the same database link require 10 or fewer network connections to the remote server. Each local shared server process may only need one connection to the remote server.

Link Type	Server Mode	Consequences
Shared	Dedicated	If 10 clients connect to a local dedicated server, and each client has 10 sessions on the same connection (thus establishing 100 sessions overall), and each session references the same remote database, then only 10 connections are needed. With a non-shared database link, 100 connections are needed.

Shared database links are not useful in all situations. Assume that only one user accesses the remote server. If this user defines a shared database link and 10 shared server processes exist in the local database, then this user can require up to 10 network connections to the remote server. Because the user can use each shared server process, each process can establish a connection to the remote server.

Clearly, a non-shared database link is preferable in this situation because it requires only one network connection. Shared database links lead to more network connections in single-user scenarios, so use shared links only when many users need to use the same link. Typically, shared links are used for public database links, but can also be used for private database links when many clients access the same local schema (and therefore the same private database link).

Creating Shared Database Links

To create a shared database link, use the keyword `SHARED` in the `CREATE DATABASE LINK` statement:

```
CREATE SHARED DATABASE LINK dblink_name
[CONNECT TO username IDENTIFIED BY password][[CONNECT TO CURRENT_USER]
AUTHENTICATED BY schema_name IDENTIFIED BY password
[USING 'service_name'];
```

The following example creates a fixed user, shared link to database `sales`, connecting as `scott` and authenticated as `keith`:

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY tiger
AUTHENTICATED BY keith IDENTIFIED BY richards
USING 'sales';
```

Whenever you use the keyword `SHARED`, the clause `AUTHENTICATED BY` is required. The schema specified in the `AUTHENTICATED BY` clause is only used for security reasons and can be considered a dummy schema. It is not affected when using shared database links, nor does it affect the users of the shared database link.

The `AUTHENTICATED BY` clause is required to prevent unauthorized clients from masquerading as a database link user and gaining access to privileged information.

See Also: *Oracle9i SQL Reference* for information about the `CREATE DATABASE LINK` statement

Configuring Shared Database Links

You can configure shared database links in the following ways:

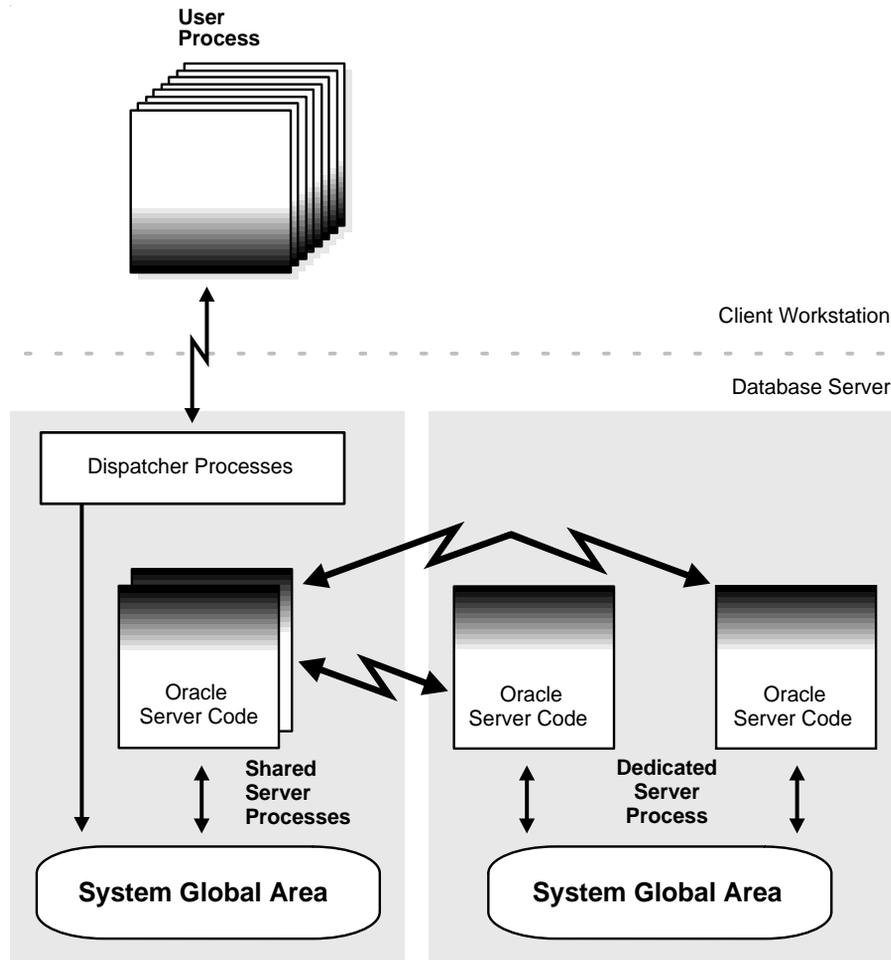
- [Creating Shared Links to Dedicated Servers](#)
- [Creating Shared Links to Shared Servers](#)

Creating Shared Links to Dedicated Servers

In the configuration illustrated in [Figure 29-1](#), a shared server process in the local server owns a dedicated remote server process. The advantage is that a direct network transport exists between the local shared server and the remote dedicated server. A disadvantage is that extra back-end server processes are needed.

Note: The remote server can either be a shared server or dedicated server. There is a dedicated connection between the local and remote servers. When the remote server is a shared server, you can force a dedicated server connection by using the `(SERVER=DEDICATED)` clause in the definition of the service name.

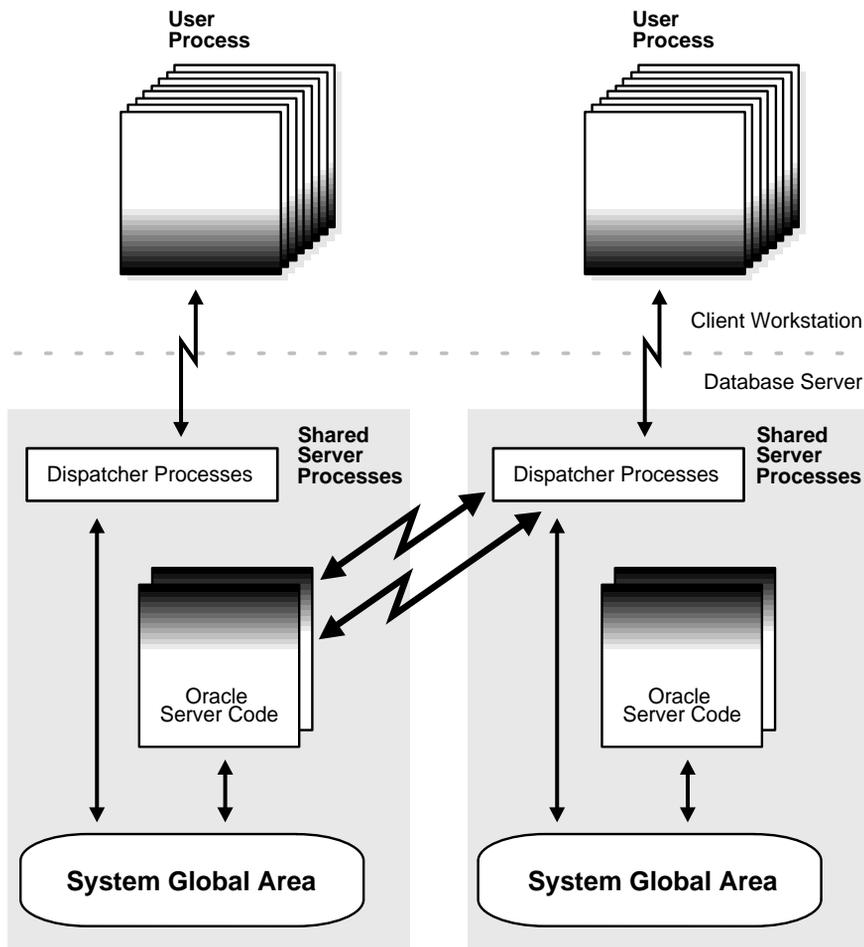
Figure 29-1 A Shared Database Link to Dedicated Server Processes



Creating Shared Links to Shared Servers

The configuration illustrated in [Figure 29-2](#) uses shared server processes on the remote server. This configuration eliminates the need for more dedicated servers, but requires the connection to go through the dispatcher on the remote server. Note that both the local and the remote server must be configured as shared servers.

Figure 29–2 Shared Database Link to Shared Server



See Also: *Oracle9i Net Services Administrator's Guide* for information about the shared server option

Managing Database Links

This section contains the following topics:

- [Closing Database Links](#)
- [Dropping Database Links](#)

- [Limiting the Number of Active Database Link Connections](#)

Closing Database Links

If you access a database link in a session, then the link remains open until you close the session. A link is open in the sense that a process is active on each of the remote databases accessed through the link. This situation has the following consequences:

- If 20 users open sessions and access the same public link in a local database, then 20 database link connections are open.
- If 20 users open sessions and each user accesses a private link, then 20 database link connections are open.
- If one user starts a session and accesses 20 different links, then 20 database link connections are open.

After you close a session, the links that were active in the session are automatically closed. You may have occasion to close the link manually. For example, close links when:

- The network connection established by a link is used infrequently in an application.
- The user session must be terminated.

If you want to close a link, issue the following statement, where *linkname* refers to the name of the link:

```
ALTER SESSION CLOSE DATABASE LINK linkname;
```

Note that this statement only closes the links that are active in your current session.

Dropping Database Links

You can drop a database link just as you can drop a table or view. If the link is private, then it must be in your schema. If the link is public, then you must have the `DROP PUBLIC DATABASE LINK` system privilege.

The statement syntax is as follows, where *dblink* is the name of the link:

```
DROP [PUBLIC] DATABASE LINK dblink;
```

Procedure for Dropping a Private Database Link

1. Connect to the local database using SQL*Plus. For example, enter:

```
CONNECT scott/tiger@local_db
```

2. Query `USER_DB_LINKS` to view the links that you own. For example, enter:

```
SELECT DB_LINK FROM USER_DB_LINKS;
```

```
DB_LINK
```

```
-----  
SALES.US.ORACLE.COM
```

```
MKTG.US.ORACLE.COM
```

```
2 rows selected.
```

3. Drop the desired link using the `DROP DATABASE LINK` statement. For example, enter:

```
DROP DATABASE LINK sales.us.oracle.com;
```

Procedure for Dropping a Public Database Link

1. Connect to the local database as a user with the `DROP PUBLIC DATABASE LINK` privilege. For example, enter:

```
CONNECT SYSTEM/password@local_db AS SYSDBA
```

2. Query `DBA_DB_LINKS` to view the public links. For example, enter:

```
SELECT DB_LINK FROM USER_DB_LINKS  
WHERE OWNER = 'PUBLIC';
```

```
DB_LINK
```

```
-----  
DBL1.US.ORACLE.COM
```

```
SALES.US.ORACLE.COM
```

```
INST2.US.ORACLE.COM
```

```
RMAN2.US.ORACLE.COM
```

```
4 rows selected.
```

3. Drop the desired link using the `DROP PUBLIC DATABASE LINK` statement. For example, enter:

```
DROP PUBLIC DATABASE LINK sales.us.oracle.com;
```

Limiting the Number of Active Database Link Connections

You can limit the number of connections from a user process to remote databases using the static initialization parameter `OPEN_LINKS`. This parameter controls the

number of remote connections that a single user session can use concurrently in distributed transactions.

Note the following considerations for setting this parameter:

- The value should be greater than or equal to the number of databases referred to in a single SQL statement that references multiple databases.
- Increase the value if several distributed databases are accessed over time. Thus, if you regularly access three database, set `OPEN_LINKS` to 3 or greater.
- The default value for `OPEN_LINKS` is 4. If `OPEN_LINKS` is set to 0, then no distributed transactions are allowed.

See Also: *Oracle9i Database Reference* for more information about `OPEN_LINKS`

Viewing Information About Database Links

The data dictionary of each database stores the definitions of all the database links in the database. You can use data dictionary tables and views to gain information about the links. This section contains the following topics:

- [Determining Which Links Are in the Database](#)
- [Determining Which Link Connections Are Open](#)

Determining Which Links Are in the Database

The following views show the database links that have been defined at the local database and stored in the data dictionary:

View	Purpose
<code>DBA_DB_LINKS</code>	Lists all database links in the database.
<code>ALL_DB_LINKS</code>	Lists all database links accessible to the connected user.
<code>USER_DB_LINKS</code>	Lists all database links owned by the connected user.

These data dictionary views contain the same basic information about database links, with some exceptions:

Column	Which Views?	Description
<code>OWNER</code>	All except <code>USER_*</code>	The user who created the database link. If the link is public, then the user is listed as <code>PUBLIC</code> .

Column	Which Views?	Description
DB_LINK	All	The name of the database link.
USERNAME	All	If the link definition includes a fixed user, then this column displays the username of the fixed user. If there is no fixed user, the column is NULL.
PASSWORD	Only USER_*	The password for logging into the remote database.
HOST	All	The net service name used to connect to the remote database.
CREATED	All	Creation time of the database link.

Any user can query `USER_DB_LINKS` to determine which database links are available to that user. Only those with additional privileges can use the `ALL_DB_LINKS` or `DBA_DB_LINKS` view.

The following script queries the `DBA_DB_LINKS` view to access link information:

```
COL OWNER FORMAT a10
COL USERNAME FORMAT A8 HEADING "USER"
COL DB_LINK FORMAT A30
COL HOST FORMAT A7 HEADING "SERVICE"
SELECT * FROM DBA_DB_LINKS
/
```

Here, the script is invoked and the resulting output is shown:

```
SQL>@link_script
```

```
OWNER          DB_LINK                                USER          SERVICE  CREATED
-----
SYS            TARGET.US.ACME.COM                     SYS           inst1    23-JUN-99
PUBLIC        DBL1.UK.ACME.COM                       BLAKE        ora51    23-JUN-99
PUBLIC        RMAN2.US.ACME.COM                      inst2        23-JUN-99
PUBLIC        DEPT.US.ACME.COM                       inst2        23-JUN-99
JANE          DBL.UK.ACME.COM                        BLAKE        ora51    23-JUN-99
SCOTT         EMP.US.ACME.COM                        SCOTT        inst2    23-JUN-99
6 rows selected.
```

Authorization for Viewing Password Information

Only `USER_DB_LINKS` contains a column for password information. However, if you are an administrative user (`SYS` or users who connect `AS SYSDBA`), then you can view passwords for all links in the database by querying the `LINK$` table. If you

are not an administrative user, you can be authorized to query the `LINK$` table by one of the following methods:

- Being granted specific object privilege for the `LINK$` table
- Being granted the `SELECT ANY DICTIONARY` system privilege

See Also: ["Understanding User Privileges and Roles"](#) on page 25-2 for more information about privileges necessary to view objects in the `SYS` schema

Viewing Password Information

You can create and run the following script in SQL*Plus to obtain password information (sample output included):

```
COL USERID FORMAT A10
COL PASSWORD FORMAT A10
SELECT USERID,PASSWORD
       FROM SYS.LINK$
       WHERE PASSWORD IS NOT NULL
/
```

SQL>@linkpwd

```
USERID      PASSWORD
-----
SYS         ORACLE
BLAKE       TYGER
SCOTT       TIGER
3 rows selected.
```

Viewing Authentication Passwords

It is possible to view `AUTHENTICATED BY ... IDENTIFIED BY ...` usernames and passwords for all links in the database by querying the `LINK$` table. You can create and run the following script in SQL*Plus to obtain password information (sample output included):

```
COL AUTHUSR FORMAT A10
COL AUTHPWD FORMAT A10
SELECT AUTHUSR AS userid, AUTHPWD AS password
       FROM SYS.LINK$
       WHERE PASSWORD IS NOT NULL
/
```

```
SQL> @authpwd
```

```

USERID      PASSWORD
-----
ELLIE      MAY
1 row selected.
```

You can also view the link and password information together in a join by creating and executing the following script (sample output included):

```

COL OWNER FORMAT A8
COL DB_LINK FORMAT A15
COL USERNAME FORMAT A8 HEADING "CON_USER"
COL PASSWORD FORMAT A8 HEADING "CON_PWD"
COL AUTHUSR FORMAT A8 HEADING "AUTH_USER"
COL AUTHPWD FORMAT A8 HEADING "AUTH_PWD"
COL HOST FORMAT A7 HEADING "SERVICE"
COL CREATED FORMAT A10

SELECT DISTINCT d.OWNER,d.DB_LINK,d.USERNAME,l.PASSWORD,
               l.AUTHUSR,l.AUTHPWD,d.HOST,d.CREATED
FROM DBA_DB_LINKS d, SYS.LINK$ l
WHERE PASSWORD IS NOT NULL
AND d.USERNAME = l.USERID
/
```

```
SQL> @user_and_pwd
```

OWNER	DB_LINK	CON_USER	CON_PWD	AUTH_USE	AUTH_PWD	SERVICE	CREATED
JANE	DBL.ACME.COM	BLAKE	TYGER	ELLIE	MAY	ora51	23-JUN-99
PUBLIC	DBL1.ACME.COM	SCOTT	TIGER			ora51	23-JUN-99
SYS	TARGET.ACME.COM	SYS	ORACLE			inst1	23-JUN-99

Determining Which Link Connections Are Open

You may find it useful to determine which database link connections are currently open in your session. Note that if you connect as SYSDBA, you cannot query a view to determine all the links open for all sessions; you can only access the link information in the session within which you are working.

The following views show the database link connections that are currently open in your current session:

View	Purpose
V\$DBLINK	Lists all open database links in your session, that is, all database links with the IN_TRANSACTION column set to YES.
GV\$DBLINK	Lists all open database links in your session along with their corresponding instances. This view is useful in an Oracle Real Application Clusters configuration.

These data dictionary views contain the same basic information about database links, with one exception:

Column	Which Views?	Description
DB_LINK	All	The name of the database link.
OWNER_ID	All	The owner of the database link.
LOGGED_ON	All	Whether the database link is currently logged on.
HETEROGENEOUS	All	Whether the database link is homogeneous (NO) or heterogeneous (YES).
PROTOCOL	All	The communication protocol for the database link.
OPEN_CURSORS	All	Whether cursors are open for the database link.
IN_TRANSACTION	All	Whether the database link is accessed in a transaction that has not yet been committed or rolled back.
UPDATE_SENT	All	Whether there was an update on the database link.
COMMIT_POINT_STRENGTH	All	The commit point strength of the transactions using the database link.
INST_ID	GV\$DBLINK only	The instance from which the view information was obtained.

For example, you can create and execute the script below to determine which links are open (sample output included):

```
COL DB_LINK FORMAT A25
COL OWNER_ID FORMAT 99999 HEADING "OWNID"
COL LOGGED_ON FORMAT A5 HEADING "LOGON"
COL HETEROGENEOUS FORMAT A5 HEADING "HETER"
COL PROTOCOL FORMAT A8
COL OPEN_CURSORS FORMAT 999 HEADING "OPN_CUR"
```

```
COL IN_TRANSACTION FORMAT A3 HEADING "TXN"
COL UPDATE_SENT FORMAT A6 HEADING "UPDATE"
COL COMMIT_POINT_STRENGTH FORMAT 99999 HEADING "C_P_S"
```

```
SELECT * FROM V$DBLINK
/
```

```
SQL> @dblink
```

DB_LINK	OWNID	LOGON	HETER	PROTOCOL	OPN_CUR	TXN	UPDATE	C_P_S
INST2.ACME.COM	0	YES	YES	UNKN	0	YES	YES	255

Creating Location Transparency

After you have configured the necessary database links, you can use various tools to hide the distributed nature of the database system from users. In other words, users can access remote objects as if they were local objects. The following sections explain how to hide distributed functionality from users:

- [Using Views to Create Location Transparency](#)
- [Using Synonyms to Create Location Transparency](#)
- [Using Procedures to Create Location Transparency](#)

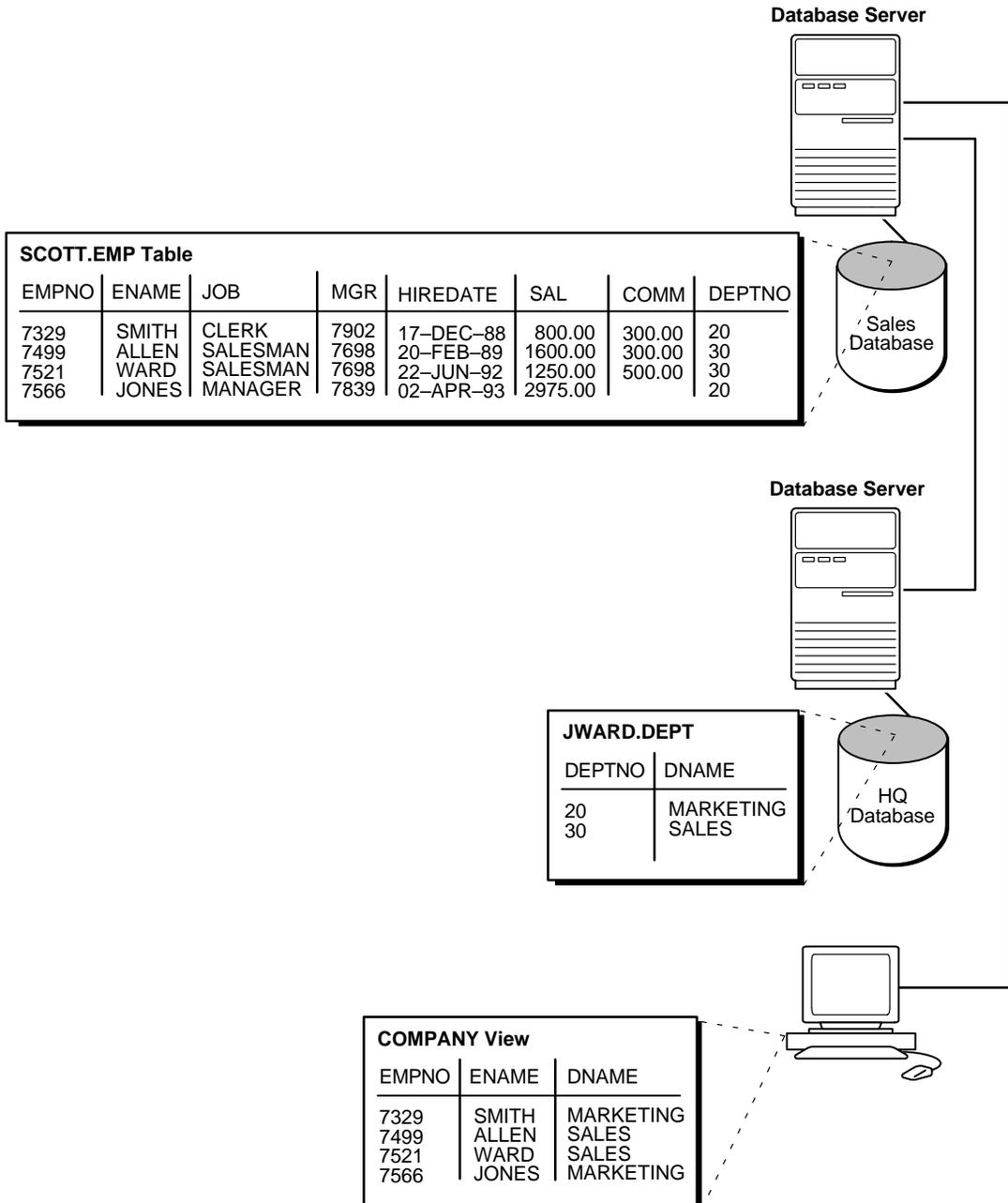
Using Views to Create Location Transparency

Local views can provide location transparency for local and remote tables in a distributed database system.

For example, assume that table `emp` is stored in a local database and table `dept` is stored in a remote database. To make these tables transparent to users of the system, you can create a view in the local database that joins local and remote data:

```
CREATE VIEW company AS
  SELECT a.empno, a.ename, b.dname
  FROM scott.emp a, jward.dept@hq.acme.com b
  WHERE a.deptno = b.deptno;
```

Figure 29-3 Views and Location Transparency



When users access this view, they do not need to know where the data is physically stored, or if data from more than one table is being accessed. Thus, it is easier for them to get required information. For example, the following query provides data from both the local and remote database table:

```
SELECT * FROM company;
```

The owner of the local view can grant only those object privileges on the local view that have been granted by the remote user. (The remote user is implied by the type of database link). This is similar to privilege management for views that reference local data.

Using Synonyms to Create Location Transparency

Synonyms are useful in both distributed and nondistributed environments because they hide the identity of the underlying object, including its location in a distributed database system. If you must rename or move the underlying object, you only need to redefine the synonym; applications based on the synonym continue to function normally. Synonyms also simplify SQL statements for users in a distributed database system.

Creating Synonyms

You can create synonyms for the following:

- Tables
- Types
- Views
- Materialized views
- Sequences
- Procedures
- Functions
- Packages

All synonyms are schema objects that are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym can allow single-word access to remote data, hiding the specific object name and the location from users of the synonym.

The syntax to create a synonym is:

```
CREATE [PUBLIC] synonym_name
FOR [schema.]object_name[@database_link_name];
```

where:

- **PUBLIC** is a keyword specifying that this synonym is available to all users. Omitting this parameter makes a synonym private, and usable only by the creator. Public synonyms can be created only by a user with `CREATE PUBLIC SYNONYM` system privilege.
- *synonym_name* specifies the alternate object name to be referenced by users and applications.
- *schema* specifies the schema of the object specified in *object_name*. Omitting this parameter uses the creator's schema as the schema of the object.
- *object_name* specifies either a table, view, sequence, materialized view, type, procedure, function or package as appropriate.
- *database_link_name* specifies the database link identifying the remote database and schema in which the object specified in *object_name* is located.

A synonym must be a uniquely named object for its schema. If a schema contains a schema object and a public synonym exists with the same name, then Oracle always finds the schema object when the user that owns the schema references that name.

Example: Creating a Public Synonym

Assume that in every database in a distributed database system, a public synonym is defined for the `scott.emp` table stored in the `hq` database:

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.acme.com;
```

You can design an employee management application without regard to where the application is used because the location of the table `scott.emp@hq.acme.com` is hidden by the public synonyms. SQL statements in the application access the table by referencing the public synonym `emp`.

Furthermore, if you move the `emp` table from the `hq` database to the `hr` database, then you only need to change the public synonyms on the nodes of the system. The employee management application continues to function properly on all nodes.

Managing Privileges and Synonyms

A synonym is a reference to an actual object. A user who has access to a synonym for a particular schema object must also have privileges on the underlying schema object itself. For example, if the user attempts to access a synonym but does not

have privileges on the table it identifies, an error occurs indicating that the table or view does not exist.

Assume `scott` creates local synonym `emp` as an alias for remote object `scott.emp@sales.acme.com`. `scott` *cannot* grant object privileges on the synonym to another local user. `scott` cannot grant local privileges for the synonym because this operation amounts to granting privileges for the remote `emp` table on the `sales` database, which is not allowed. This behavior is different from privilege management for synonyms that are aliases for local tables or views.

Therefore, you cannot manage local privileges when synonyms are used for location transparency. Security for the base object is controlled entirely at the remote node. For example, user `admin` cannot grant object privileges for the `EMP_SYN` synonym.

Unlike a database link referenced in a view or procedure definition, a database link referenced in a synonym is resolved by first looking for a private link owned by the schema in effect at the time the reference to the synonym is parsed. Therefore, to ensure the desired object resolution, it is especially important to specify the underlying object's schema in the definition of a synonym.

Using Procedures to Create Location Transparency

PL/SQL program units called **procedures** can provide location transparency. You have these options:

- [Using Local Procedures to Reference Remote Data](#)
- [Using Local Procedures to Call Remote Procedures](#)
- [Using Local Synonyms to Reference Remote Procedures](#)

Using Local Procedures to Reference Remote Data

Procedures or functions (either standalone or in packages) can contain SQL statements that reference remote data. For example, consider the procedure created by the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
    DELETE FROM emp@hq.acme.com
    WHERE empno = enum;
END;
```

When a user or application calls the `fire_emp` procedure, it is not apparent that a remote table is being modified.

A second layer of location transparency is possible when the statements in a procedure indirectly reference remote data using local procedures, views, or synonyms. For example, the following statement defines a local synonym:

```
CREATE SYNONYM emp FOR emp@hq.acme.com;
```

Given this synonym, you can create the `fire_emp` procedure using the following statement:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
    DELETE FROM emp WHERE empno = enum;
END;
```

If you rename or move the table `emp@hq`, then you only need to modify the local synonym that references the table. None of the procedures and applications that call the procedure require modification.

Using Local Procedures to Call Remote Procedures

You can use a local procedure to call a remote procedure. The remote procedure can then execute the required DML. For example, assume that `scott` connects to `local_db` and creates the following procedure:

```
CONNECT scott/tiger@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
    EXECUTE term_emp@hq.acme.com;
END;
```

Now, assume that `scott` connects to the remote database and creates the remote procedure:

```
CONNECT scott/tiger@hq.acme.com

CREATE PROCEDURE term_emp (enum NUMBER)
AS
BEGIN
    DELETE FROM emp WHERE empno = enum;
END;
```

When a user or application connected to `local_db` calls the `fire_emp` procedure, this procedure in turn calls the remote `term_emp` procedure on `hq.acme.com`.

Using Local Synonyms to Reference Remote Procedures

For example, `scott` connects to the local `sales.acme.com` database and creates the following procedure:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.acme.com
WHERE empno = enum;
END;
```

User `peggy` then connects to the `supply.acme.com` database and creates the following synonym for the procedure that `scott` created on the remote `sales` database:

```
SQL> CONNECT peggy/hill@supply
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.acme.com;
```

A local user on `supply` can use this synonym to execute the procedure on `sales`.

Managing Procedures and Privileges

Assume a local procedure includes a statement that references a remote table or view. The owner of the local procedure can grant the `execute` privilege to any user, thereby giving that user the ability to execute the procedure and, indirectly, access remote data.

In general, procedures aid in security. Privileges for objects referenced within a procedure do not need to be explicitly granted to the calling users.

Managing Statement Transparency

Oracle allows the following standard DML statements to reference remote tables:

- `SELECT` (queries)
- `INSERT`
- `UPDATE`
- `DELETE`
- `SELECT ... FOR UPDATE` (not always supported in Heterogeneous Systems)
- `LOCK TABLE`

Queries including joins, aggregates, subqueries, and `SELECT ... FOR UPDATE` can reference any number of local and remote tables and views. For example, the following query joins information from two remote tables:

```
SELECT e.empno, e.ename, d.dname
       FROM scott.emp@sales.division3.acme.com e, jward.dept@hq.acme.com d
       WHERE e.deptno = d.deptno;
```

`UPDATE`, `INSERT`, `DELETE`, and `LOCK TABLE` statements can reference both local and remote tables. No programming is necessary to update remote data. For example, the following statement inserts new rows into the remote table `emp` in the `scott.sales` schema by selecting rows from the `emp` table in the `jward` schema in the local database:

```
INSERT INTO scott.emp@sales.division3.acme.com
          SELECT * FROM jward.emp;
```

Restrictions:

Several restrictions apply to statement transparency.

- Within a single SQL statement, all referenced `LONG` and `LONG RAW` columns, sequences, updated tables, and locked tables must be located at the same node.
- Oracle does not allow remote DDL statements (for example, `CREATE`, `ALTER`, and `DROP`) in homogeneous systems except through remote execution of procedures of the `DBMS_SQL` package, as in this example:

```
DBMS_SQL.PARSE@link_name(crs, 'drop table emp', v7);
```

Note that in Heterogeneous Systems, a pass-through facility allows you to execute DDL.

- The `LIST CHAINED ROWS` clause of an `ANALYZE` statement cannot reference remote tables.
- In a distributed database system, Oracle always evaluates environmentally-dependent SQL functions such as `SYSDATE`, `USER`, `UID`, and `USERENV` with respect to the local server, no matter where the statement (or portion of a statement) executes.

Note: Oracle supports the `USERENV` function for queries only.

- A number of performance restrictions relate to access of remote objects:

- Remote views do not have statistical data.
- Queries on partitioned tables may not be optimized.
- No more than 20 indexes are considered for a remote table.
- No more than 20 columns are used for a composite index.

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

- There is a restriction in Oracle's implementation of distributed read consistency that can cause one node to be in the past with respect to another node. In accordance with read consistency, a query may end up retrieving consistent, but out-of-date data. See "[Managing Read Consistency](#)" on page 32-25 to learn how to manage this problem.

Managing a Distributed Database: Scenarios

This section gives examples of various types of statements involving management of database links:

- [Creating a Public Fixed User Database Link](#)
- [Creating a Public Fixed User Shared Database Link](#)
- [Creating a Public Connected User Database Link](#)
- [Creating a Public Connected User Shared Database Link](#)
- [Creating a Public Current User Database Link](#)

Creating a Public Fixed User Database Link

The following example connects to the local database as `jane` and creates a public fixed user database link to database `sales` for `scott`. The database is accessed through its net service name `sldb`:

```
CONNECT jane/does@local

CREATE PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO scott IDENTIFIED BY tiger
  USING 'sldb';
```

Consequences:

Any user connected to the local database can use the `sales.division3.acme.com` database link to connect to the remote database. Each user connects to the schema `scott` in the remote database.

To access the table `emp` table in `scott`'s remote schema, a user can issue the following SQL query:

```
SELECT * FROM emp@sales.division3.acme.com;
```

Note that each application or user session creates a separate connection to the common account on the server. The connection to the remote database remains open for the duration of the application or user session.

Creating a Public Fixed User Shared Database Link

The following example connects to the local database as `dana` and creates a public link to the `sales` database (using its net service name `sldb`). The link allows a connection to the remote database as `scott` and authenticates this user as `scott`:

```
CONNECT dana/sculley@local
```

```
CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO scott IDENTIFIED BY tiger
  AUTHENTICATED BY scott IDENTIFIED BY tiger
  USING 'sldb';
```

Consequences:

Any user connected to the local shared server can use this database link to connect to the remote `sales` database through a shared server process. The user can then query tables in the `scott` schema.

In the above example, each local shared server can establish one connection to the remote server. Whenever a local shared server process needs to access the remote server through the `sales.division3.acme.com` database link, the local shared server process reuses established network connections.

Creating a Public Connected User Database Link

The following example connects to the local database as `larry` and creates a public link to the database with the net service name `sldb`:

```
CONNECT larry/oracle@local
```

```
CREATE PUBLIC DATABASE LINK redwood
  USING 'sldb';
```

Consequences:

Any user connected to the local database can use the `redwood` database link. The connected user in the local database who uses the database link determines the remote schema.

If `scott` is the connected user and uses the database link, then the database link connects to the remote schema `scott`. If `fox` is the connected user and uses the database link, then the database link connects to remote schema `fox`.

The following statement fails for local user `fox` in the local database when the remote schema `fox` cannot resolve the `emp` schema object. That is, if the `fox` schema in the `sales.division3.acme.com` does not have `emp` as a table, view, or (public) synonym, an error will be returned.

```
CONNECT fox/mulder@local

SELECT * FROM emp@redwood;
```

Creating a Public Connected User Shared Database Link

The following example connects to the local database as `neil` and creates a shared, public link to the `sales` database (using its net service name `sldb`). The user is authenticated by the `userid/password` of `crazy/horse`. The following statement creates a public, connected user, shared database link:

```
CONNECT neil/young@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
  AUTHENTICATED BY crazy IDENTIFIED BY horse
  USING 'sldb';
```

Consequences:

Each user connected to the local server can use this shared database link to connect to the remote database and query the tables in the corresponding remote schema.

Each local, shared server process establishes one connection to the remote server. Whenever a local server process needs to access the remote server through the `sales.division3.acme.com` database link, the local process reuses established network connections, even if the connected user is a different user.

If this database link is used frequently, eventually every shared server in the local database will have a remote connection. At this point, no more physical connections are needed to the remote server, even if new users use this shared database link.

Creating a Public Current User Database Link

The following example connects to the local database as the connected user and creates a public link to the `sales` database (using its net service name `sldb`). The following statement creates a public current user database link:

```
CONNECT bart/simpson@local

CREATE PUBLIC DATABASE LINK sales.division3.acme.com
  CONNECT TO CURRENT_USER
  USING 'sldb';
```

Note: To use this link, the current user must be a global user.

Consequences:

Assume `scott` creates local procedure `fire_emp` that deletes a row from the remote `emp` table, and grants `execute` privilege on `fire_emp` to `ford`.

```
CONNECT scott/tiger@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
  DELETE FROM emp@sales.division3.acme.com
  WHERE empno=enum;
END;

GRANT EXECUTE ON fire_emp TO ford;
```

Now, assume that `ford` connects to the local database and runs `scott`'s procedure:

```
CONNECT ford/fairlane@local_db

EXECUTE PROCEDURE scott.fire_emp (enum 10345);
```

When `ford` executes the procedure `scott.fire_emp`, the procedure runs under `scott`'s privileges. Because a current user database link is used, the connection is established to `scott`'s remote schema—not `ford`'s remote schema. Note that `scott` must be a global user while `ford` does not have to be a global user.

Note: If a connected user database link were used instead, the connection would be to `ford`'s remote schema. For more information about invoker's-rights and privileges, see the *PL/SQL User's Guide and Reference*.

You can accomplish the same result by using a fixed user database link to `scott`'s remote schema. With fixed user database links, however, security can be compromised because `scott`'s username and password are available in readable format in the database.

Developing Applications for a Distributed Database System

This chapter describes considerations important when developing an application to run in a distributed database system. It contains the following topics:

- [Managing the Distribution of an Application's Data](#)
- [Controlling Connections Established by Database Links](#)
- [Maintaining Referential Integrity in a Distributed System](#)
- [Tuning Distributed Queries](#)
- [Handling Errors in Remote Procedures](#)

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for more information about application development in an Oracle environment

Managing the Distribution of an Application's Data

In a distributed database environment, coordinate with the database administrator to determine the best location for the data. Some issues to consider are:

- Number of transactions posted from each location
- Amount of data (portion of table) used by each node
- Performance characteristics and reliability of the network
- Speed of various nodes, capacities of disks
- Importance of a node or link when it is unavailable
- Need for referential integrity among tables

Controlling Connections Established by Database Links

When a global object name is referenced in a SQL statement or remote procedure call, database links establish a connection to a session in the remote database on behalf of the local user. The remote connection and session are only created if the connection has not already been established previously for the local user session.

The connections and sessions established to remote databases persist for the duration of the local user's session, unless the application or user explicitly terminates them. Note that when you issue a `SELECT` statement across a database link, a transaction lock is placed on the rollback segments. To re-release the segment, you must issue a `COMMIT` or `ROLLBACK` statement.

Terminating remote connections established using database links is useful for disconnecting high cost connections that are no longer required by the application. You can terminate a remote connection and session using the `ALTER SESSION` statement with the `CLOSE DATABASE LINK` clause. For example, assume you issue the following transactions:

```
SELECT * FROM emp@sales;  
COMMIT;
```

The following statement terminates the session in the remote database pointed to by the `sales` database link:

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

To close a database link connection in your user session, you must have the `ALTER SESSION` system privilege.

Note: Before closing a database link, first close all cursors that use the link and then end your current transaction if it uses the link.

See Also: *Oracle9i SQL Reference* for more information about the `ALTER SESSION` statement.

Maintaining Referential Integrity in a Distributed System

If a part of a distributed statement fails, for example, due to an integrity constraint violation, Oracle returns error number `ORA-02055`. Subsequent statements or procedure calls return error number `ORA-02067` until a rollback or rollback to savepoint is issued.

Design your application to check for any returned error messages that indicate that a portion of the distributed update has failed. If you detect a failure, you should roll back the entire transaction before allowing the application to proceed.

Oracle does not permit declarative referential integrity constraints to be defined across nodes of a distributed system. In other words, a declarative referential integrity constraint on one table cannot specify a foreign key that references a primary or unique key of a remote table. Nevertheless, you can maintain parent/child table relationships across nodes using triggers.

If you decide to define referential integrity across the nodes of a distributed database using triggers, be aware that network failures can limit the accessibility of not only the parent table, but also the child table. For example, assume that the child table is in the `sales` database and the parent table is in the `hq` database. If the network connection between the two databases fails, some DML statements against the child table (those that insert rows into the child table or update a foreign key value in the child table) cannot proceed because the referential integrity triggers must have access to the parent table in the `hq` database.

See Also: *Oracle9i Application Developer's Guide - Fundamentals* for more information about using triggers to enforce referential integrity.

Tuning Distributed Queries

The local Oracle database server breaks the distributed query into a corresponding number of remote queries, which it then sends to the remote nodes for execution.

The remote nodes execute the queries and send the results back to the local node. The local node then performs any necessary post-processing and returns the results to the user or application.

You have several options for designing your application to optimize query processing. This section contains the following topics:

- [Using Collocated Inline Views](#)
- [Using Cost-Based Optimization](#)
- [Using Hints](#)
- [Analyzing the Execution Plan](#)

Using Collocated Inline Views

The most effective way of optimizing distributed queries is to access the remote databases as little as possible and to retrieve only the required data.

For example, assume you reference five remote tables from two different remote databases in a distributed query and have a complex filter (for example, `WHERE r1.salary + r2.salary > 50000`). You can improve the performance of the query by rewriting the query to access the remote databases once and to apply the filter at the remote site. This rewrite causes less data to be transferred to the query execution site.

Rewriting your query to access the remote database once is achieved by using collocated inline views. The following terms need to be defined:

Term	Definition
Collocated	Two or more tables located in the same database.
Inline view	A <code>SELECT</code> statement that is substituted for a table in a parent <code>SELECT</code> statement. The embedded <code>SELECT</code> statement, shown within the parentheses is an example of an inline view: <pre>SELECT e.empno, e.ename, d.deptno, d.dname FROM (SELECT empno, ename from emp@orc1.world) e, dept d;</pre>
Collocated inline view	An inline view that selects data from multiple tables from a single database only. It reduces the amount of times that the remote database is accessed, improving the performance of a distributed query.

Oracle Corporation recommends that you form your distributed query using collocated inline views to increase the performance of your distributed query. Oracle's cost-based optimization can transparently rewrite many of your distributed queries to take advantage of the performance gains offered by collocated inline views.

Using Cost-Based Optimization

In addition to rewriting your queries with collocated inline views, the cost-based optimization method optimizes distributed queries according to the gathered statistics of the referenced tables and the computations performed by the optimizer.

For example, cost-based optimization analyzes the following query. The example assumes that table statistics are available. Note that it analyzes the query inside a `CREATE TABLE` statement:

```
CREATE TABLE AS (
  SELECT l.a, l.b, r1.c, r1.d, r1.e, r2.b, r2.c
  FROM local l, remotel r1, remote2 r2
  WHERE l.c = r.c
  AND r1.c = r2.c
  AND r.e > 300
);
```

and rewrites it as:

```
CREATE TABLE AS (
  SELECT l.a, l.b, v.c, v.d, v.e
  FROM (
    SELECT r1.c, r1.d, r1.e, r2.b, r2.c
    FROM remotel r1, remote2 r2
    WHERE r1.c = r2.c
    AND r1.e > 300
  ) v, local l
  WHERE l.c = r1.c
);
```

The alias `v` is assigned to the inline view, which can then be referenced as a table in the above `SELECT` statement. Creating a collocated inline view reduces the amount of queries performed at a remote site, thereby reducing costly network traffic.

How Does Cost-Based Optimization Work?

The optimizer's main task is to rewrite a distributed query to use collocated inline views. This optimization is performed in three steps:

1. All mergeable views are merged.
2. Optimizer performs collocated query block test.
3. Optimizer rewrites query using collocated inline views.

After the query is rewritten, it is executed and the data set is returned to the user.

While cost-based optimization is performed transparently to the user, it is unable to improve the performance of several distributed query scenarios. Specifically, if your distributed query contains any of the following, cost-based optimization is not effective:

- Aggregates
- Subqueries
- Complex SQL

If your distributed query contains one of the above, see ["Using Hints"](#) on page 30-8 to learn how you can modify your query and use hints to improve the performance of your distributed query.

Setting Up Cost-Based Optimization

After you have set up your system to use cost-based optimization to improve the performance of distributed queries, the operation is transparent to the user. In other words, the optimization occurs automatically when the query is issued.

You need to complete the following tasks to set up your system to take advantage of Oracle's optimizer:

- [Setting Up the Environment](#)
- [Analyzing Tables](#)

Setting Up the Environment To enable cost-based optimization, set the `OPTIMIZER_MODE` initialization parameter to `CHOOSE` or `COST`. You can set this parameter by:

- Modifying the `OPTIMIZER_MODE` parameter in the initialization parameter file
- Setting it at session level by issuing an `ALTER SESSION` statement

Issue one of the following statements to set the `OPTIMIZER_MODE` initialization parameter at the session level:

```
ALTER SESSION OPTIMIZER_MODE = CHOOSE;  
ALTER SESSION OPTIMIZER_MODE = COST;
```

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for information on setting the `OPTIMIZER_MODE` initialization parameter in the parameter file and for configuring your system to use a cost-based optimization method.

Analyzing Tables For cost-based optimization to select the most efficient path for a distributed query, you must provide accurate statistics for the tables involved. You do this using the `DBMS_STATS` package or the `ANALYZE` statement.

Note: You must connect locally with respect to the tables when executing the `DBMS_STATS` procedure or `ANALYZE` statement. For example, you cannot execute the following:

```
ANALYZE TABLE remote@remote.com COMPUTE STATISTICS;
```

You must first connect to the remote site and then execute the above `ANALYZE` statement, or the equivalent `DBMS_STATS` procedure.

The following `DBMS_STATS` procedures enable the gathering of certain classes of optimizer statistics:

- `GATHER_INDEX_STATS`
- `GATHER_TABLE_STATS`
- `GATHER_SCHEMA_STATS`
- `GATHER_DATABASE_STATS`

For example, assume that distributed transactions routinely access the `scott.dept` table. To ensure that the cost-based optimizer is still picking the best plan, execute the following:

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS ('scott', 'dept');
END;
```

See Also:

- *Oracle9i Database Performance Tuning Guide and Reference* for information about generating statistics
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for additional information on using the `DBMS_STATS` package

Using Hints

If a statement is not sufficiently optimized, then you can use hints to extend the capability of cost-based optimization. Specifically, if you write your own query to utilize collocated inline views, instruct the cost-based optimizer not to rewrite your distributed query.

Additionally, if you have special knowledge about the database environment (such as statistics, load, network and CPU limitations, distributed queries, and so forth), you can specify a hint to guide cost-based optimization. For example, if you have written your own optimized query using collocated inline views that are based on your knowledge of the database environment, specify the `NO_MERGE` hint to prevent the optimizer from rewriting your query.

This technique is especially helpful if your distributed query contains an aggregate, subquery, or complex SQL. Because this type of distributed query cannot be rewritten by the optimizer, specifying `NO_MERGE` causes the optimizer to skip the steps described in "[How Does Cost-Based Optimization Work?](#)" on page 30-5.

The `DRIVING_SITE` hint allows you to define a remote site to act as the query execution site. In this way, the query executes on the remote site, which then returns the data to the local site. This hint is especially helpful when the remote site contains the majority of the data.

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for more information about using hints.

Using the `NO_MERGE` Hint

The `NO_MERGE` hint prevents Oracle from merging an inline view into a potentially non-collocated SQL statement (see "[Using Hints](#)" on page 30-8). This hint is embedded in the `SELECT` statement and can appear either at the beginning of the `SELECT` statement with the inline view as an argument or in the query block that defines the inline view.

```
/* with argument */

SELECT /*+NO_MERGE(v)*/ t1.x, v.avg_y
  FROM t1, (SELECT x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
  WHERE t1.x = v.x AND t1.y = 1;

/* in query block */

SELECT t1.x, v.avg_y
  FROM t1, (SELECT /*+NO_MERGE*/ x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
```

```
WHERE t1.x = v.x AND t1.y = 1;
```

Typically, you use this hint when you have developed an optimized query based on your knowledge of your database environment.

Using the DRIVING_SITE Hint

The `DRIVING_SITE` hint allows you to specify the site where the query execution is performed. It is best to let cost-based optimization determine where the execution should be performed, but if you prefer to override the optimizer, you can specify the execution site manually.

Following is an example of a `SELECT` statement with a `DRIVING_SITE` hint:

```
SELECT /*+DRIVING_SITE(dept)*/ * FROM emp, dept@remote.com
WHERE emp.deptno = dept.deptno;
```

Analyzing the Execution Plan

An important aspect to tuning distributed queries is analyzing the execution plan. The feedback that you receive from your analysis is an important element to testing and verifying your database. Verification becomes especially important when you want to compare plans. For example, comparing the execution plan for a distributed query optimized by cost-based optimization to a plan for a query manually optimized using hints, collocated inline views, and other techniques.

See Also: *Oracle9i Database Performance Tuning Guide and Reference* for detailed information about execution plans, the `EXPLAIN PLAN` statement, and how to interpret the results.

Preparing the Database to Store the Plan

Before you can view the execution plan for the distributed query, prepare the database to store the execution plan. You can perform this preparation by executing a script. Execute the following script to prepare your database to store an execution plan:

```
SQL> @UTLXPLAN.SQL
```

Note: The `utlxplan.sql` file can be found in the `$ORACLE_HOME/rdbms/admin` directory.

After you execute `utlxplan.sql`, a table, `PLAN_TABLE`, is created in the current schema to temporarily store the execution plan.

Generating the Execution Plan

After you have prepared the database to store the execution plan, you are ready to view the plan for a specified query. Instead of directly executing a SQL statement, append the statement to the `EXPLAIN PLAN FOR` clause. For example, you can execute the following:

```
EXPLAIN PLAN FOR
  SELECT d.dname
     FROM dept d
        WHERE d.deptno
           IN (SELECT deptno
              FROM emp@orc2.world
             GROUP BY deptno
             HAVING COUNT (deptno) >3
            )
/
```

Viewing the Execution Plan

After you have executed the above SQL statement, the execution plan is stored temporarily in the `PLAN_TABLE` that you created earlier. To view the results of the execution plan, execute the following script:

```
@UTLXPPL.SQL
```

Note: The `utlxpls.sql` can be found in the `$ORACLE_HOME/rdbms/admin` directory.

Executing the `utlxpls.sql` script displays the execution plan for the `SELECT` statement that you specified. The results are formatted as follows:

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT						
NESTED LOOPS						
VIEW						
REMOTE						
TABLE ACCESS BY INDEX RO	DEPT					

INDEX UNIQUE SCAN	PK_DEPT					
-------------------	---------	--	--	--	--	--

If you are manually optimizing distributed queries by writing your own collocated inline views or using hints, it is best to generate an execution plan before and after your manual optimization. With both execution plans, you can compare the effectiveness of your manual optimization and make changes as necessary to improve the performance of the distributed query.

To view the SQL statement that will be executed at the remote site, execute the following select statement:

```
SELECT OTHER
FROM PLAN_TABLE
WHERE operation = 'REMOTE';
```

Following is sample output:

```
SELECT DISTINCT "A1"."DEPTNO" FROM "EMP" "A1"
GROUP BY "A1"."DEPTNO" HAVING COUNT("A1"."DEPTNO")>3
```

Note: If you are having difficulty viewing the entire contents of the OTHER column, execute the following SQL*Plus command:

```
SET LONG 9999999
```

Handling Errors in Remote Procedures

When Oracle executes a procedure locally or at a remote location, four types of exceptions can occur:

- PL/SQL user-defined exceptions, which must be declared using the keyword `EXCEPTION`
- PL/SQL predefined exceptions such as the `NO_DATA_FOUND` keyword
- SQL errors such as `ORA-00900` and `ORA-02015`
- Application exceptions generated using the `RAISE_APPLICATION_ERROR()` procedure

When using local procedures, you can trap these messages by writing an exception handler such as the following:

```
BEGIN
...
```

```
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    /* ... handle the exception */
END;
```

Notice that the `WHEN` clause requires an exception name. If the exception does not have a name, for example, exceptions generated with `RAISE_APPLICATION_ERROR`, you can assign one using `PRAGMA_EXCEPTION_INIT`. For example:

```
DECLARE
  null_salary EXCEPTION;
  PRAGMA EXCEPTION_INIT(null_salary, -20101);
BEGIN
  ...
  RAISE_APPLICATION_ERROR(-20101, 'salary is missing');
  ...
EXCEPTION
  WHEN null_salary THEN
    ...
END;
```

When calling a remote procedure, exceptions can be handled by an exception handler in the local procedure. The remote procedure must return an error number to the local, calling procedure, which then handles the exception as shown in the previous example. Note that PL/SQL user-defined exceptions always return `ORA-06510` to the local procedure.

Therefore, it is not possible to distinguish between two different user-defined exceptions based on the error number. All other remote exceptions can be handled in the same manner as local exceptions.

See Also: *PL/SQL User's Guide and Reference* for more information about PL/SQL procedures

Distributed Transactions Concepts

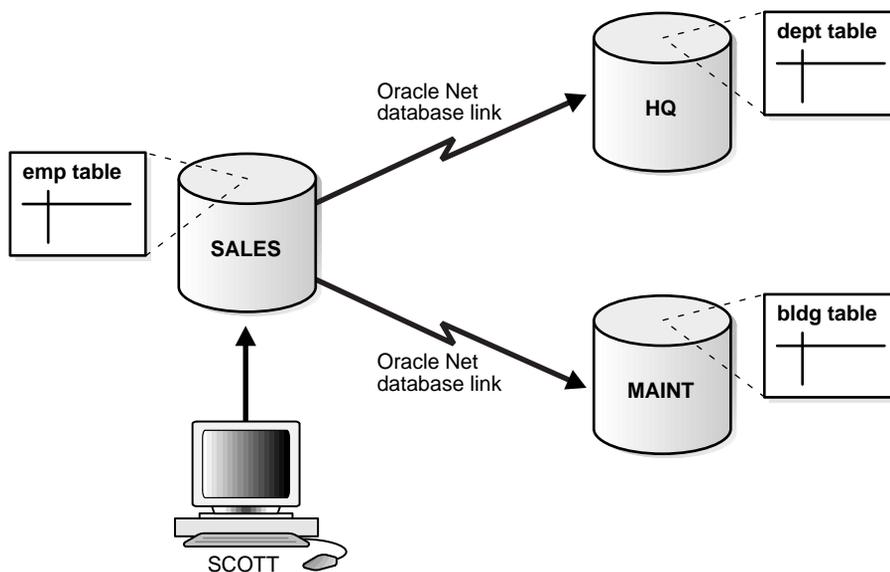
This chapter describes what distributed transactions are and how Oracle maintains their integrity. The following topics are contained in this chapter:

- [What Are Distributed Transactions?](#)
- [Session Trees for Distributed Transactions](#)
- [Two-Phase Commit Mechanism](#)
- [In-Doubt Transactions](#)
- [Distributed Transaction Processing: Case Study](#)

What Are Distributed Transactions?

A **distributed transaction** includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, assume the database configuration depicted in [Figure 31-1](#):

Figure 31-1 *Distributed System*



The following distributed transaction executed by `scott` updates the local `sales` database, the remote `hq` database, and the remote `maint` database:

```
UPDATE scott.dept@hq.us.acme.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.acme.com
  SET room = 1225
  WHERE room = 1163;
COMMIT;
```

Note: If all statements of a transaction reference only a single remote node, then the transaction is remote, not distributed.

There are two types of permissible operations in distributed transactions:

- [DML and DDL Transactions](#)
- [Transaction Control Statements](#)

DML and DDL Transactions

The following list describes DML and DDL operations supported in a distributed transaction:

- CREATE TABLE AS SELECT
- DELETE
- INSERT (default and direct load)
- LOCK TABLE
- SELECT
- SELECT FOR UPDATE

You can execute DML and DDL statements in parallel, and INSERT direct load statements serially, but note the following restrictions:

- All remote operations must be SELECT statements.
- These statements must not be clauses in another distributed transaction.
- If the table referenced in the *table_expression_clause* of an INSERT, UPDATE, or DELETE statement is remote, then execution is serial rather than parallel.
- You cannot perform remote operations after issuing parallel DML/DDL or direct load INSERT.
- If the transaction begins using XA or OCI, it executes serially.
- No loopback operations can be performed on the transaction originating the parallel operation. For example, you cannot reference a remote object that is actually a synonym for a local object.
- If you perform a distributed operation other than a SELECT in the transaction, no DML is parallelized.

Transaction Control Statements

The following list describes supported transaction control statements:

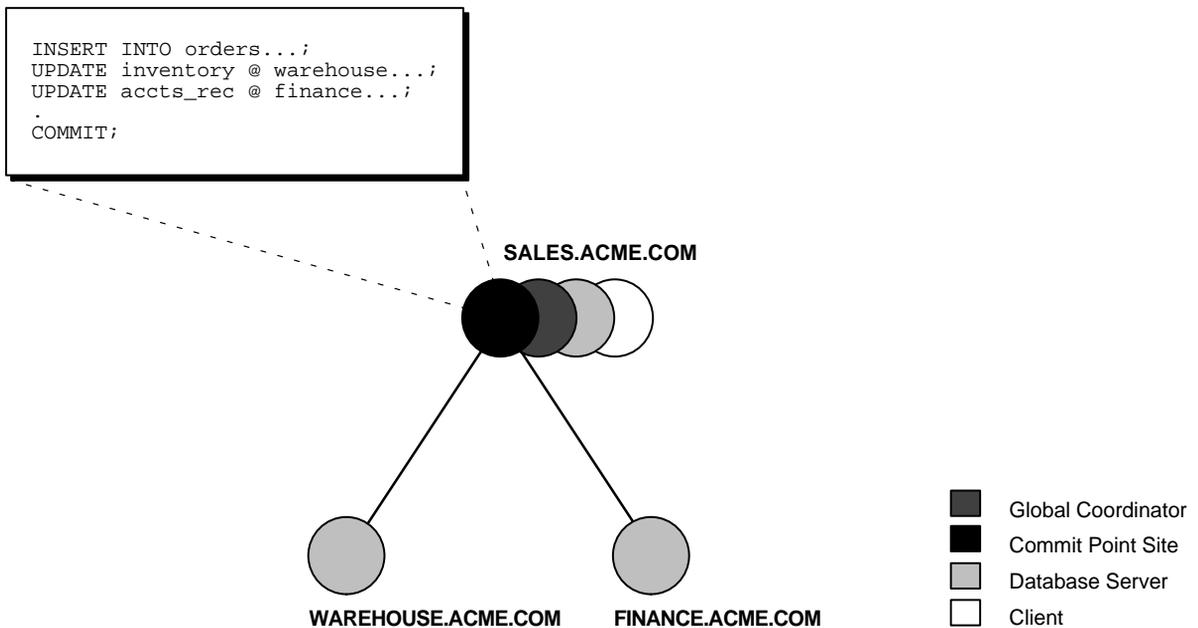
- COMMIT
- ROLLBACK
- SAVEPOINT

See Also: *Oracle9i SQL Reference* for more information about these SQL statements

Session Trees for Distributed Transactions

As the statements in a distributed transaction are issued, Oracle defines a **session tree** of all nodes participating in the transaction. A session tree is a hierarchical model that describes the relationships among sessions and their roles. [Figure 31-2](#) illustrates a session tree:

Figure 31-2 Example of a Session Tree



All nodes participating in the session tree of a distributed transaction assume one or more of the following roles:

Role	Description
Client	A node that references information in a database belonging to a different node.
Database server	A node that receives a request for information from another node.
Global coordinator	The node that originates the distributed transaction.
Local coordinator	A node that is forced to reference data on other nodes to complete its part of the transaction.
Commit point site	The node that commits or rolls back the transaction as instructed by the global coordinator.

The role a node plays in a distributed transaction is determined by:

- Whether the transaction is local or remote
- The **commit point strength** of the node ("[Commit Point Site](#)" on page 31-6)
- Whether all requested data is available at a node, or whether other nodes need to be referenced to complete the transaction
- Whether the node is read-only

Clients

A node acts as a client when it references information from another node's database. The referenced node is a database server. In [Figure 31-2](#), the node `sales` is a client of the nodes that host the `warehouse` and `finance` databases.

Database Servers

A database server is a node that hosts a database from which a client requests data.

In [Figure 31-2](#), an application at the `sales` node initiates a distributed transaction that accesses data from the `warehouse` and `finance` nodes. Therefore, `sales.acme.com` has the role of client node, and `warehouse` and `finance` are both database servers. In this example, `sales` is a database server *and* a client because the application also modifies data in the `sales` database.

Local Coordinators

A node that must reference data on other nodes to complete its part in the distributed transaction is called a local coordinator. In [Figure 31-2](#), `sales` is a local coordinator because it coordinates the nodes it directly references: `warehouse` and `finance`. The node `sales` also happens to be the global coordinator because it coordinates all the nodes involved in the transaction.

A local coordinator is responsible for coordinating the transaction among the nodes it communicates directly with by:

- Receiving and relaying transaction status information to and from those nodes
- Passing queries to those nodes
- Receiving queries from those nodes and passing them on to other nodes
- Returning the results of queries to the nodes that initiated them

Global Coordinator

The node where the distributed transaction originates is called the global coordinator. The database application issuing the distributed transaction is directly connected to the node acting as the global coordinator. For example, in [Figure 31-2](#), the transaction issued at the node `sales` references information from the database servers `warehouse` and `finance`. Therefore, `sales.acme.com` is the global coordinator of this distributed transaction.

The global coordinator becomes the parent or root of the session tree. The global coordinator performs the following operations during a distributed transaction:

- Sends all of the distributed transaction's SQL statements, remote procedure calls, and so forth to the directly referenced nodes, thus forming the session tree
- Instructs all directly referenced nodes other than the commit point site to prepare the transaction
- Instructs the commit point site to initiate the global commit of the transaction if all nodes prepare successfully
- Instructs all nodes to initiate a global rollback of the transaction if there is an abort response

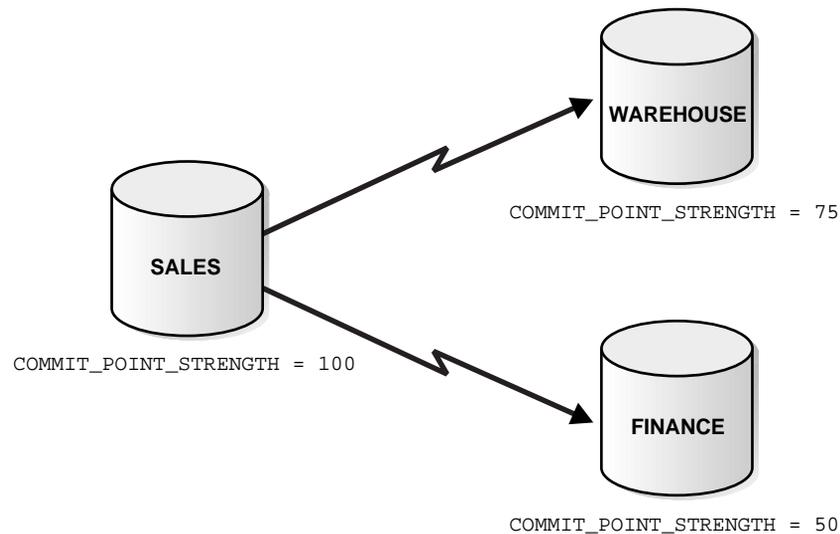
Commit Point Site

The job of the commit point site is to initiate a commit or roll back operation as instructed by the global coordinator. The system administrator always designates

one node to be the commit point site in the session tree by assigning all nodes a commit point strength. The node selected as commit point site should be the node that stores the most critical data.

Figure 31-3 illustrates an example of distributed system, with `sales` serving as the commit point site:

Figure 31-3 Commit Point Site



The commit point site is distinct from all other nodes involved in a distributed transaction in these ways:

- The commit point site never enters the prepared state. Consequently, if the commit point site stores the most critical data, this data never remains in-doubt, even if a failure occurs. In failure situations, failed nodes remain in a prepared state, holding necessary locks on data until in-doubt transactions are resolved.
- The commit point site commits before the other nodes involved in the transaction. In effect, the outcome of a distributed transaction at the commit point site determines whether the transaction at all nodes is committed or rolled back: the other nodes follow the lead of the commit point site. The global coordinator ensures that all nodes complete the transaction in the same manner as the commit point site.

How a Distributed Transaction Commits

A distributed transaction is considered committed after all non-commit point sites are prepared, and the transaction has been actually committed at the commit point site. The online redo log at the commit point site is updated as soon as the distributed transaction is committed at this node.

Because the commit point log contains a record of the commit, the transaction is considered committed even though some participating nodes may still be only in the prepared state and the transaction not yet actually committed at these nodes. In the same way, a distributed transaction is considered *not* committed if the commit has not been logged at the commit point site.

Commit Point Strength

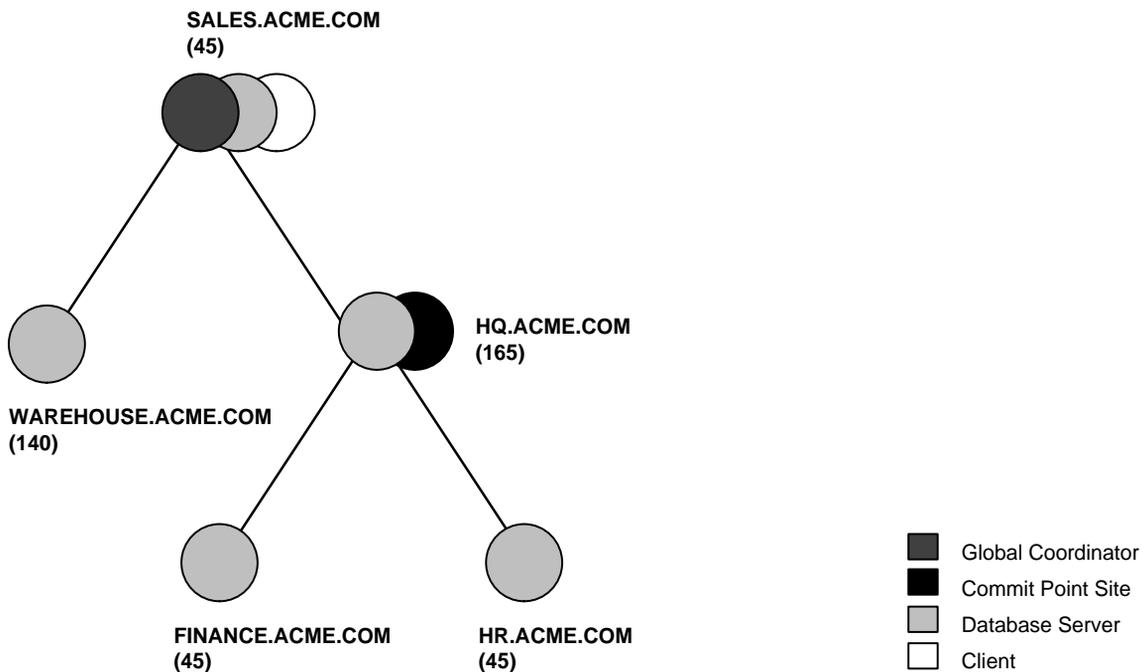
Every database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit. Specifically, the commit point strength determines whether a given node is the commit point site in the distributed transaction and thus commits before all of the other nodes. This value is specified using the initialization parameter `COMMIT_POINT_STRENGTH`. This section explains how Oracle determines the commit point site.

The commit point site, which is determined at the beginning of the prepare phase, is selected only from the nodes participating in the transaction. The following sequence of events occurs:

1. Of the nodes directly referenced by the global coordinator, Oracle selects the node with the highest commit point strength as the commit point site.
2. The initially-selected node determines if any of the nodes from which it has to obtain information for this transaction has a higher commit point strength.
3. Either the node with the highest commit point strength directly referenced in the transaction or one of its servers with a higher commit point strength becomes the commit point site.
4. After the final commit point site has been determined, the global coordinator sends prepare responses to all nodes participating in the transaction.

Figure 31–4 shows in a sample session tree the commit point strengths of each node (in parentheses) and shows the node chosen as the commit point site:

Figure 31–4 Commit Point Strengths and Determination of the Commit Point Site



The following conditions apply when determining the commit point site:

- A read-only node cannot be the commit point site.
- If multiple nodes directly referenced by the global coordinator have the same commit point strength, then Oracle designates one of these as the commit point site.
- If a distributed transaction ends with a rollback, then the prepare and commit phases are not needed. Consequently, Oracle never determines a commit point site. Instead, the global coordinator sends a `ROLLBACK` statement to all nodes and ends the processing of the distributed transaction.

As [Figure 31–4](#) illustrates, the commit point site and the global coordinator can be different nodes of the session tree. The commit point strength of each node is communicated to the coordinators when the initial connections are made. The coordinators retain the commit point strengths of each node they are in direct communication with so that commit point sites can be efficiently selected during two-phase commits. Therefore, it is not necessary for the commit point strength to be exchanged between a coordinator and a node each time a commit occurs.

See Also:

- ["Specifying the Commit Point Strength of a Node"](#) on page 32-2 to learn how to set the commit point strength of a node
- *Oracle9i Database Reference* for more information about the initialization parameter `COMMIT_POINT_STRENGTH`

Two-Phase Commit Mechanism

Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because Oracle must coordinate the committing or rolling back of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

Oracle ensures the integrity of data in a distributed transaction using the **two-phase commit mechanism**. In the **prepare phase**, the initiating node in the transaction asks the other participating nodes to promise to commit or roll back the transaction. During the **commit phase**, the initiating node asks all participating nodes to commit the transaction. If this outcome is not possible, then all nodes are asked to roll back.

All participating nodes in a distributed transaction should perform the same action: they should either all commit or all perform a rollback of the transaction. Oracle automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the **global database** (the collection of databases participating in the transaction) using the two-phase commit mechanism. This mechanism is completely transparent, requiring no programming on the part of the user or application developer.

The commit mechanism has the following distinct phases, which Oracle performs automatically whenever a user commits a distributed transaction:

Phase	Description
Prepare phase	The initiating node, called the global coordinator , asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back.
Commit phase	If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction.
Forget phase	The global coordinator forgets about the transaction.

This section contains the following topics:

- [Prepare Phase](#)
- [Commit Phase](#)
- [Forget Phase](#)

Prepare Phase

The first phase in committing a distributed transaction is the prepare phase. In this phase, Oracle does not actually commit or roll back the transaction. Instead, all nodes referenced in a distributed transaction (except the commit point site, described in the "[Commit Point Site](#)" on page 31-6) are told to prepare to commit. By preparing, a node:

- Records information in the online redo logs so that it can subsequently either commit or roll back the transaction, regardless of intervening failures
- Places a distributed lock on modified tables, which prevents reads

When a node responds to the global coordinator that it is prepared to commit, the prepared node *promises* to either commit or roll back the transaction later—but does not make a unilateral decision on whether to commit or roll back the transaction. The promise means that if an instance failure occurs at this point, the node can use the redo records in the online log to recover the database back to the prepare phase.

Note: Queries that start after a node has prepared cannot access the associated locked data until all phases complete. The time is insignificant unless a failure occurs (see "[Deciding How to Handle In-Doubt Transactions](#)" on page 32-7).

Types of Responses in the Prepare Phase

When a node is told to prepare, it can respond in the following ways:

Response	Meaning
Prepared	Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared.
Read-only	No data on the node has been, or can be, modified (only queried), so no preparation is necessary.
Abort	The node cannot successfully prepare.

Prepared Response When a node has successfully prepared, it issues a **prepared message**. The message indicates that the node has records of the changes in the online log, so it is prepared either to commit or perform a rollback. The message also guarantees that locks held for the transaction can survive a failure.

Read-Only Response When a node is asked to prepare, and the SQL statements affecting the database do not change the node's data, the node responds with a **read-only message**. The message indicates that the node will not participate in the commit phase.

There are three cases in which all or part of a distributed transaction is read-only:

Case	Conditions	Consequence
Partially read-only	Any of the following occurs: <ul style="list-style-type: none"> ▪ Only queries are issued at one or more nodes. ▪ No data is changed. ▪ Changes rolled back due to triggers firing or constraint violations. 	The read-only nodes recognize their status when asked to prepare. They give their local coordinators a read-only response. Thus, the commit phase completes faster because Oracle eliminates read-only nodes from subsequent processing.
Completely read-only with prepare phase	All of following occur: <ul style="list-style-type: none"> ▪ No data changes. ▪ Transaction is <i>not</i> started with SET TRANSACTION READ ONLY statement. 	All nodes recognize that they are read-only during prepare phase, so no commit phase is required. The global coordinator, not knowing whether all nodes are read-only, must still perform the prepare phase.
Completely read-only without two-phase commit	All of following occur: <ul style="list-style-type: none"> ▪ No data changes. ▪ Transaction is started with SET TRANSACTION READ ONLY statement. 	Only queries are allowed in the transaction, so global coordinator does not have to perform two-phase commit. Changes by other transactions do not degrade global transaction-level read consistency because of global SCN coordination among nodes. The transaction does not use rollback segments.

Note that if a distributed transaction is set to read-only, then it does not use rollback segments. If many users connect to the database and their transactions are *not* set to READ ONLY, then they allocate rollback space even if they are only performing queries.

Abort Response When a node cannot successfully prepare, it performs the following actions:

1. Releases resources currently held by the transaction and rolls back the local portion of the transaction.
2. Responds to the node that referenced it in the distributed transaction with an abort message.

These actions then propagate to the other nodes involved in the distributed transaction so that they can roll back the transaction and guarantee the integrity of the data in the global database. This response enforces the primary rule of a distributed transaction: *all nodes involved in the transaction either all commit or all roll back the transaction at the same logical time.*

Steps in the Prepare Phase

To complete the prepare phase, each node excluding the commit point site performs the following steps:

1. The node requests that its **descendants**, that is, the nodes subsequently referenced, prepare to commit.
2. The node checks to see whether the transaction changes data on itself or its descendants. If there is no change to the data, then the node skips the remaining steps and returns a read-only response (see "[Read-Only Response](#)" on page 31-12).
3. The node allocates the resources it needs to commit the transaction if data is changed.
4. The node saves redo records corresponding to changes made by the transaction to its online redo log.
5. The node guarantees that locks held for the transaction are able to survive a failure.
6. The node responds to the initiating node with a prepared response (see "[Prepared Response](#)" on page 31-12) or, if its attempt or the attempt of one of its descendents to prepare was unsuccessful, with an abort response (see "[Abort Response](#)" on page 31-13).

These actions guarantee that the node can subsequently commit or roll back the transaction on the node. The prepared nodes then wait until a COMMIT or ROLLBACK request is received from the global coordinator.

After the nodes are prepared, the distributed transaction is said to be **in-doubt** (see ["In-Doubt Transactions"](#) on page 31-15). It retains in-doubt status until all changes are either committed or rolled back.

Commit Phase

The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, *all* nodes other than the commit point site referenced in the distributed transaction have guaranteed that they are prepared, that is, they have the necessary resources to commit the transaction.

Steps in the Commit Phase

The commit phase consists of the following steps:

1. The global coordinator instructs the commit point site to commit.
2. The commit point site commits.
3. The commit point site informs the global coordinator that it has committed.
4. The global and local coordinators send a message to all nodes instructing them to commit the transaction.
5. At each node, Oracle commits the local portion of the distributed transaction and releases locks.
6. At each node, Oracle records an additional redo entry in the local redo log, indicating that the transaction has committed.
7. The participating nodes notify the global coordinator that they have committed.

When the commit phase is complete, the data on all nodes of the distributed system is consistent.

Guaranteeing Global Database Consistency

Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction. The SCN functions as an internal Oracle timestamp that uniquely identifies a committed version of the database.

In a distributed system, the SCNs of communicating nodes are coordinated when all of the following actions occur:

- A connection occurs using the path described by one or more database links
- A distributed SQL statement executes

- A distributed transaction commits

Among other benefits, the coordination of SCNs among the nodes of a distributed system ensures global read-consistency at both the statement and transaction level. If necessary, global time-based recovery can also be completed.

During the prepare phase, Oracle determines the highest SCN at all nodes involved in the transaction. The transaction then commits with the high SCN at the commit point site. The commit SCN is then sent to all prepared nodes with the commit decision.

See Also: ["Managing Read Consistency"](#) on page 32-25 for information about managing time lag issues in read consistency

Forget Phase

After the participating nodes notify the commit point site that they have committed, the commit point site can forget about the transaction. The following steps occur:

1. After receiving notice from the global coordinator that all nodes have committed, the commit point site erases status information about this transaction.
2. The commit point site informs the global coordinator that it has erased the status information.
3. The global coordinator erases its own information about the transaction.

In-Doubt Transactions

The two-phase commit mechanism ensures that all nodes either commit or perform a rollback together. What happens if any of the three phases fails because of a system or network error? The transaction becomes in-doubt.

Distributed transactions can become in-doubt in the following ways:

- A server machine running Oracle software crashes
- A network connection between two or more Oracle databases involved in distributed processing is disconnected
- An unhandled software error occurs

The RECO process automatically resolves in-doubt transactions when the machine, network, or software problem is resolved. Until RECO can resolve the transaction,

the data is locked for both reads and writes. Oracle blocks reads because it cannot determine which version of the data to display for a query.

This section contains the following topics:

- [Automatic Resolution of In-Doubt Transactions](#)
- [Manual Resolution of In-Doubt Transactions](#)
- [Relevance of System Change Numbers for In-Doubt Transactions](#)

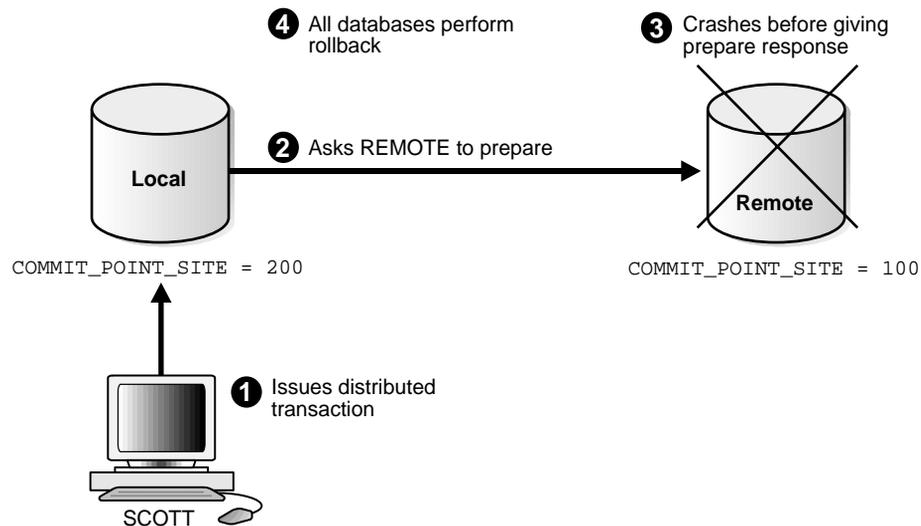
Automatic Resolution of In-Doubt Transactions

In the majority of cases, Oracle resolves the in-doubt transaction automatically. Assume that there are two nodes, `local` and `remote`, in the following scenarios. The local node is the commit point site. User `scott` connects to `local` and executes and commits a distributed transaction that updates `local` and `remote`.

Failure During the Prepare Phase

[Figure 31-5](#) illustrates the sequence of events when there is a failure during the prepare phase of a distributed transaction:

Figure 31-5 *Failure During Prepare Phase*



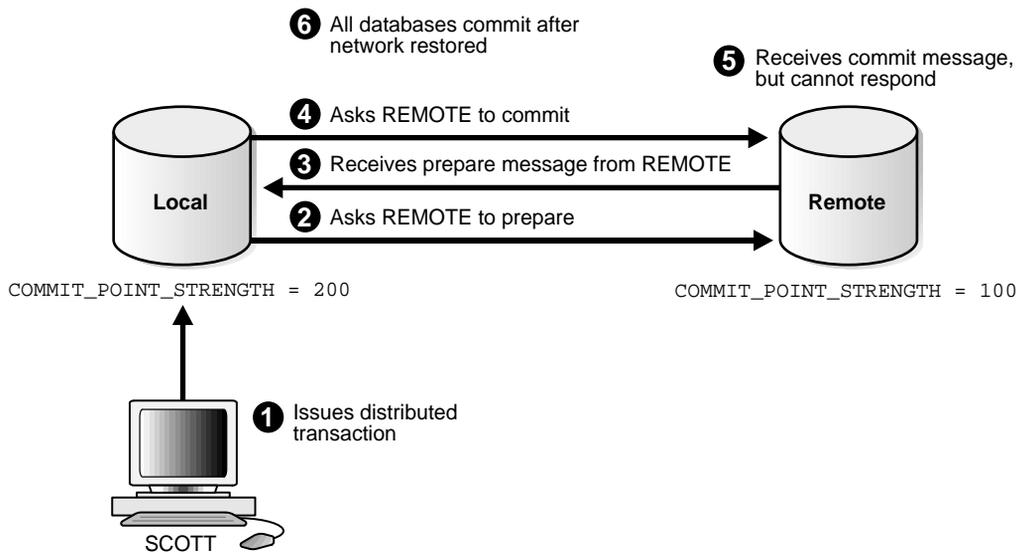
The following steps occur:

1. User `SCOTT` connects to `local` and executes a distributed transaction.
2. The global coordinator, which in this example is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The `remote` database crashes before issuing the prepare response back to `local`.
4. The transaction is ultimately rolled back on each database by the RECO process when the remote site is restored.

Failure During the Commit Phase

Figure 31-6 illustrates the sequence of events when there is a failure during the commit phase of a distributed transaction:

Figure 31-6 Failure During Commit Phase



The following steps occur:

1. User `SCOTT` connects to `local` and executes a distributed transaction.

2. The global coordinator, which in this case is also the commit point site, requests all databases other than the commit point site to promise to commit or roll back when told to do so.
3. The commit point site receives a prepared message from `remote` saying that it will commit.
4. The commit point site commits the transaction locally, then sends a commit message to `remote` asking it to commit.
5. The `remote` database receives the commit message, but cannot respond because of a network failure.
6. The transaction is ultimately committed on the remote database by the RECO process after the network is restored.

See Also: ["Deciding How to Handle In-Doubt Transactions"](#) on page 32-7 for a description of failure situations and how Oracle resolves intervening failures during two-phase commit

Manual Resolution of In-Doubt Transactions

You should only need to resolve an in-doubt transaction in the following cases:

- The in-doubt transaction has locks on critical data or rollback segments.
- The cause of the machine, network, or software failure cannot be repaired quickly.

Resolution of in-doubt transactions can be complicated. The procedure requires that you do the following:

- Identify the transaction identification number for the in-doubt transaction.
- Query the `DBA_2PC_PENDING` and `DBA_2PC_NEIGHBORS` views to determine whether the databases involved in the transaction have committed.
- If necessary, force a commit using the `COMMIT FORCE` statement or a rollback using the `ROLLBACK FORCE` statement.

See Also: The following sections explain how to resolve in-doubt transactions:

- ["Deciding How to Handle In-Doubt Transactions"](#) on page 32-7
- ["Manually Overriding In-Doubt Transactions"](#) on page 32-10

Relevance of System Change Numbers for In-Doubt Transactions

A **system change number** (SCN) is an internal timestamp for a committed version of the database. The Oracle database server uses the SCN clock value to guarantee transaction consistency. For example, when a user commits a transaction, Oracle records an SCN for this commit in the online redo log.

Oracle uses SCNs to coordinate distributed transactions among different databases. For example, Oracle uses SCNs in the following way:

1. An application establishes a connection using a database link.
2. The distributed transaction commits with the highest global SCN among all the databases involved.
3. The commit global SCN is sent to all databases involved in the transaction.

SCNs are important for distributed transactions because they function as a synchronized commit timestamp of a transaction—even if the transaction fails. If a transaction becomes in-doubt, an administrator can use this SCN to coordinate changes made to the global database. The global SCN for the transaction commit can also be used to identify the transaction later, for example, in distributed recovery.

Distributed Transaction Processing: Case Study

In this scenario, a company has separate Oracle database servers, `sales.acme.com` and `warehouse.acme.com`. As users insert sales records into the `sales` database, associated records are being updated at the `warehouse` database.

This case study of distributed processing illustrates:

- The definition of a session tree
- How a commit point site is determined
- When prepare messages are sent
- When a transaction actually commits
- What information is stored locally about the transaction

Stage 1: Client Application Issues DML Statements

At the Sales department, a salesperson uses SQL*Plus to enter a sales order and then commit it. The application issues a number of SQL statements to enter the

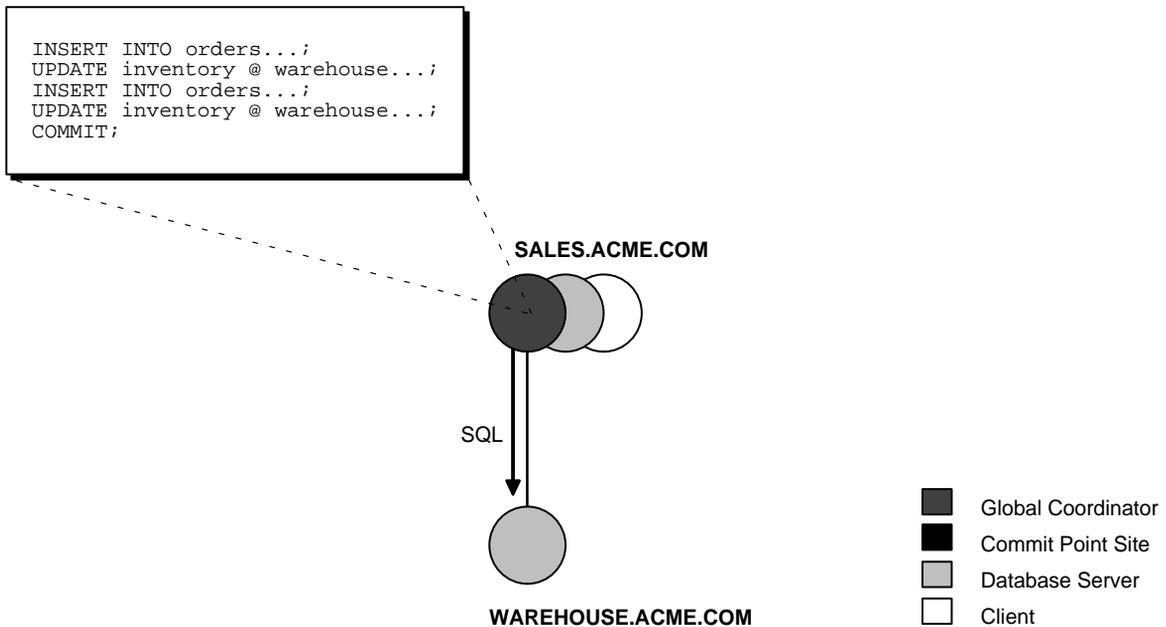
order into the sales database and update the inventory in the warehouse database:

```
CONNECT scott/tiger@sales.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
COMMIT;
```

These SQL statements are part of a single distributed transaction, guaranteeing that all issued SQL statements succeed or fail as a unit. Treating the statements as a unit prevents the possibility of an order being placed and then inventory not being updated to reflect the order. In effect, the transaction guarantees the consistency of data in the global database.

As each of the SQL statements in the transaction executes, the session tree is defined, as shown in Figure 31-7.

Figure 31-7 Defining the Session Tree



Note the following aspects of the transaction:

- An order entry application running on the `sales` database initiates the transaction. Therefore, `sales.acme.com` is the global coordinator for the distributed transaction.
- The order entry application inserts a new sales record into the `sales` database and updates the inventory at the warehouse. Therefore, the nodes `sales.acme.com` and `warehouse.acme.com` are both database servers.
- Because `sales.acme.com` updates the inventory, it is a client of `warehouse.acme.com`.

This stage completes the definition of the session tree for this distributed transaction. Each node in the tree has acquired the necessary data locks to execute the SQL statements that reference local data. These locks remain even after the SQL statements have been executed until the two-phase commit is completed.

Stage 2: Oracle Determines Commit Point Site

Oracle determines the commit point site immediately following the `COMMIT` statement. `sales.acme.com`, the global coordinator, is determined to be the commit point site, as shown in [Figure 31-8](#).

See Also: "[Commit Point Strength](#)" on page 31-8 for more information about how the commit point site is determined

Figure 31-8 Determining the Commit Point Site



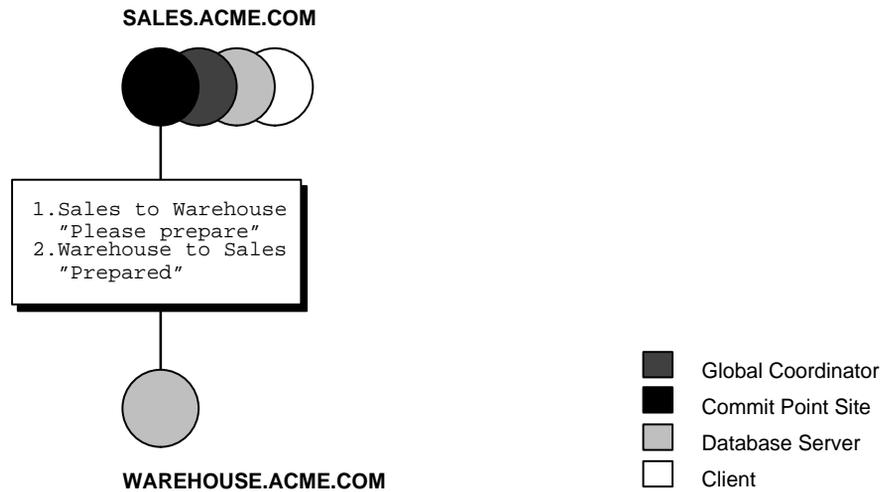
Stage 3: Global Coordinator Sends Prepare Response

The prepare stage involves the following steps:

1. After Oracle determines the commit point site, the global coordinator sends the prepare message to all directly referenced nodes of the session tree, *excluding* the commit point site. In this example, `warehouse.acme.com` is the only node asked to prepare.
2. Node `warehouse.acme.com` tries to prepare. If a node can guarantee that it can commit the locally dependent part of the transaction and can record the commit information in its local redo log, then the node can successfully prepare. In this example, only `warehouse.acme.com` receives a prepare message because `sales.acme.com` is the commit point site.
3. Node `warehouse.acme.com` responds to `sales.acme.com` with a prepared message.

As each node prepares, it sends a message back to the node that asked it to prepare. Depending on the responses, one of the following can happen:

- If *any* of the nodes asked to prepare responds with an abort message to the global coordinator, then the global coordinator tells all nodes to roll back the transaction, and the operation is completed.
- If *all* nodes asked to prepare respond with a prepared or a read-only message to the global coordinator, that is, they have successfully prepared, then the global coordinator asks the commit point site to commit the transaction.

Figure 31–9 Sending and Acknowledging the Prepare Message

Stage 4: Commit Point Site Commits

The committing of the transaction by the commit point site involves the following steps:

1. Node `sales.acme.com`, receiving acknowledgment that `warehouse.acme.com` is prepared, instructs the commit point site to commit the transaction.
2. The commit point site now commits the transaction locally and records this fact in its local redo log.

Even if `warehouse.acme.com` has not yet committed, the outcome of this transaction is predetermined. In other words, the transaction *will* be committed at all nodes even if a given node's ability to commit is delayed.

Stage 5: Commit Point Site Informs Global Coordinator of Commit

This stage involves the following steps:

1. The commit point site tells the global coordinator that the transaction has committed. Because the commit point site and global coordinator are the same node in this example, no operation is required. The commit point site knows that the transaction is committed because it recorded this fact in its online log.

2. The global coordinator confirms that the transaction has been committed on all other nodes involved in the distributed transaction.

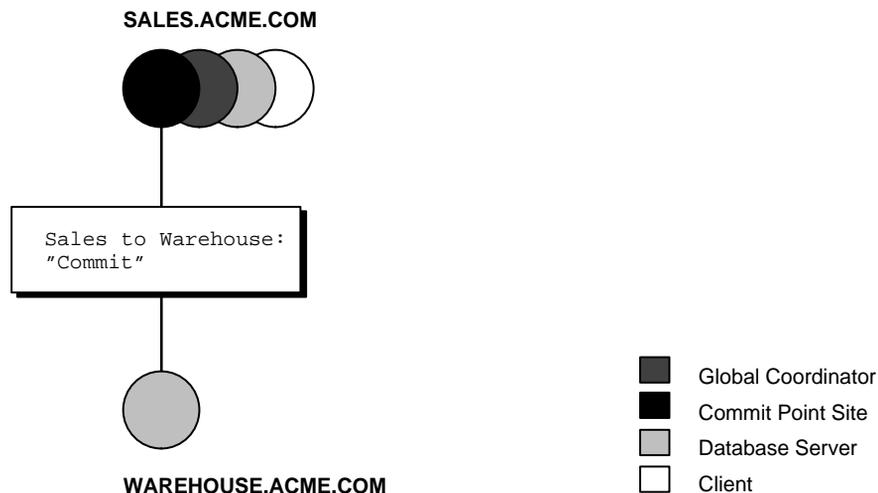
Stage 6: Global and Local Coordinators Tell All Nodes to Commit

The committing of the transaction by all the nodes in the transaction involves the following steps:

1. After the global coordinator has been informed of the commit at the commit point site, it tells all other directly referenced nodes to commit.
2. In turn, any local coordinators instruct their servers to commit, and so on.
3. Each node, including the global coordinator, commits the transaction and records appropriate redo log entries locally. As each node commits, the resource locks that were being held locally for that transaction are released.

In [Figure 31-10](#), `sales.acme.com`, which is both the commit point site and the global coordinator, has already committed the transaction locally. `sales` now instructs `warehouse.acme.com` to commit the transaction.

Figure 31-10 *Instructing Nodes to Commit*



Stage 7: Global Coordinator and Commit Point Site Complete the Commit

The completion of the commit of the transaction occurs in the following steps:

1. After all referenced nodes and the global coordinator have committed the transaction, the global coordinator informs the commit point site of this fact.
2. The commit point site, which has been waiting for this message, erases the status information about this distributed transaction.
3. The commit point site informs the global coordinator that it is finished. In other words, the commit point site forgets about committing the distributed transaction. This action is permissible because all nodes involved in the two-phase commit have committed the transaction successfully, so they will never have to determine its status in the future.
4. The global coordinator finalizes the transaction by forgetting about the transaction itself.

After the completion of the `COMMIT` phase, the distributed transaction is itself complete. The steps described above are accomplished automatically and in a fraction of a second.

Managing Distributed Transactions

This chapter describes how to manage and troubleshoot distributed transactions. The following topics are included in this chapter:

- [Specifying the Commit Point Strength of a Node](#)
- [Naming Transactions](#)
- [Viewing Information About Distributed Transactions](#)
- [Deciding How to Handle In-Doubt Transactions](#)
- [Manually Overriding In-Doubt Transactions](#)
- [Purging Pending Rows from the Data Dictionary](#)
- [Manually Committing an In-Doubt Transaction: Example](#)
- [Data Access Failures Due To Locks](#)
- [Simulating Distributed Transaction Failure](#)
- [Managing Read Consistency](#)

Specifying the Commit Point Strength of a Node

The database with the highest commit point strength determines which node commits first in a distributed transaction. When specifying a commit point strength for each node, ensure that the most critical server will be nonblocking if a failure occurs during a prepare or commit phase. The `COMMIT_POINT_STRENGTH` initialization parameter determines a node's commit point strength.

The default value is operating system-dependent. The range of values is any integer from 0 to 255. For example, to set the commit point strength of a database to 200, include the following line in that database's initialization parameter file:

```
COMMIT_POINT_STRENGTH = 200
```

The commit point strength is only used to determine the commit point site in a distributed transaction.

When setting the commit point strength for a database, note the following considerations:

- Because the commit point site stores information about the status of the transaction, the commit point site should not be a node that is frequently unreliable or unavailable in case other nodes need information about the transaction's status.
- Set the commit point strength for a database relative to the amount of critical shared data in the database. For example, a database on a mainframe computer usually shares more data among users than a database on a PC. Therefore, set the commit point strength of the mainframe to a higher value than the PC.

See Also: ["Commit Point Site"](#) on page 31-6 for a conceptual overview of commit points

Naming Transactions

Starting with Oracle9i you can name a transaction. This is useful for identifying a specific distributed transaction and replaces the use of the `COMMIT COMMENT` statement for this purpose.

To name a transaction, use the `SET TRANSACTION . . . NAME` statement. For example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  
NAME 'update inventory checkpoint 0';
```

This example shows that the user started a new transaction with isolation level equal to `SERIALIZABLE` and named it `'update inventory checkpoint 0'`.

For distributed transactions, the name is sent to participating sites when a transaction is committed. If a `COMMIT COMMENT` exists, it is ignored when a transaction name exists.

The transaction name is displayed in the `NAME` column of the `V$TRANSACTION` view, and is placed in the `TRAN_COMMENT` column of the view when the transaction is committed.

Viewing Information About Distributed Transactions

The data dictionary of each database stores information about all open distributed transactions. You can use data dictionary tables and views to gain information about the transactions. This section contains the following topics:

- [Determining the ID Number and Status of Prepared Transactions](#)
- [Tracing the Session Tree of In-Doubt Transactions](#)

Determining the ID Number and Status of Prepared Transactions

The following view shows the database links that have been defined at the local database and stored in the data dictionary:

View	Purpose
<code>DBA_2PC_PENDING</code>	Lists all in-doubt distributed transactions. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

Use this view to determine the global commit number for a particular transaction ID. You can use this global commit number when manually resolving an in-doubt transaction.

The following table shows the most relevant columns (for a description of all the columns in the view, see *Oracle9i Database Reference*):

Table 32–1 DBA_2PC_PENDING

Column	Description
LOCAL_TRAN_ID	<p>Local transaction identifier in the format <i>integer.integer.integer</i>.</p> <p>Note: When the LOCAL_TRAN_ID and the GLOBAL_TRAN_ID for a connection are the same, the node is the global coordinator of the transaction.</p>
GLOBAL_TRAN_ID	<p>Global database identifier in the format <i>global_db_name.db_hex_id.local_tran_id</i>, where <i>db_hex_id</i> is an eight-character hexadecimal value used to uniquely identify the database. This common transaction ID is the same on every node for a distributed transaction.</p> <p>Note: When the LOCAL_TRAN_ID and the GLOBAL_TRAN_ID for a connection are the same, the node is the global coordinator of the transaction.</p>
STATE	<p>STATE can have the following values:</p> <ul style="list-style-type: none"> ■ Collecting This category normally applies only to the global coordinator or local coordinators. The node is currently collecting information from other database servers before it can decide whether it can prepare. ■ Prepared The node has prepared and may or may not have acknowledged this to its local coordinator with a prepared message. However, no commit request has been received. The node remains prepared, holding any local resource locks necessary for the transaction to commit. ■ Committed The node (any type) has committed the transaction, but other nodes involved in the transaction may not have done the same. That is, the transaction is still pending at one or more nodes. ■ Forced Commit A pending transaction can be forced to commit at the discretion of a database administrator. This entry occurs if a transaction is manually committed at a local node. ■ Forced termination (rollback) A pending transaction can be forced to roll back at the discretion of a database administrator. This entry occurs if this transaction is manually rolled back at a local node.

Table 32–1 DBA_2PC_PENDING(Cont.)

Column	Description
MIXED	YES means that part of the transaction was committed on one node and rolled back on another node.
TRAN_COMMENT	Transaction comment or, if using transaction naming, the transaction name is placed here when the transaction is committed.
HOST	Name of the host machine.
COMMIT#	Global commit number for committed transactions.

Execute the following script, named `pending_txn_script`, to query pertinent information in `DBA_2PC_PENDING` (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
COL GLOBAL_TRAN_ID FORMAT A30
COL STATE FORMAT A8
COL MIXED FORMAT A3
COL HOST FORMAT A10
COL COMMIT# FORMAT A10

SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
FROM DBA_2PC_PENDING
/
```

```
SQL> @pending_txn_script
```

```
LOCAL_TRAN_ID GLOBAL_TRAN_ID STATE MIX HOST COMMIT#
-----
1.15.870 HQ.ACME.COM.ef192da4.1.15.870 commit no dlsun183 115499
```

This output indicates that local transaction 1.15.870 has been committed on this node, but it may be pending on one or more other nodes. Because `LOCAL_TRAN_ID` and the local part of `GLOBAL_TRAN_ID` are the same, the node is the global coordinator of the transaction.

Tracing the Session Tree of In-Doubt Transactions

The following view shows which in-doubt transactions are incoming from a remote client and which are outgoing to a remote server:

View	Purpose
DBA_2PC_NEIGHBORS	<p>Lists all incoming (from remote client) and outgoing (to remote server) in-doubt distributed transactions. It also indicates whether the local node is the commit point site in the transaction.</p> <p>The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.</p>

When a transaction is in-doubt, you may need to determine which nodes performed which roles in the session tree. Use to this view to determine:

- All the incoming and outgoing connections for a given transaction
- Whether the node is the commit point site in a given transaction
- Whether the node is a global coordinator in a given transaction (because its local transaction ID and global transaction ID are the same)

The following table shows the most relevant columns (for an account of all the columns in the view, see *Oracle9i Database Reference*):

Table 32–2 DBA_2PC_NEIGHBORS

Column	Description
LOCAL_TRAN_ID	<p>Local transaction identifier with the format <i>integer.integer.integer</i>.</p> <p>Note: When LOCAL_TRAN_ID and GLOBAL_TRAN_ID.DBA_2PC_PENDING for a connection are the same, the node is the global coordinator of the transaction.</p>
IN_OUT	IN for incoming transactions; OUT for outgoing transactions.
DATABASE	For incoming transactions, the name of the client database that requested information from this local node; for outgoing transactions, the name of the database link used to access information on a remote server.
DBUSER_OWNER	For incoming transactions, the local account used to connect by the remote database link; for outgoing transactions, the owner of the database link.

Table 32–2 DBA_2PC_NEIGHBORS

Column	Description
INTERFACE	<p>C is a commit message; N is either a message indicating a prepared state or a request for a read-only commit.</p> <p>When IN_OUT is OUT, C means that the child at the remote end of the connection is the commit point site and knows whether to commit or terminate. N means that the local node is informing the remote node that it is prepared.</p> <p>When IN_OUT is IN, C means that the local node or a database at the remote end of an outgoing connection is the commit point site. N means that the remote node is informing the local node that it is prepared.</p>

Execute the following script, named `neighbors_script`, to query pertinent information in `DBA_2PC_PENDING` (sample output included):

```
COL LOCAL_TRAN_ID FORMAT A13
COL IN_OUT FORMAT A6
COL DATABASE FORMAT A25
COL DBUSER_OWNER FORMAT A15
COL INTERFACE FORMAT A3
SELECT LOCAL_TRAN_ID, IN_OUT, DATABASE, DBUSER_OWNER, INTERFACE
FROM DBA_2PC_NEIGHBORS
/
```

```
SQL> CONNECT SYS/password@hq.acme.com
SQL> @neighbors_script
```

```
LOCAL_TRAN_ID IN_OUT DATABASE DBUSER_OWNER INT
-----
1.15.870      out   SALES.ACME.COM      SYS          C
```

This output indicates that the local node sent an outgoing request to remote server `sales` to commit transaction `1.15.870`. If `sales` committed the transaction but no other node did, then you know that `sales` is the commit point site—because the commit point site always commits first.

Deciding How to Handle In-Doubt Transactions

A transaction is in-doubt when there is a failure during any aspect of the two-phase commit. Distributed transactions become in-doubt in the following ways:

- A server machine running Oracle software crashes
- A network connection between two or more Oracle databases involved in distributed processing is disconnected
- An unhandled software error occurs

See Also: ["In-Doubt Transactions"](#) on page 31-15 for a conceptual overview of in-doubt transactions

You can manually force the commit or rollback of a local, in-doubt distributed transaction. Because this operation can generate consistency problems, perform it only when specific conditions exist.

This section contains the following topics:

- [Discovering Problems with a Two-Phase Commit](#)
- [Determining Whether to Perform a Manual Override](#)
- [Analyzing the Transaction Data](#)

Discovering Problems with a Two-Phase Commit

The user application that commits a distributed transaction is informed of a problem by one of the following error messages:

```
ORA-02050: transaction ID rolled back,  
           some remote dbs may be in-doubt  
ORA-02051: transaction ID committed,  
           some remote dbs may be in-doubt  
ORA-02054: transaction ID in-doubt
```

A robust application should save information about a transaction if it receives any of the above errors. This information can be used later if manual distributed transaction recovery is desired.

No action is required by the administrator of any node that has one or more in-doubt distributed transactions due to a network or system failure. The automatic recovery features of Oracle transparently complete any in-doubt transaction so that the same outcome occurs on all nodes of a session tree (that is, all commit or all roll back) after the network or system failure is resolved.

In extended outages, however, you can force the commit or rollback of a transaction to release any locked data. Applications must account for such possibilities.

Determining Whether to Perform a Manual Override

Override a specific in-doubt transaction manually *only* when one of the following situations exists:

- The in-doubt transaction locks data that is required by other transactions. This situation occurs when the `ORA-01591` error message interferes with user transactions.
- An in-doubt transaction prevents the extents of a rollback segment from being used by other transactions. The first portion of an in-doubt distributed transaction's local transaction ID corresponds to the ID of the rollback segment, as listed by the data dictionary views `DBA_2PC_PENDING` and `DBA_ROLLBACK_SEGS`.
- The failure preventing the two-phase commit phases to complete cannot be corrected in an acceptable time period. Examples of such cases include a telecommunication network that has been damaged or a damaged database that requires a long recovery time.

Normally, you should make a decision to locally force an in-doubt distributed transaction in consultation with administrators at other locations. A wrong decision can lead to database inconsistencies that can be difficult to trace and that you must manually correct.

If the conditions above do not apply, *always* allow the automatic recovery features of Oracle to complete the transaction. If any of the above criteria are met, however, consider a local override of the in-doubt transaction.

Analyzing the Transaction Data

If you decide to force the transaction to complete, analyze available information with the following goals in mind.

Find a Node That Committed or Rolled Back

Use the `DBA_2PC_PENDING` view to find a node that has either committed or rolled back the transaction. If you can find a node that has already resolved the transaction, then you can follow the action taken at that node.

Look For Transaction Comments

See if any information is given in the `TRAN_COMMENT` column of `DBA_2PC_PENDING` for the distributed transaction. Comments are included in the `COMMENT` clause of the `COMMIT` statement, or if transaction naming is used, the transaction name is placed in the `TRAN_COMMENT` field when the transaction is committed.

For example, an in-doubt distributed transaction's comment can indicate the origin of the transaction and what type of transaction it is:

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```

The `SET TRANSACTION . . . NAME` statement could also have been used (and is preferable) to provide this information in a transaction name.

See Also: ["Naming Transactions"](#) on page 32-2

Look For Transaction Advice

See if any information is given in the `ADVICE` column of `DBA_2PC_PENDING` for the distributed transaction. An application can prescribe advice about whether to force the commit or force the rollback of separate parts of a distributed transaction with the `ADVISE` clause of the `ALTER SESSION` statement.

The advice sent during the prepare phase to each node is the advice in effect at the time the most recent DML statement executed at that database in the current transaction.

For example, consider a distributed transaction that moves an employee record from the `emp` table at one node to the `emp` table at another node. The transaction can protect the record—even when administrators independently force the in-doubt transaction at each node—by including the following sequence of SQL statements:

```
ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq ... ; /*advice to commit at HQ */
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/

ALTER SESSION ADVISE NOTHING;
```

If you manually force the in-doubt transaction following the given advice, the worst that can happen is that each node has a copy of the employee record; the record cannot disappear.

Manually Overriding In-Doubt Transactions

Use the `COMMIT` or `ROLLBACK` statement with the `FORCE` option and a text string that indicates either the local or global transaction ID of the in-doubt transaction to commit.

Note: In all examples, the transaction is committed or rolled back on the local node, and the local pending transaction table records a value of forced commit or forced termination for the `STATE` column of this transaction's row.

This section contains the following topics:

- [Manually Committing an In-Doubt Transaction](#)
- [Manually Rolling Back an In-Doubt Transaction](#)

Manually Committing an In-Doubt Transaction

Before attempting to commit the transaction, ensure that you have the proper privileges. Note the following requirements:

If the transaction was committed by...	Then you must have this privilege...
You	FORCE TRANSACTION
Another user	FORCE ANY TRANSACTION

Committing Using Only the Transaction ID

The following SQL statement commits an in-doubt transaction:

```
COMMIT FORCE 'transaction_id';
```

The variable `transaction_id` is the identifier of the transaction as specified in either the `LOCAL_TRAN_ID` or `GLOBAL_TRAN_ID` columns of the `DBA_2PC_PENDING` data dictionary view.

For example, assume that you query `DBA_2PC_PENDING` and determine that `LOCAL_TRAN_ID` for a distributed transaction is `1:45.13`.

You then issue the following SQL statement to force the commit of this in-doubt transaction:

```
COMMIT FORCE '1.45.13';
```

Committing Using an SCN

Optionally, you can specify the SCN for the transaction when forcing a transaction to commit. This feature allows you to commit an in-doubt transaction with the SCN assigned when it was committed at other nodes.

Consequently, you maintain the synchronized commit time of the distributed transaction even if there is a failure. Specify an SCN only when you can determine the SCN of the same transaction already committed at another node.

For example, assume you want to manually commit a transaction with the following global transaction ID:

```
SALES.ACME.COM.55d1c563.1.93.29
```

First, query the `DBA_2PC_PENDING` view of a remote database also involved with the transaction in question. Note the SCN used for the commit of the transaction at that node. Specify the SCN when committing the transaction at the local node. For example, if the SCN is 829381993, issue:

```
COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29', 829381993;
```

See Also: *Oracle9i SQL Reference* for more information about using the `COMMIT` statement

Manually Rolling Back an In-Doubt Transaction

Before attempting to roll back the in-doubt distributed transaction, ensure that you have the proper privileges. Note the following requirements:

If the transaction was committed by...	Then you must have this privilege...
You	FORCE TRANSACTION
Another user	FORCE ANY TRANSACTION

The following SQL statement rolls back an in-doubt transaction:

```
ROLLBACK FORCE 'transaction_id';
```

The variable *transaction_id* is the identifier of the transaction as specified in either the `LOCAL_TRAN_ID` or `GLOBAL_TRAN_ID` columns of the `DBA_2PC_PENDING` data dictionary view.

For example, to roll back the in-doubt transaction with the local transaction ID of 2.9.4, use the following statement:

```
ROLLBACK FORCE '2.9.4';
```

Note: You cannot roll back an in-doubt transaction to a savepoint.

See Also: *Oracle9i SQL Reference* for more information about using the ROLLBACK statement

Purging Pending Rows from the Data Dictionary

Before RECO recovers an in-doubt transaction, the transaction appears in DBA_2PC_PENDING.STATE as COLLECTING, COMMITTED, or PREPARED. If you force an in-doubt transaction using COMMIT FORCE or ROLLBACK FORCE, then the states FORCED COMMIT or FORCED ROLLBACK may appear.

Automatic recovery normally deletes entries in these states. The only exception is when recovery discovers a forced transaction that is in a state inconsistent with other sites in the transaction. In this case, the entry can be left in the table and the MIXED column in DBA_2PC_PENDING has a value of YES. These entries can be cleaned up with the DBMS_TRANSACTION.PURGE_MIXED procedure.

If automatic recovery is not possible because a remote database has been permanently lost, then recovery cannot identify the re-created database because it receives a new database ID when it is re-created. In this case, you must use the PURGE_LOST_DB_ENTRY procedure in the DBMS_TRANSACTION package to clean up the entries. The entries do not hold up database resources, so there is no urgency in cleaning them up.

Executing the PURGE_LOST_DB_ENTRY Procedure

To manually remove an entry from the data dictionary, use the following syntax (where *trans_id* is the identifier for the transaction):

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('trans_id');
```

For example, to purge pending distributed transaction 1.44.99, enter the following statement in SQL*Plus:

```
EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('1.44.99');
```

Execute this procedure only if significant reconfiguration has occurred so that automatic recovery cannot resolve the transaction. Examples include:

- Total loss of the remote database
- Reconfiguration in software resulting in loss of two-phase commit capability
- Loss of information from an external transaction coordinator such as a TPMonitor

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

Determining When to Use `DBMS_TRANSACTION`

The following tables indicates what the various states indicate about the distributed transaction what the administrator's action should be:

STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Collecting	Rolled back	Rolled back	None	<code>PURGE_LOST_DB_ENTRY</code> (only if autorecovery cannot resolve transaction)
Committed	Committed	Committed	None	<code>PURGE_LOST_DB_ENTRY</code> (only if autorecovery cannot resolve transaction)
Prepared	Unknown	Prepared	None	Force commit or rollback
Forced commit	Unknown	Committed	None	<code>PURGE_LOST_DB_ENTRY</code> (only if autorecovery cannot resolve transaction)
Forced rollback	Unknown	Rolled back	None	<code>PURGE_LOST_DB_ENTRY</code> (only if autorecovery cannot resolve transaction)
Forced commit	Mixed	Committed	Manually remove inconsistencies then use <code>PURGE_MIXED</code>	-

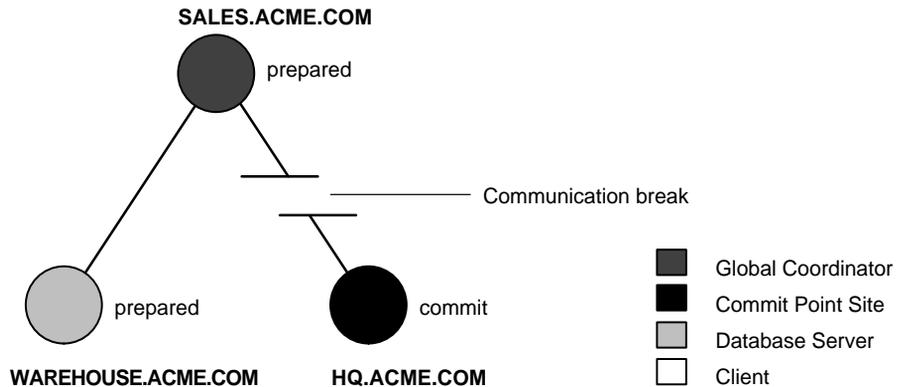
STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Forced rollback	Mixed	Rolled back	Manually remove inconsistencies then use PURGE_MIXED	-

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

Manually Committing an In-Doubt Transaction: Example

Figure 32-1, illustrates a failure during the commit of a distributed transaction. In this failure case, the prepare phase completes. During the commit phase, however, the commit point site's commit confirmation never reaches the global coordinator, even though the commit point site committed the transaction. Inventory data is locked and cannot be accessed because the in-doubt transaction is critical to other transactions. Further, the locks must be held until the in-doubt transaction either commits or rolls back.

Figure 32-1 Example of an In-Doubt Distributed Transaction



You can manually force the local portion of the in-doubt transaction by following the steps detailed in the following sections:

[Step 1: Record User Feedback](#)

[Step 2: Query DBA_2PC_PENDING](#)

Step 3: Query DBA_2PC_NEIGHBORS on Local Node

Step 4: Querying Data Dictionary Views on All Nodes

Step 5: Commit the In-Doubt Transaction

Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING

Step 1: Record User Feedback

The users of the local database system that conflict with the locks of the in-doubt transaction receive the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

In this case, 1.21.17 is the local transaction ID of the in-doubt distributed transaction. You should request and record this ID number from users that report problems to identify which in-doubt transactions should be forced.

Step 2: Query DBA_2PC_PENDING

After connecting with SQL*Plus to warehouse, query the local DBA_2PC_PENDING data dictionary view to gain information about the in-doubt transaction:

```
CONNECT SYS/password@warehouse.acme.com
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

Oracle returns the following information:

Column Name	Value
LOCAL_TRAN_ID	1.21.17
GLOBAL_TRAN_ID	SALES.ACME.COM.55d1c563.1.93.29
STATE	prepared
MIXED	no
ADVICE	
TRAN_COMMENT	Sales/New Order/Trans_type 10B
FAIL_TIME	31-MAY-91
FORCE_TIME	
RETRY_TIME	31-MAY-91
OS_USER	SWILLIAMS
OS_TERMINAL	TWA139:
HOST	system1
DB_USER	SWILLIAMS
COMMIT#	

Determining the Global Transaction ID

The global transaction ID is the common transaction ID that is the same on every node for a distributed transaction. It is of the form:

global_database_name.hhhhhhh.local_transaction_id

where:

- *global_database_name* is the database name of the global coordinator.
- *hhhhhhh* is the internal database identifier of the global coordinator (in hexadecimal).
- *local_transaction_id* is the corresponding local transaction ID assigned on the global coordinator.

Note that the last portion of the global transaction ID and the local transaction ID match at the global coordinator. In the example, you can tell that *warehouse* is *not* the global coordinator because these numbers do not match:

```
LOCAL_TRAN_ID      1.21.17
GLOBAL_TRAN_ID    ... 1.93.29
```

Determining the State of the Transaction

The transaction on this node is in a prepared state:

```
STATE      prepared
```

Therefore, *warehouse* waits for its coordinator to send either a commit or a rollback request.

Looking For Comments or Advice

The transaction's comment or advice can include information about this transaction. If so, use this comment to your advantage. In this example, the origin and transaction type is in the transaction's comment:

```
TRAN_COMMENT      Sales/New Order/Trans_type 10B
```

It could also be provided as a transaction name with a `SET TRANSACTION ... NAME` statement.

This information can reveal something that helps you decide whether to commit or rollback the local portion of the transaction. If useful comments do not accompany an in-doubt transaction, you must complete some extra administrative work to trace the session tree and find a node that has resolved the transaction.

Step 3: Query DBA_2PC_NEIGHBORS on Local Node

The purpose of this step is to climb the session tree so that you find coordinators, eventually reaching the global coordinator. Along the way, you may find a coordinator that has resolved the transaction. If not, you can eventually work your way to the commit point site, which will always have resolved the in-doubt transaction. To trace the session tree, query the DBA_2PC_NEIGHBORS view on each node.

In this case, you query this view on the warehouse database:

```
CONNECT SYS/password@warehouse.acme.com
SELECT * FROM DBA_2PC_NEIGHBORS
  WHERE LOCAL_TRAN_ID = '1.21.17'
  ORDER BY SESS#, IN_OUT;
```

Column Name	Value
LOCAL_TRAN_ID	1.21.17
IN_OUT	in
DATABASE	SALES.ACME.COM
DBUSER_OWNER	SWILLIAMS
INTERFACE	N
DBID	000003F4
SESS#	1
BRANCH	0100

Obtaining Database Role and Database Link Information

The DBA_2PC_NEIGHBORS view provides information about connections associated with an in-doubt transaction. Information for each connection is different, based on whether the connection is **inbound** (IN_OUT = in) or **outbound** (IN_OUT = out):

IN_OUT	Meaning	DATABASE	DBUSER_OWNER
in	Your node is a server of another node.	Lists the name of the client database that connected to your node.	Lists the local account for the database link connection that corresponds to the in-doubt transaction.
out	Your node is a client of other servers.	Lists the name of the database link that connects to the remote node.	Lists the owner of the database link for the in-doubt transaction.

In this example, the `IN_OUT` column reveals that the warehouse database is a server for the `sales` client, as specified in the `DATABASE` column:

```
IN_OUT          in
DATABASE        SALES.ACME.COM
```

The connection to `warehouse` was established through a database link from the `swilliams` account, as shown by the `DBUSER_OWNER` column:

```
DBUSER_OWNER    SWILLIAMS
```

Determining the Commit Point Site

Additionally, the `INTERFACE` column tells whether the local node or a subordinate node is the commit point site:

```
INTERFACE       N
```

Neither `warehouse` nor any of its descendants is the commit point site, as shown by the `INTERFACE` column.

Step 4: Querying Data Dictionary Views on All Nodes

At this point, you can contact the administrator at the located nodes and ask each person to repeat Steps 2 and 3 using the global transaction ID.

Note: If you can directly connect to these nodes with another network, you can repeat Steps 2 and 3 yourself.

For example, the following results are returned when Steps 2 and 3 are performed at `sales` and `hq`.

Checking the Status of Pending Transactions at sales

At this stage, the `sales` administrator queries the `DBA_2PC_PENDING` data dictionary view:

```
SQL> CONNECT SYS/password@sales.acme.com
SQL> SELECT * FROM DBA_2PC_PENDING
      > WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';
```

Column Name	Value
LOCAL_TRAN_ID	1.93.29
GLOBAL_TRAN_ID	SALES.ACME.COM.55d1c563.1.93.29

```

STATE                prepared
MIXED                no
ADVICE
TRAN_COMMENT         Sales/New Order/Trans_type 10B
FAIL_TIME            31-MAY-91
FORCE_TIME
RETRY_TIME           31-MAY-91
OS_USER              SWILLIAMS
OS_TERMINAL          TWA139:
HOST                 system1
DB_USER              SWILLIAMS
COMMIT#
    
```

Determining the Coordinators and Commit Point Site at sales

Next, the sales administrator queries `DBA_2PC_NEIGHBORS` to determine the global and local coordinators as well as the commit point site:

```

SELECT * FROM DBA_2PC_NEIGHBORS
WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29'
ORDER BY SESS#, IN_OUT;
    
```

This query returns three rows:

- The connection to warehouse
- The connection to hq
- The connection established by the user

Reformatted information corresponding to the rows for the warehouse connection appears below:

Column Name	Value
LOCAL_TRAN_ID	1.93.29
IN_OUT	OUT
DATABASE	WAREHOUSE.ACME.COM
DBUSER_OWNER	SWILLIAMS
INTERFACE	N
DBID	55d1c563
SESS#	1
BRANCH	1

Reformatted information corresponding to the rows for the hq connection appears below:

Column Name	Value
LOCAL_TRAN_ID	1.93.29
IN_OUT	OUT
DATABASE	HQ.ACME.COM
DBUSER_OWNER	ALLEN
INTERFACE	C
DBID	00000390
SESS#	1
BRANCH	1

The information from the previous queries reveal the following:

- `sales` is the global coordinator because the local transaction ID and global transaction ID match.
- Two outbound connections are established from this node, but no inbound connections. `sales` is not the server of another node.
- `hq` or one of its servers is the commit point site.

Checking the Status of Pending Transactions at HQ:

At this stage, the `hq` administrator queries the `DBA_2PC_PENDING` data dictionary view:

```
SELECT * FROM DBA_2PC_PENDING@hq.acme.com
WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';
```

Column Name	Value
LOCAL_TRAN_ID	1.45.13
GLOBAL_TRAN_ID	SALES.ACME.COM.55d1c563.1.93.29
STATE	COMMIT
MIXED	NO
ACTION	
TRAN_COMMENT	Sales/New Order/Trans_type 10B
FAIL_TIME	31-MAY-91
FORCE_TIME	
RETRY_TIME	31-MAY-91
OS_USER	SWILLIAMS
OS_TERMINAL	TWA139:
HOST	SYSTEM1
DB_USER	SWILLIAMS
COMMIT#	129314

At this point, you have found a node that resolved the transaction. As the view reveals, it has been committed and assigned a commit ID number:

STATE	COMMIT
COMMIT#	129314

Therefore, you can force the in-doubt transaction to commit at your local database. It is a good idea to contact any other administrators you know that could also benefit from your investigation.

Step 5: Commit the In-Doubt Transaction

You contact the administrator of the `sales` database, who manually commits the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS/password@sales.acme.com
SQL> COMMIT FORCE 'SALES.ACME.COM,55d1c563.1.93.29';
```

As administrator of the `warehouse` database, you manually commit the in-doubt transaction using the global ID:

```
SQL> CONNECT SYS/password@warehouse.acme.com
SQL> COMMIT FORCE 'SALES.ACME.COM,55d1c563.1.93.29';
```

Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING

After you manually force a transaction to commit or roll back, the corresponding row in the pending transaction table remains. The state of the transaction is changed depending on how you forced the transaction.

Every Oracle database has a **pending transaction table**. This is a special table that stores information about distributed transactions as they proceed through the two-phase commit phases. You can query a database's pending transaction table through the `DBA_2PC_PENDING` data dictionary view (see [Table 32-1](#)).

Also of particular interest in the pending transaction table is the mixed outcome flag as indicated in `DBA_2PC_PENDING.MIXED`. You can make the wrong choice if a pending transaction is forced to commit or roll back. For example, the local administrator rolls back the transaction, but the other nodes commit it. Incorrect decisions are detected automatically, and the damage flag for the corresponding pending transaction's record is set (`MIXED=yes`).

The RECO (Recoverer) background process uses the information in the pending transaction table to finalize the status of in-doubt transactions. You can also use the

information in the pending transaction table to manually override the automatic recovery procedures for pending distributed transactions.

All transactions automatically resolved by RECO are removed from the pending transaction table. Additionally, all information about in-doubt transactions correctly resolved by an administrator (as checked when RECO reestablishes communication) are automatically removed from the pending transaction table. However, all rows resolved by an administrator that result in a mixed outcome across nodes remain in the pending transaction table of all involved nodes until they are manually deleted using `DBMS_TRANSACTIONS.PURGE_MIXED`.

Data Access Failures Due To Locks

When you issue a SQL statement, Oracle attempts to lock the resources needed to successfully execute the statement. If the requested data is currently held by statements of other uncommitted transactions, however, and remains locked for a long time, a timeout occurs.

Consider the following scenarios involving data access failure:

- [Transaction Timeouts](#)
- [Locks from In-Doubt Transactions](#)

Transaction Timeouts

A DML statement that requires locks on a remote database can be blocked if another transaction own locks on the requested data. If these locks continue to block the requesting SQL statement, then the following sequence of events occurs:

1. A timeout occurs.
2. Oracle rolls back the statement.
3. Oracle returns this error message to the user:

```
ORA-02049: time-out: distributed transaction waiting for lock
```

Because the transaction did not modify data, no actions are necessary as a result of the timeout. Applications should proceed as if a deadlock has been encountered. The user who executed the statement can try to reexecute the statement later. If the lock persists, then the user should contact an administrator to report the problem.

Locks from In-Doubt Transactions

A query or DML statement that requires locks on a local database can be blocked indefinitely due to the locked resources of an in-doubt distributed transaction. In this case, Oracle issues the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction identifier
```

In this case, Oracle rolls back the SQL statement immediately. The user who executed the statement can try to reexecute the statement later. If the lock persists, the user should contact an administrator to report the problem, *including* the ID of the in-doubt distributed transaction.

The chances of the above situations occurring are rare considering the low probability of failures during the critical portions of the two-phase commit. Even if such a failure occurs, and assuming quick recovery from a network or system failure, problems are automatically resolved without manual intervention. Thus, problems usually resolve before they can be detected by users or database administrators.

Simulating Distributed Transaction Failure

You can force the failure of a distributed transaction for the following reasons:

- To observe RECO automatically resolving the local portion of the transaction
- To practice manually resolving in-doubt distributed transactions and observing the results

The RECO background process of an Oracle instance automatically resolves failures involving distributed transactions. At exponentially growing time intervals, the RECO background process of a node attempts to recover the local portion of an in-doubt distributed transaction.

RECO can use an existing connection or establish a new connection to other nodes involved in the failed transaction. When a connection is established, RECO automatically resolves all in-doubt transactions. Rows corresponding to any resolved in-doubt transactions are automatically removed from each database's pending transaction table.

You can enable and disable RECO using the `ALTER SYSTEM` statement with the `ENABLE/DISABLE DISTRIBUTED RECOVERY` options. For example, you can temporarily disable RECO to force the failure of a two-phase commit and manually resolve the in-doubt transaction.

The following statement disables RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Alternatively, the following statement enables RECO so that in-doubt transactions are automatically resolved:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

Note: Single-process instances (for example, a PC running MS-DOS) have no separate background processes, and therefore no RECO process. Therefore, when a single-process instance that participates in a distributed system is started, you must manually enable distributed recovery using the statement above.

See Also: Your Oracle operating system specific documentation for more information about distributed transaction recovery for single-process instances

Managing Read Consistency

An important restriction exists in Oracle's implementation of distributed read consistency. The problem arises because each system has its own SCN, which you can view as the database's internal timestamp. The Oracle database server uses the SCN to decide which version of data is returned from a query.

The SCNs in a distributed transaction are synchronized at the end of each remote SQL statement and at the start and end of each transaction. Between two nodes that have heavy traffic and especially distributed updates, the synchronization is frequent. Nevertheless, no practical way exists to keep SCNs in a distributed system absolutely synchronized: a window always exists in which one node may have an SCN that is somewhat in the past with respect to the SCN of another node.

Because of the SCN gap, you can execute a query that uses a slightly old snapshot, so that the most recent changes to the remote database are not seen. In accordance with read consistency, a query can therefore retrieve consistent, but out-of-date data. Note that all data retrieved by the query will be from the old SCN, so that if a locally executed update transaction updates two tables at a remote node, then data selected from both tables in the next remote access contain data prior to the update.

One consequence of the SCN gap is that two consecutive `SELECT` statements can retrieve different data even though no DML has been executed between the two

statements. For example, you can issue an update statement and then commit the update on the remote database. When you issue a `SELECT` statement on a view based on this remote table, the view does not show the update to the row. The next time that you issue the `SELECT` statement, the update is present.

You can use the following techniques to ensure that the SCNs of the two machines are synchronized just before a query:

- Because SCNs are synchronized at the end of a remote query, precede each remote query with a dummy remote query to the same site, for example, `SELECT * FROM DUAL@REMOTE.`
- Because SCNs are synchronized at the start of every remote transaction, commit or roll back the current transaction before issuing the remote query.

A

- abort response, 31-13
 - two-phase commit, 31-13
- accounts
 - operating system
 - database administrator, 1-10
 - user
 - SYS and SYSTEM, 1-11
- ADD LOGFILE MEMBER option
 - ALTER DATABASE statement, 7-13
- ADD LOGFILE option
 - ALTER DATABASE statement, 7-13
- ADD PARTITION clause, 17-27
- ADD SUBPARTITION clause, 17-29, 17-30
- ADMIN OPTION
 - about, 25-12
 - revoking roles/privileges, 25-16
- ADMIN_TABLES procedure, 22-4
 - DBMS_REPAIR package
 - ADMIN_TABLES procedure, 22-3
 - examples
 - building orphan key table, 22-10
 - building repair table, 22-9
- ADMINISTER_RESOURCE_MANAGER system
 - privilege, 27-8
- administration
 - distributed databases, 29-1
 - tools, 28-31
- administrators
 - application, 1-4
- AFTER SUSPEND system event, 14-21
- AFTER SUSPEND trigger, 14-21
 - example of registering, 14-23
- agent
 - Heterogeneous Services, definition of, 28-5
- aggregate functions, 29-33
- alert file
 - job failures and, 10-10
- alert log
 - about, 5-15
 - location of, 5-16
 - size of, 5-16
 - using, 5-15
 - when written, 5-17
- ALL_DB_LINKS view, 29-21
- ALL_JOBS view
 - jobs in system, viewing, 10-15
- allocation
 - extents, 15-12
 - minimizing extents for rollback segments, 13-24
- ALTER CLUSTER statement
 - ALLOCATE EXTENT clause, 18-9
 - using for hash clusters, 19-9
 - using for index clusters, 18-9
- ALTER DATABASE statement
 - ADD LOGFILE MEMBER option, 7-13
 - ADD LOGFILE option, 7-13
 - ARCHIVELOG option, 8-5
 - CLEAR LOGFILE option, 7-19
 - CLEAR UNARCHIVED LOGFILE option, 7-7
 - database partially available to users, 4-9
 - DATAFILE...OFFLINE DROP clause, 12-9
 - datafiles online or offline, 11-24, 12-10
 - default temporary tablespace, specifying, 2-24
 - DROP LOGFILE MEMBER option, 7-17
 - DROP LOGFILE option, 7-16
 - MOUNT clause, 4-9

- NOARCHIVELOG option, 8-5
- OPEN clause, 4-10
- READ ONLY clause, 4-10
- RENAME FILE clause, 12-13
- tempfiles online or offline, 11-24, 12-10
- UNRECOVERABLE DATAFILE option, 7-20
- ALTER FUNCTION statement
 - COMPILE clause, 21-25
- ALTER INDEX statement
 - COALESCE clause, 16-8
 - for maintaining partitioned indexes, 17-23 to 17-61
 - MONITORING USAGE clause, 16-21
- ALTER PACKAGE statement
 - COMPILE clause, 21-25
- ALTER PROCEDURE statement
 - COMPILE clause, 21-25
- ALTER PROFILE statement
 - altering resource limits, 24-21
- ALTER RESOURCE COST statement, 24-21
- ALTER ROLE statement
 - changing authorization method, 25-7
- ALTER ROLLBACK SEGMENT statement
 - bringing segments online, 13-22
 - changing storage parameters, 13-21
 - taking segment offline, 13-23
- ALTER SEQUENCE statement, 20-13
- ALTER SESSION statement
 - ADVISE clause, 32-10
 - CLOSE DATABASE LINK clause, 30-2
 - SET SQL_TRACE initialization parameter, 5-17
 - setting time zone, 2-28
 - system privilege, 30-2
- ALTER SYSTEM statement
 - ARCHIVE LOG ALL option, 8-9
 - ARCHIVE LOG option, 8-8
 - DISABLE DISTRIBUTED RECOVERY clause, 32-25
 - ENABLE DISTRIBUTED RECOVERY clause, 32-25
 - ENABLE RESTRICTED SESSION clause, 4-11
 - QUIESCE RESTRICTED, 4-15
 - RESUME clause, 4-17
 - SCOPE clause for SET, 2-47
 - SET RESOURCE_LIMIT option, 24-19
 - SET RESOURCE_MANAGER_PLAN, 27-25
 - SET SHARED_SERVERS initialization parameter, 5-10
 - setting initialization parameters, 2-47
 - SUSPEND clause, 4-17
 - SWITCH LOGFILE option, 7-18
 - to enable Database Resource Manager, 27-24
 - UNQUIESCE, 4-16
- ALTER TABLE
 - MODIFY DEFAULT ATTRIBUTES FOR PARTITION clause, 17-43
- ALTER TABLE statement
 - ADD (column) clause, 15-13
 - ALLOCATE EXTENT clause, 15-12
 - DEALLOCATE UNUSED clause, 15-13
 - DISABLE ALL TRIGGERS clause, 21-14
 - DISABLE integrity constraint clause, 21-18
 - DROP COLUMN clause, 15-14
 - DROP integrity constraint clause, 21-19
 - DROP UNUSED COLUMNS clause, 15-15
 - ENABLE ALL TRIGGERS clause, 21-13
 - ENABLE integrity constraint clause, 21-18
 - external tables, 15-38
 - for maintaining partitions, 17-23 to 17-61
 - MODIFY (column) clause, 15-13
 - MODIFY DEFAULT ATTRIBUTES clause, 17-43
 - modifying index-organized table attributes, 15-30
 - MOVE clause, 15-12, 15-30
 - reasons for use, 15-10
 - RENAME COLUMN clause, 15-14
 - SET UNUSED clause, 15-15
- ALTER TABLESPACE statement
 - ADD DATAFILE parameter, 11-11
 - ONLINE option, example, 11-23
 - READ ONLY option, 11-25
 - READ WRITE option, 11-27
 - RENAME DATAFILE clause, 12-11
 - taking datafiles/tempfiles online/offline, 11-23, 12-9
- ALTER TRIGGER statement
 - DISABLE clause, 21-14
 - ENABLE clause, 21-13
- ALTER USER privilege, 24-6

- ALTER USER statement
 - default roles, 25-21
 - GRANT CONNECT THROUGH clause, 24-16
 - REVOKE CONNECT THROUGH clause, 24-16
- ALTER VIEW statement
 - COMPILE clause, 21-25
- altering indexes, 16-19 to 16-21
- altering storage parameters, 15-11
- altering users, 24-6
- ANALYZE statement, 21-5
 - CASCADE clause, 21-7
 - corruption reporting, 22-5
 - ESTIMATE STATISTICS SAMPLE clause, 21-5
 - LIST CHAINED ROWS clause, 21-7
 - listing chained rows, 21-7
 - VALIDATE STRUCTURE clause, 21-6
 - VALIDATE STRUCTURE ONLINE clause, 21-7
 - validating structure, 22-4
- ANALYZE TABLE statement, 30-7
- analyzing redo log files, 9-1
- analyzing schema objects, 21-3 to 21-6
- analyzing tables
 - cost-based optimization, 30-7
- application administrators, 23-12
- application context, 23-4
- application developers
 - privileges for, 23-10
 - roles for, 23-11
- application development
 - constraints, 30-3
 - database links
 - controlling connections, 30-2
 - distributed databases, 30-1
 - analyzing execution plan, 30-9
 - controlling connections, 30-2
 - handling errors, 30-3
 - handling RPC errors, 30-11
 - managing distribution of data, 30-2
 - managing referential integrity, 30-3
 - optimizing distributed queries, 28-47
 - overview, 28-44
 - remote procedure calls, 28-46
 - tuning distributed queries, 30-3
 - tuning using collocated inline views, 30-4
 - using cost-based optimization, 30-5
 - using hints to tune queries, 30-8
 - distributing data, 30-2
 - referential integrity, 30-3
 - remote connections
 - terminating, 30-2
 - security for, 23-10
- applications
 - administrator, 1-4
 - errors
 - RAISE_APPLICATION_ERROR()
 - procedure, 30-11
 - applications administrator, 1-4
- AQ_ADMINISTRATOR_ROLE role, 25-6
- AQ_USER_ROLE role, 25-6
- ARCH process
 - specifying multiple processes, 8-19
- architecture
 - Optimal Flexible Architecture (OFA), 2-7
- ARCHIVE LOG option
 - ALTER SYSTEM statement, 8-8
- archive processes, 5-12
- ARCHIVE_LAG_TARGET initialization
 - parameter, 7-11
- archived redo logs, 8-2
 - archiving modes, 8-5
 - controlling destination availability state, 8-14
 - destination status, 8-12
 - destinations
 - mandatory, 8-16
 - minimum number of, 8-16
 - re-archiving to failed, 8-18
 - sample scenarios, 8-17
 - failed destinations and, 8-16
 - multiplexing, 8-10
 - normal transmission of, 8-14
 - specifying destinations for, 8-10
 - standby transmission of, 8-14
 - status information, 8-24
 - transmitting, 8-14
 - tuning, 8-19
- archived redo mandatory destinations
 - for archived redo logs, 8-16
- ARCHIVELOG mode, 8-3
 - advantages, 8-3
 - archiving, 8-2

- automatic archiving in, 8-3
- definition of, 8-3
- distributed databases, 8-4
- enabling, 8-5
- manual archiving in, 8-3
- running in, 8-3
- switching to, 8-5
- taking datafiles offline and online in, 12-9
- archivelog process (ARCn)
 - tracing, 8-21
- archiver, 5-12
- archiving
 - advantages, 8-2
 - automatic
 - disabling, 8-8
 - disabling at instance startup, 8-8
 - enabling after instance startup, 8-7
 - enabling at instance startup, 8-7
 - changing archiving mode, 8-5
 - controlling destination availability state, 8-14
 - controlling number of processes, 8-7
 - destination status, 8-12
 - destinations
 - failure, 8-16
 - disabling, 8-5, 8-8
 - disadvantages, 8-2
 - enabling, 8-5, 8-7
 - manual, 8-9
 - multiple ARCH processes, 8-19
 - privileges
 - disabling, 8-8
 - enabling, 8-6
 - for manual archiving, 8-9
 - setting initial mode, 8-5
 - to failed destinations, 8-18
 - trace, controlling, 8-21
 - tuning, 8-19
 - viewing information on, 8-24
- AUDIT statement
 - BY proxy clause, 26-13
 - schema objects, 26-12
 - statement auditing, 26-11
 - system privileges, 26-11
- audit trail, 26-15
 - archiving, 26-17
 - auditing changes to, 26-18
 - controlling size of, 26-15
 - creating and deleting, 26-19
 - deleting views, 26-20
 - dropping, 26-19
 - interpreting, 26-20
 - maximum size of, 26-16
 - protecting integrity of, 26-18
 - purging records from, 26-16
 - recording changes to, 26-18
 - reducing size of, 26-17
 - table that holds, 26-2
 - views on, 26-19
- AUDIT_FILE_DEST initialization parameter, 26-8
 - setting for OS auditing, 26-8
- AUDIT_SYS_OPERATIONS initialization parameter, 26-8
 - auditing SYS, 26-6
- AUDIT_TRAIL initialization parameter, 26-8
 - auditing SYS, 26-6
 - setting, 26-8
- auditing, 26-2
 - audit option levels, 26-9
 - audit trail records, 26-4
 - database links, 28-31
 - default options, 26-13
 - disabling default options, 26-15
 - disabling options, 26-8, 26-13, 26-14
 - disabling options versus auditing, 26-14
 - enabling options, 26-8
 - privileges for, 26-8
 - enabling options versus auditing, 26-10
 - fine-grained, 26-18
 - guidelines, 26-2
 - historical information, 26-4
 - information stored in OS file, 26-5
 - keeping information manageable, 26-3
 - managing the audit trail, 26-19
 - multi-tier environments, 26-13
 - operating-system audit trails, 26-2
 - policies for, 23-20
 - privilege audit options, 26-11
 - privileges required for object, 26-12
 - privileges required for system, 26-12
 - schema objects, 26-12

- session level, 26-11
- statement, 26-11
- statement level, 26-11
- suspicious activity, 26-3
- SYS, 26-6
- system privileges, 26-11
- to OS file, 26-8
- using the database, 26-2
- viewing
 - active object options, 26-22
 - active privilege options, 26-22
 - active statement options, 26-22
 - default object options, 26-23
- views, 26-19
- AUTHENTICATED BY clause
 - CREATE DATABASE LINK statement, 29-16
- authentication
 - by database, 24-9
 - by SSL, 24-8, 24-14
 - database links, 28-25
 - directory service, 24-14
 - external, 24-11
 - global, 24-13
 - operating system, 1-17
 - password policy, 23-5
 - proxy, 24-16
 - selecting a method, 1-15
 - specifying when creating a user, 24-3
 - users, 23-2
 - using password file, 1-19
 - ways to authenticate users, 24-8
- authorization
 - changing for roles, 25-7
 - global, 24-13
 - omitting for roles, 25-7
 - operating-system role management and, 25-9
 - roles, about, 25-8
- automatic undo management, 2-24

B

- background processes, 5-11 to 5-13
 - FMON, 12-17
- BACKGROUND_DUMP_DEST initialization
 - parameter, 5-16

- backups
 - after creating new databases, 2-22
 - guidelines, 1-7
 - effects of archiving on, 8-3
- BLANK_TRIMMING initialization
 - parameter, 15-13
- BLOCKSIZE clause
 - of CREATE TABLESPACE, 11-19
- broken jobs
 - about, 10-12
 - running, 10-13
- BUFFER_POOL storage parameter
 - description, 14-11
- buffers
 - buffer cache in SGA, 2-40

C

- calls
 - remote procedure, 28-46
- CASCADE clause
 - when dropping unique or primary keys, 21-19
- cascading revokes, 25-19
- CATAUDIT.SQL script
 - running, 26-19
- CATBLOCK.SQL script, 5-15
- CATNOAUD.SQL script
 - running, 26-20
- centralized user management
 - distributed systems, 28-27
- chained rows
 - eliminating from table, procedure, 21-8
- CHAINED_ROWS table
 - used by ANALYZE statement, 21-7
- change vectors, 7-2
- CHAR datatype
 - increasing column length, 15-13
- character sets
 - multibyte characters in role names, 25-7
 - multibyte characters in role passwords, 25-8
 - specifying when creating a database, 2-3
- CHECK_OBJECT procedure, 22-2, 22-4, 22-5
 - example, 22-10
- checkpoint process, 5-12
- checksums

- for data blocks, 12-14
 - redo log blocks, 7-18
- CJQ0 background process, 10-2
- CLEAR LOGFILE option
 - ALTER DATABASE statement, 7-19
- clearing redo log files, 7-7, 7-19
 - restrictions, 7-19
- client/server architectures
 - distributed databases, 28-6
 - direct and indirect connections, 28-7
 - Globalization Support, 28-47
- CLOSE DATABASE LINK clause
 - ALTER SESSION statement, 30-2
- closing database links, 29-19
- clustered tables. *See* clusters.
- clusters
 - allocating extents, 18-9
 - altering, 18-8
 - analyzing, 21-3 to 21-6
 - cluster indexes, 18-10
 - altering, 18-9
 - creating, 18-8
 - dropping, 18-11
 - cluster keys
 - columns for, 18-4
 - definition, 18-2
 - SIZE parameter, 18-5
 - clustered tables, 18-2, 18-4, 18-7, 18-11
 - ALTER TABLE restrictions, 18-9
 - columns for cluster key, 18-4
 - creating, 18-6
 - deallocating extents, 18-9
 - dropping, 18-10
 - estimating space, 18-5, 18-6
 - guidelines for managing, 18-4 to 18-6
 - hash
 - contrasted with index, 19-2
 - hash clusters, 19-1 to 19-9
 - index
 - contrasted with hash, 19-2
 - location, 18-6
 - overview of, 18-2
 - privileges
 - for altering, 18-8
 - for creating, 18-6
 - for dropping, 18-11
 - selecting tables, 18-4
 - single-table hash clusters, 19-5
 - specifying PCTFREE for, 14-4
 - truncating, 21-9
 - validating structure, 21-6
- COALESCE PARTITION clause, 17-32
- coalescing indexes
 - costs, 16-8
- collocated inline views
 - tuning distributed queries, 30-4
- columns
 - adding, 15-13
 - displaying information about, 21-34
 - dropping, 15-14 to 15-16
 - granting privileges for selected, 25-15
 - granting privileges on, 25-15
 - increasing length, 15-13
 - INSERT privilege and, 25-15
 - listing users granted to, 25-28
 - modifying definition, 15-13
 - privileges, 25-15
 - renaming, 15-14
 - revoking privileges on, 25-18
- COMMENT statement, 15-40
- COMMIT COMMENT statement
 - used with distributed transactions, 32-2, 32-10
- commit phase, 31-11, 31-23
 - two-phase commit, 31-14
- commit point site, 31-7
 - commit point strength, 31-8, 32-2
 - determining, 31-9
 - distributed transactions, 31-6, 31-8
 - how Oracle determines, 31-8
- commit point strength
 - definition, 31-8
 - specifying, 32-2
- COMMIT statement
 - FORCE clause, 32-10, 32-11, 32-12
 - forcing, 32-9
 - two-phase commit and, 28-36
- COMMIT_POINT_STRENGTH initialization
 - parameter, 31-8, 32-2
- committing transactions
 - distributed

- commit point site, 31-7
- composite limits
 - costs and, 24-22
- composite partitioned tables
 - subpartition template, modifying, 17-48
- composite partitioning
 - creating tables using, 17-14
 - default partition, 17-10
 - range-list, 17-8, 17-15
 - when to use, 17-7
- CONNECT command
 - starting an instance, 4-3
- CONNECT INTERNAL
 - desupported, 1-15
- CONNECT role, 25-5
- connected user database links, 29-12
 - advantages and disadvantages, 28-16
 - creating, 29-12
 - definition, 28-16
 - example, 28-19
 - REMOTE_OS_AUTHENT initialization parameter, 28-17
- connection qualifiers
 - database links and, 29-13
- connections
 - auditing, 26-11
 - remote
 - terminating, 30-2
- constraints
 - See also* integrity constraints
 - application development issues, 30-3
 - disabling at table creation, 21-17
 - dropping integrity constraints, 21-19
 - enable novalidate state, 21-16
 - enabling example, 21-17
 - enabling when violations exist, 21-16
 - exceptions, 21-16, 21-21
 - exceptions to integrity constraints, 21-21
 - integrity constraint states, 21-15
 - keeping index when disabling, 21-18
 - keeping index when dropping, 21-18
 - ORA-02055
 - constrain violation, 30-3
 - renaming, 21-19
 - setting at table creation, 21-17
 - when to disable, 21-15
- control files
 - adding, 6-5
 - changing size, 6-5
 - conflicts with data dictionary, 6-9
 - creating
 - about, 6-2
 - additional control files, 6-5
 - initially, 6-4
 - new files, 6-6
 - creating as Oracle-managed files, 3-17
 - default name, 2-36, 6-5
 - dropping, 6-11
 - errors during creation, 6-10
 - guidelines for, 6-2 to 6-4
 - importance of multiplexed, 6-3
 - location of, 6-3
 - log sequence numbers, 7-5
 - mirrored, 6-3
 - mirroring, 2-36
 - moving, 6-5
 - multiplexed
 - importance of, 6-3
 - names, 6-2
 - number of, 6-3
 - overwriting existing, 2-36
 - relocating, 6-5
 - renaming, 6-5
 - requirement of one, 6-2
 - size of, 6-4
 - specifying names before database creation, 2-36
 - troubleshooting, 6-9
 - unavailable during startup, 4-6
- CONTROL_FILES initialization parameter
 - overwriting existing control files, 2-36
 - setting
 - before database creation, 6-4
 - names for, 6-2
 - setting before database creation, 2-36
 - warning about setting, 2-36
- corruption
 - data block
 - repairing, 22-2 to 22-15
- cost-based optimization, 30-5
 - distributed databases, 28-47

- hints, 30-8
- using for distributed queries, 30-5
- costs
 - resource limits and, 24-22
- CREATE CLUSTER statement
 - creating clusters, 18-7
 - example, 18-7
 - for hash clusters, 19-4
 - HASH IS option, 19-4, 19-6
 - HASHKEYS option, 19-4, 19-7
 - SIZE option, 19-6
- CREATE CONTROLFILE statement
 - about, 6-6
 - checking for inconsistencies, 6-9
 - NORESETLOGS option, 6-8
 - RESETLOGS option, 6-8
- CREATE DATABASE LINK statement, 29-9
- CREATE DATABASE statement
 - CONTROLFILE REUSE option, 6-5
 - DEFAULT TEMPORARY TABLESPACE clause, 2-24
 - EXTENT MANAGEMENT LOCAL clause, 2-26
 - MAXLOGFILES option, 7-10
 - MAXLOGMEMBERS parameter, 7-10
 - password for SYS, 2-23
 - password for SYSTEM, 2-23
 - setting time zone, 2-28
 - specifying FORCE LOGGING, 2-29
 - UNDO TABLESPACE clause, 2-24
 - used to create an undo tablespace, 13-6
 - using Oracle-managed files, 3-8
- CREATE INDEX statement
 - NOLOGGING, 16-7
 - ON CLUSTER option, 18-8
 - partitioned indexes, 17-12 to 17-15
 - using, 16-10
 - with a constraint, 16-11
- CREATE PROFILE statement
 - about, 24-20
- CREATE ROLE statement
 - IDENTIFIED BY option, 25-8
 - IDENTIFIED EXTERNALLY option, 25-9
- CREATE ROLLBACK SEGMENT statement
 - about, 13-19
- CREATE SCHEMA statement
 - multiple tables and views, 21-2
- CREATE SEQUENCE statement, 20-12
- CREATE SPFILE statement, 2-46
- CREATE SYNONYM statement, 20-14
- CREATE TABLE statement
 - AS SELECT clause, 15-4, 15-8
 - CLUSTER option, 18-7
 - COMPRESS clause, 15-29
 - creating partitioned tables, 17-11 to 17-21
 - creating temporary table, 15-8
 - INCLUDING clause, 15-28
 - index-organized tables, 15-25
 - MONITORING clause, 15-9
 - NOLOGGING clause, 15-4
 - ORGANIZATION EXTERNAL clause, 15-35
 - OVERFLOW clause, 15-27
 - parallelizing, 15-8
 - PCTTHRESHOLD clause, 15-28
 - TABLESPACE clause, specifying, 15-3
 - use of, 15-7
- CREATE TABLESPACE
 - BLOCKSIZE CLAUSE, using, 11-19
 - FORCE LOGGING clause, using, 11-20
 - Oracle-managed files, 3-14
- CREATE TABLESPACE statement
 - example, 11-10
 - SEGMENT MANAGEMENT clause, 11-8
- CREATE TEMPORARY TABLESPACE
 - Oracle-managed files, 3-16
- CREATE TEMPORARY TABLESPACE statement, 11-13
- CREATE UNDO TABLESPACE
 - Oracle-managed files, 3-14
- CREATE UNDO TABLESPACE statement
 - using to create an undo tablespace, 13-6
- CREATE UNIQUE INDEX statement
 - using, 16-11
- CREATE USER statement
 - IDENTIFIED BY option, 24-3
 - IDENTIFIED EXTERNALLY option, 24-3
- CREATE VIEW statement
 - about, 20-2
 - OR REPLACE option, 20-11
 - WITH CHECK OPTION, 20-3
- CREATE_SIMPLE_PLAN procedure

- Database Resource Manager, 27-10
- creating a database
 - default temporary tablespace, specifying, 2-24
- creating an audit trail, 26-19
- creating connected user links
 - scenario, 29-35, 29-36
- creating current user links
 - scenario, 29-37
- creating database links, 29-8
 - connected user, 29-12
 - current user, 29-12
 - example, 28-19
 - fixed user, 29-11
 - private, 29-9
 - public, 29-10
 - service names within link names, 29-13
 - specifying types, 29-9
- creating databases, 2-1, 8-5
 - backing up the new database, 2-22
 - executing CREATE DATABASE, 2-18
 - manually from a script, 2-5
 - preparing to, 2-2
 - prerequisites for, 2-4
 - problems encountered while, 2-31
 - UNDO MANAGEMENT clause, 2-24
 - upgrading to a new release, 2-5
 - using Database Configuration Assistant, 2-5
 - with locally managed tablespaces, 2-26
- creating datafiles, 12-5
- creating fixed user links
 - scenario, 29-34, 29-35
- creating indexes
 - after inserting table data, 16-3
 - associated with integrity constraints, 16-11
 - NOLOGGING, 16-7
 - USING INDEX clause, 16-11
- creating profiles, 24-20
- creating sequences, 20-12
- creating synonyms, 20-14
- creating views, 20-2
- current user database links, 29-12
 - advantages and disadvantages, 28-18
 - cannot access in shared schema, 28-28
 - creating, 29-12
 - definition, 28-16

- example, 28-19
- schema independence, 28-28
- cursors
 - and closing database links, 30-2

D

- data
 - loading using external tables, 15-35
 - security of, 23-3
- data block corruption
 - repairing, 22-2 to 22-15
- data blocks
 - altering size of, 2-37
 - managing space in, 14-2 to 14-7
 - non-standard block size, 2-37
 - PCTFREE in clusters, 18-5
 - shared in clusters, 18-2
 - specifying size of, 2-37
 - standard block size, 2-37
 - transaction entry settings, 14-8
 - verifying, 12-14
- data dictionary
 - changing storage parameters, 21-30
 - changing storage parameters of, 21-27
 - conflicts with control files, 6-9
 - purging pending rows from, 32-13, 32-14
 - schema object views, 21-30
 - segments in the, 21-28
 - V\$DBFILE view, 2-31
 - V\$LOGFILE view, 2-31
- data dictionary views
 - DBA_DB_LINKS, 29-21, 32-3, 32-5
 - USER, 32-3, 32-5
- data encryption
 - distributed systems, 28-30
- data manipulation language
 - statements allowed in distributed transactions, 28-33
- database
 - granting privileges, 25-11
 - granting roles, 25-11
- database administrators, 1-2
 - application administrator versus, 23-12
 - initial priorities, 1-4 to 1-8

- operating system account, 1-10
- password files for, 1-16
- responsibilities of, 1-2
- roles
 - about, 1-13
 - for security, 23-9
- security and privileges of, 1-10
- security for, 23-8
- security officer versus, 1-3, 23-2
- SYS and SYSTEM accounts, 1-11
- utilities for, 1-26

database authentication, 24-9

Database Configuration Assistant

- advantages, 2-6
- configuring options, 2-9
- creating databases, 2-7 to 2-9
- defined, 2-5
- deleting databases, 2-9
- managing templates, 2-9
- templates, using, 2-11

database links

- advantages, 28-11
- auditing, 28-31
- authentication, 28-25
 - without passwords, 28-26
- closing, 29-19, 30-2
- connected user, 29-12, 29-35
 - advantages and disadvantages, 28-16
 - definition, 28-16
- connections
 - controlling, 30-2
 - determining open, 29-24
- creating, 29-8
 - connected user, 29-12, 29-35
 - connected user, shared, 29-36
 - current user, 29-12, 29-37
 - example, 28-19
 - fixed user, 29-11, 29-34
 - fixed user, shared, 29-35
 - obtaining necessary privileges, 29-8
 - private, 29-9
 - public, 29-10
 - scenarios, 29-34
 - shared, 29-14, 29-15
 - specifying types, 29-9
 - current user, 28-15, 29-12
 - advantages and disadvantages, 28-18
 - definition, 28-16
 - data dictionary views
 - ALL, 32-3, 32-5
 - DBA_DB_LINKS, 32-3, 32-5
 - USER, 29-21, 32-3, 32-5
 - definition, 28-8
 - distributed queries, 28-34
 - distributed transactions, 28-35
 - dropping, 29-19
 - enforcing global naming, 29-3
 - enterprise users and, 28-28
 - fixed user, 29-34
 - advantages and disadvantages, 28-17
 - definition, 28-16
 - global
 - definition, 28-15
 - global names, 28-12
 - global object names, 28-36
 - handling errors, 30-3
 - job queues and, 10-9
 - limiting number of connections, 29-20
 - listing, 29-21, 32-3, 32-5
 - managing, 29-18
 - minimizing network connections, 29-14
 - name resolution, 28-36
 - schema objects, 28-38
 - views, synonyms, and procedures, 28-42
 - when global database name is
 - complete, 28-37
 - when global database name is partial, 28-37
 - when no global database name is
 - specified, 28-37
 - names for, 28-14
 - passwords, viewing, 29-22
 - private
 - definition, 28-15
 - public
 - definition, 28-15
 - referential integrity in, 30-3
 - remote queries, 28-33
 - remote transactions, 28-33, 28-35
 - resolution, 28-36
 - restrictions, 28-22

- roles on remote database, 28-23
- schema objects, 28-20
 - name resolution, 28-22
 - synonyms for, 28-21
- service names used within link names, 29-13
- shared, 28-10
 - configuring, 29-16
 - creating, 29-14
 - creating links to dedicated servers, 29-16
 - creating links to shared servers, 29-17
 - determining whether to use, 29-14
- shared SQL, 28-34
- tuning distributed queries, 30-3
- tuning queries with hints, 30-8
- tuning using collocated inline views, 30-4
- types of links, 28-15
- types of users, 28-16
- users
 - specifying, 29-11
- using cost-based optimization, 30-5
- viewing, 29-21

Database Resource Manager

- active session pool with queuing, 27-6
- administering system privilege, 27-8 to 27-10
- automatic consumer group switching, 27-7
- CREATE_SIMPLE_PLAN procedure, 27-10
- description, 27-2
- enabling, 27-24
- execution time limit, 27-7
- managing resource consumer groups, 27-20
 - changing resource consumer groups, 27-21
 - granting the switch privilege, 27-21, 27-22
 - revoking the switch privilege, 27-23
 - setting initial resource consumer group, 27-21
 - switching a session, 27-21
 - switching sessions for a user, 27-22
- multiple level CPU resource allocation, 27-6
- pending area, 27-12 to 27-14
- resource allocation methods, 27-4
 - ACTIVE_SESS_POOL_MTH, 27-15
 - CPU resource, 27-14
 - EMPHASIS, 27-14
 - limiting degree of parallelism, 27-15
 - PARALLEL_DEGREE_LIMIT_
 - ABSOLUTE, 27-15
 - PARALLEL_DEGREE_LIMIT_MTH, 27-15
 - QUEUEING_MTH, 27-15
 - ROUND-ROBIN, 27-16
- resource consumer groups, 27-3
 - creating, 27-16 to 27-17
 - DEFAULT_CONSUMER_GROUP, 27-16, 27-17, 27-21, 27-23
 - deleting, 27-17
 - LOW_GROUP, 27-17, 27-29
 - managing, 27-20 to 27-23
 - OTHER_GROUPS, 27-6, 27-13, 27-16, 27-19, 27-28
 - parameters, 27-16
 - SYS_GROUP, 27-17, 27-28
 - updating, 27-17
- resource plan directives, 27-4, 27-12
 - deleting, 27-19
 - specifying, 27-17 to 27-20
 - updating, 27-19
- resource plans, 27-3
 - creating, 27-10 to 27-16
 - DELETE_PLAN_CASCADE, 27-16
 - deleting, 27-15
 - examples, 27-4, 27-25
 - parameters, 27-14
 - plan schemas, 27-6, 27-12, 27-16, 27-25, 27-32
 - subplans, 27-5, 27-6, 27-16
 - SYSTEM_PLAN, 27-15, 27-17, 27-28
 - top plan, 27-6, 27-13, 27-24
 - updating, 27-15
 - specifying a parallel degree limit, 27-7
 - undo pool, 27-7
 - used for quiescing a database, 4-15
 - validating plan schema changes, 27-12
 - views, 27-31
- database users
 - enrolling, 1-7
- database writer, 5-12
- database writer process
 - calculating checksums for data blocks, 12-14
- DATABASE_PROPERTIES view
 - name of default temporary tablespace, 2-25
- databases
 - administering, 1-1

- administration of distributed, 29-1
- altering availability, 4-9 to 4-11
- auditing, 26-1
- backing up, 2-22
 - after creation of, 1-7
- configuring options using DBCA, 2-9
- control files of, 6-2
- creating, 8-5
 - opening and, 1-6
- creating manually, 2-14 to 2-22
- creating using DBCA, 2-7
- default temporary tablespace, specifying, 2-24
- deleting using DBCA, 2-9
- design of
 - implementing, 1-7
- distributed
 - site autonomy of, 28-24
- dropping, 2-31
- global database name, about, 2-35
- global database names in distributed systems, 2-36
- hardware evaluation, 1-5
- logical structure of, 1-5
- mounting a database, 4-6
- mounting to an instance, 4-9
- names, about, 2-36
- names, conflicts in, 2-36
- opening a closed database, 4-10
- password encryption, 23-5
- physical structure, 1-6
- physical structure of, 1-6
- planning, 1-5
- production, 23-10, 23-12
- quiescing, 4-14
- read-only, opening, 4-10
- recovery, 4-8
- renaming, 6-6, 6-8
- restricting access, 4-11
- resuming, 4-17
- security. *See also* security.
- shutting down, 4-11 to 4-14
- specifying control files, 2-36
- starting up, 4-3 to 4-9
- structure of
 - distributed database, 1-6

- suspending, 4-17
- templates (DBCA), 2-9
- test, 23-10
- troubleshooting creation problems, 2-31
- tuning
 - archiving large databases, 8-19
 - responsibilities for, 1-8
- undo management, 2-24
- upgrading, 2-5
- user responsibilities, 1-4
- viewing datafiles and redo log files, 2-31
- with locally managed tablespaces, 2-26

datafiles

- adding to a tablespace, 12-5
- bringing online and offline, 12-8
- checking associated tablespaces, 11-51
- creating, 12-5
- creating as Oracle-managed files, 3-14
- database administrators access, 1-10
- default directory, 12-5
- definition, 12-2
- deleting, 11-29
- dropping, 12-9, 12-14
- dropping Oracle-managed files, 3-21
- file numbers, 12-2
- fully specifying filenames, 12-5
- guidelines for managing, 12-2 to 12-4
- identifying filenames, 12-12
- location, 12-4
- mapping files to physical devices, 12-15 to 12-27
- minimum number of, 12-2
- MISSING, 6-9
- monitoring using views, 12-28
- online, 12-9
- relocating, 12-10, 12-13
- relocating, example, 12-12
- renaming, 12-10, 12-13
- renaming for single tables, 12-11
- reusing, 12-5
- size of, 12-4
- statements to create, 12-5
- storing separately from redo log files, 12-4
- taking offline, 11-23
- unavailable when database is opened, 4-6
- V\$DBFILE and V\$LOGFILE views, 2-31

- verifying data blocks, 12-14
- DB_BLOCK_CHECKING initialization
 - parameter, 22-4, 22-5
- DB_BLOCK_CHECKSUM initialization
 - parameter, 12-14
 - enabling redo block checking with, 7-18
- DB_BLOCK_SIZE initialization parameter
 - setting, 2-37
- DB_CACHE_SIZE initialization parameter
 - setting, 2-40
- DB_CREATE_FILE_DEST initialization parameter
 - described, 3-5
- DB_CREATE_ONLINE_LOG_DEST_n initialization
 - parameter
 - described, 3-5
- DB_DOMAIN initialization parameter
 - setting before database creation, 2-35, 2-36
- DB_FILES initialization parameter, 12-3
- DB_NAME initialization parameter
 - setting before database creation, 2-35
- DB_nK_CACHE_SIZE initialization parameter
 - using with transportable tablespaces, 11-41
- DB_nK_CACHE_SIZE initialization parameters
 - setting, 2-40
- DB_VERIFY utility, 22-4, 22-5
- DBA role, 1-13, 25-5
- DBA. *See* database administrators.
- DBA_2PC_NEIGHBORS view, 32-5
 - using to trace session tree, 32-6
- DBA_2PC_PENDING view, 32-3, 32-13, 32-22
 - using to list in-doubt transactions, 32-3
- DBA_DATA_FILES view, 11-50
- DBA_DB_LINKS view, 29-21, 32-3, 32-5
- DBA_JOBS view
 - jobs in system, viewing, 10-15
- DBA_JOBS_RUNNING
 - running jobs, viewing, 10-15
- DBA_RESUMABLE view, 14-21
- DBA_ROLLBACK_SEGS view, 13-25, 13-26
- DBA_SEGMENTS view, 11-50
- DBA_TEMP_FILES view, 11-50
- DBA_TS_QUOTAS view, 11-50
- DBA_UNDO_EXTENTS view
 - undo tablespace extents, 13-12
- DBA_USERS view, 11-50
- DBCA. *See* Database Configuration Assistant
- DBMS_FLASHBACK package
 - setting undo retention period for, 13-10
- DBMS_JOB package, 10-3
- DBMS_LOGMNR.D.BUILD procedure, 9-6
- DBMS_METADATA package
 - GET_DDL function, 21-31
 - using for object definition, 21-31
- DBMS_REDEFINITION package
 - redefining tables online, 15-17
- DBMS_REPAIR package, 22-2 to 22-15
 - CHECK_OBJECT procedure, 22-2
 - DUMP_ORPHAN_KEYS procedure, 22-3
 - examples, 22-8 to 22-15
 - limitations, 22-3
 - procedures, 22-2
 - SEGMENT_FIX_STATUS procedure, 22-3
 - SKIP_CORRUPT_BLOCKS procedure, 22-3
 - using, 22-3 to 22-8
- DBMS_REPAIR procedure
 - FIX_CORRUPT_BLOCKS procedure, 22-2
 - REBUILD_FREELISTS procedure, 22-3
- DBMS_RESOURCE_MANAGER package, 27-4, 27-9, 27-20, 27-21
 - procedures (table of), 27-8
- DBMS_RESOURCE_MANAGER_PRIVS
 - package, 27-9, 27-20
 - procedures (table of), 27-9
- DBMS_RESUMABLE package, 14-22
- DBMS_SESSION package, 27-23
- DBMS_SPACE package, 14-25
 - example for unused space, 21-32
 - FREE_BLOCK procedure, 21-31
 - SPACE_USAGE procedure, 21-31
 - UNUSED_SPACE procedure, 21-31
- DBMS_SPACE_ADMIN package, 11-30 to 11-33
- DBMS_STATS package, 21-4
 - MONITORING clause of CREATE TABLE, 15-9
- DBMS_STORAGE_MAP package
 - invoking for file mapping, 12-22
- DBMS_STORAGE_MAPPING package, 12-22, 12-23
- DBMS_TRANSACTION package
 - PURGE_LOST_DB_ENTRY procedure, 32-13
- DBMS_UTILITY package

- ANALYZE_SCHEMA procedure
 - used for computing statistics, 21-6
- DEALLOCATE UNUSED clause, 14-26
- deallocating unused space, 14-25
 - DBMS_SPACE package, 14-25
 - DEALLOCATE UNUSED clause, 14-26
 - examples, 14-26
 - high water mark, 14-25
- declarative referential integrity constraints, 30-3
- dedicated server processes, 5-2
 - trace files for, 5-15
- default
 - audit options, 26-13
 - disabling, 26-15
- DEFAULT keyword
 - list partitioning, 17-13
- default partitions, 17-7
- default roles, 25-21
- default subpartition, 17-10
- DEFAULT_CONSUMER_GROUP for Database Resource Manager, 27-16, 27-17, 27-21, 27-23
- defaults
 - profile, 24-20
 - role, 24-7
 - tablespace quota, 24-4
 - user tablespaces, 24-3
- DELETE_CATALOG_ROLE role, 25-6
- DELETE_CATALOG_ROLE roll, 25-4
- dependencies
 - displaying, 21-34
- destinations
 - archived redo logs
 - optional, 8-16
 - sample scenarios, 8-17
- developers, application, 23-10
- dictionary protection mechanism, 25-2
- dictionary-managed tablespaces, 11-10 to 11-12
 - migrating SYSTEM to locally managed, 11-34
- Digital's POLYCENTER Manager on NetView, 28-32
- directory service
 - See also* enterprise directory service.
- DISABLE ROW MOVEMENT clause, 17-10
- disabling audit options, 26-13, 26-14
- disabling auditing, 26-8
- disabling recoverer process
 - distributed transactions, 32-25
- disabling resource limits, 24-19
- disconnections
 - auditing, 26-11
- dispatcher processes, 5-6, 5-10, 5-13
- DISPATCHERS initialization parameter
 - setting initially, 5-6
- distributed applications
 - distributing data, 30-2
- distributed databases
 - administration
 - overview, 28-23
 - application development
 - analyzing execution plan, 30-9
 - controlling connections, 30-2
 - handling errors, 30-3
 - handling RPC errors, 30-11
 - managing distribution of data, 30-2
 - managing referential integrity, 30-3
 - tuning distributed queries, 30-3
 - tuning using collocated inline views, 30-4
 - using cost-based optimization, 30-5
 - using hints to tune queries, 30-8
 - client/server architectures, 28-6
 - commit point strength, 31-8
 - cost-based optimization, 28-47
 - distributed processing, 28-3
 - distributed queries, 28-34
 - distributed updates, 28-34
 - distributing an application's data, 30-2
 - global database names
 - how they are formed, 29-2
 - global object names, 28-22, 29-2
 - global users
 - schema-dependent, 28-27
 - schema-independent, 28-28
- Globalization Support, 28-47
- location transparency, 28-44
 - creating, 29-26
 - creating using procedures, 29-30
 - creating using synonyms, 29-28
 - creating using views, 29-26
 - restrictions, 29-33
- management tools, 28-31

- managing read consistency, 32-25
- nodes of, 28-6
- overview, 28-2
- referential integrity
 - application development, 30-3
- remote object security, 29-28
- remote queries and updates, 28-33
- replicated databases and, 28-4
- resumable space allocation, 14-18
- running in ARCHIVELOG mode, 8-4
- running in NOARCHIVELOG mode, 8-4
- scenarios, 29-34
- security, 28-24
- site autonomy, 28-24
- SQL transparency, 28-45
- starting a remote instance, 4-9
- transaction processing, 28-33
- transparency, 28-44
 - queries, 29-32
 - updates, 29-32
- distributed processing
 - distributed databases, 28-3
- distributed queries, 28-34
 - analyzing tables, 30-7
 - application development issues, 30-3
 - cost-based optimization, 30-5
 - optimizing, 28-47
- distributed systems
 - data encryption, 28-30
- distributed transactions, 28-35
 - case study, 31-19
 - commit point site, 31-7
 - commit point strength, 31-8
 - committing, 31-8
 - database server role, 31-5
 - defined, 31-2
 - DML and DDL, 31-3
 - failure during, 32-23
 - global coordinator, 31-6
 - local coordinator, 31-6
 - lock timeout interval, 32-23
 - locked resources, 32-23
 - locks for in-doubt, 32-24
 - management, 31-1, 32-1
 - manually overriding in-doubt, 32-9
 - naming, 32-2, 32-10
 - recovery in single-process systems, 32-25
 - session trees, 31-4
 - clients, 31-5
 - commit point site, 31-6, 31-8
 - database servers, 31-5
 - global coordinators, 31-6
 - local coordinators, 31-6
 - setting advice, 32-10
 - specifying
 - commit point strength, 32-2
 - tracing session tree, 32-5
 - transaction control statements, 31-4
 - transaction timeouts, 32-23
 - two-phase commit, 31-10
 - discovering problems, 32-8
 - example, 31-19
 - viewing information about, 32-3
- distributed updates, 28-34
- distributing I/O, 2-2
- DML. *See* data manipulation language
- DRIVING_SITE hint, 30-9
- DROP CLUSTER statement
 - CASCADE CONSTRAINTS option, 18-10
 - dropping cluster, 18-10
 - dropping cluster index, 18-10
 - dropping hash cluster, 19-9
 - INCLUDING TABLES option, 18-10
- DROP LOGFILE MEMBER option
 - ALTER DATABASE statement, 7-17
- DROP LOGFILE option
 - ALTER DATABASE statement, 7-16
- DROP PARTITION clause, 17-32
- DROP PROFILE statement, 24-23
- DROP ROLE statement, 25-10, 25-11
- DROP ROLLBACK SEGMENT statement, 13-25
- DROP SYNONYM statement, 20-15
- DROP TABLE statement
 - about, 15-23
 - CASCADE CONSTRAINTS option, 15-23
 - for clustered tables, 18-11
- DROP TABLESPACE statement, 11-30
- DROP USER privilege, 24-8
- DROP USER statement, 24-8
- dropping an audit trail, 26-19

- dropping columns from tables, 15-14
 - marking unused, 15-15
 - remove unused columns, 15-15
- dropping database links, 29-19
- dropping datafiles
 - Oracle managed, 3-21
- dropping profiles, 24-23
- dropping tables
 - CASCADE clause, 15-23
 - consequences of, 15-23
 - privileges, 15-23
- dropping tempfiles
 - Oracle managed, 3-21
- dropping users, 24-7
- DUMP_ORPHAN_KEYS procedure, 22-3, 22-6, 22-7
 - example, 22-13

E

- EMPHASIS resource allocation method, 27-14
- ENABLE ROW MOVEMENT clause, 17-10, 17-11
- enabling recoverer process
 - distributed transactions, 32-25
- enabling resource limits, 24-19
- encryption
 - database passwords, 23-5, 24-9
- enterprise directory service, 23-8, 25-10
- enterprise roles, 23-8, 24-14, 25-10
- enterprise users, 23-8, 24-14, 25-10
 - definition, 28-28
- errors
 - alert log and, 5-15
 - messages
 - trapping, 30-11
 - ORA-00028, 5-22
 - ORA-00900, 30-11
 - ORA-01090, 4-11
 - ORA-01173, 6-10
 - ORA-01176, 6-10
 - ORA-01177, 6-10
 - ORA-01578, 12-14
 - ORA-01591, 32-24
 - ORA-02015, 30-11
 - ORA-02049, 32-23

- ORA-02050, 32-8
- ORA-02051, 32-8
- ORA-02054, 32-8
- ORA-02055
 - integrity constrain violation, 30-3
- ORA-02067
 - rollback required, 30-3
- ORA-06510
 - PL/SQL error, 30-12
- ORA-1215, 6-10
- ORA-1216, 6-10
- ORA-1547, 21-30
- ORA-1628 through 1630, 21-30
- remote procedures, 30-11
- snapshot too old, 13-9, 13-18
- trace files and, 5-15
 - when creating a database, 2-31
 - when creating control file, 6-10
 - while starting a database, 4-8
 - while starting an instance, 4-8
- exception handler, 30-11
 - local, 30-12
- EXCEPTION keyword, 30-11
- exceptions
 - assigning names
 - PRAGMA_EXCEPTION_INIT, 30-12
 - integrity constraints, 21-21
 - user-defined
 - PL/SQL, 30-11
- EXCHANGE PARTITION clause, 17-37
- EXCHANGE SUBPARTITION clause, 17-37
- EXECUTE_CATALOG_ROLE role, 25-6
- EXECUTE_CATALOG_ROLE roll, 25-3
- executing jobs
 - enabling processes for, 10-2
- execution plans
 - analyzing for distributed queries, 30-9
- EXP_FULL_DATABASE role, 25-5
- Export utility
 - about, 1-26
 - restricted mode and, 4-7
- exporting jobs, 10-6
- EXTENT MANAGEMENT LOCAL clause
 - CREATE DATABASE, 2-26
- extents

- allocating
 - clusters, 18-9
- allocating for tables, 15-12
- data dictionary views for, 21-33
- deallocating
 - clusters, 18-9
- displaying free extents, 21-36
- displaying information on, 21-35
- external authentication
 - by network, 24-13
 - by operating system, 24-12
- external procedures
 - managing processes for, 5-20
- external tables
 - altering, 15-38
 - creating, 15-35
 - defined, 15-33
 - dropping, 15-39
 - object privileges, 15-39
 - object privileges for directory, 15-39
 - system privileges, 15-39
 - uploading data example, 15-35

F

- failures
 - media
 - multiplexed online redo logs, 7-5
- features, new, xviii to lxi
- file mapping
 - examples, 12-25
 - how it works, 12-16
 - how to use, 12-21
 - overview, 12-16
 - structures, 12-18
 - views, 12-23
- file system
 - used for Oracle-managed files, 3-3
- FILE_MAPPING initialization parameter, 12-22
- filenames
 - Oracle-managed files, 3-7
- files
 - Oracle-managed, 3-1 to 3-28
- fine-grained access control, 23-4
- fine-grained auditing, 26-18

- FIX_CORRUPT_BLOCKS procedure, 22-2, 22-7
 - example, 22-12
- fixed user database links
 - 07_DICTIONARY_ACCESSIBILITY initialization parameter, 28-18
 - advantages and disadvantages, 28-17
 - creating, 29-11
 - definition, 28-16
 - example, 28-20
- flashback query
 - setting retention period for, 13-10
- FMON background process, 12-17
- FMPUTL external process
 - used for file mapping, 12-18
- FOR PARTITION clause, 17-44
- FORCE clause
 - COMMIT statement, 32-10
 - ROLLBACK statement, 32-10
- FORCE LOGGING clause
 - CREATE CONTROLFILE, 2-30
 - CREATE DATABASE, 2-29
 - CREATE TABLESPACE, 11-20
 - performance considerations, 2-30
- forcing
 - COMMIT or ROLLBACK, 32-4, 32-9
- forcing a log switch, 7-18
 - using ARCHIVE_LAG_TIME, 7-10
 - with the ALTER SYSTEM statement, 7-18
- forget phase
 - two-phase commit, 31-15
- free space
 - coalescing, 11-16
 - listing free extents, 21-36
 - tablespaces and, 11-51
- FREELIST GROUPS storage parameter
 - description, 14-11
- FREELISTS GROUPS parameter, 11-8, 11-9
- FREELISTS parameter, 11-8, 11-9
- FREELISTS storage parameter
 - description, 14-11
- function-based indexes, 16-14 to 16-18
- functions
 - recompiling, 21-25

G

- generic connectivity
 - definition, 28-6
- global authentication and authorization, 24-13
- global cache service, 5-13
- global coordinators, 31-6
 - distributed transactions, 31-6
- global database consistency
 - distributed databases and, 31-14
- global database links, 28-15
 - creating, 29-11
- global database name, 2-35
- global database names
 - changing the domain, 29-4
 - database links, 28-12
 - distributed databases
 - how they are formed, 29-2
 - enforcing for database links, 28-14
 - enforcing global naming, 29-3
 - impact of changing, 28-42
 - querying, 29-4
- global object names
 - database links, 28-36
 - distributed databases, 29-2
- global roles, 24-13, 25-10
- global users, 24-13, 29-37
 - distributed systems
 - schema-dependent, 28-27
 - schema-independent, 28-28
- GLOBAL_NAME view
 - using to determine global database name, 29-4
- GLOBAL_NAMES initialization parameter, 28-14
- Globalization Support
 - client/server architectures, 28-48
 - distributed databases
 - clients and servers may diverge, 28-47
 - heterogeneous systems, 28-49
 - homogeneous systems, 28-48
- GRANT ANY OBJECT PRIVILEGE system
 - privilege, 25-14, 25-17
- GRANT CONNECT THROUGH clause
 - for proxy authorization, 24-16
- GRANT statement, 25-11
 - ADMIN OPTION, 25-12

- creating a new user, 25-12
- object privileges, 25-12
- SYSOPER/SYSDBA privileges, 1-24
- system privileges and roles, 25-11
 - when takes effect, 25-20
 - WITH GRANT OPTION, 25-13
- granting privileges and roles
 - listing grants, 25-26
 - SYSOPER/SYSDBA privileges, 1-24
- GV\$DBLINK view, 29-25

H

- hardware
 - evaluating, 1-5
- hash clusters
 - advantages and disadvantages, 19-2 to 19-3
 - altering, 19-9
 - choosing key, 19-5
 - contrasted with index clusters, 19-2
 - controlling space use of, 19-5
 - creating, 19-4
 - dropping, 19-9
 - estimating storage, 19-8
 - examples, 19-7
 - hash function, 19-2, 19-3, 19-4, 19-6
 - HASH IS option, 19-4, 19-6
 - HASHKEYS option, 19-4, 19-7
 - single-table, 19-5
 - SIZE option, 19-6
- hash functions
 - for hash cluster, 19-2
- hash partitioning
 - creating tables using, 17-12
 - index-organized tables, 17-21
 - when to use, 17-5
- heterogeneous distributed systems
 - definition, 28-5
- Heterogeneous Services
 - overview, 28-5
- high water mark, 14-25
- hints, 30-8
 - DRIVING_SITE, 30-9
 - NO_MERGE, 30-8
 - using to tune distributed queries, 30-8

historical tables
 moving time window, 17-61
HP's OpenView, 28-32
HS_ADMIN_ROLE role, 25-6

I

IBM's NetView/6000, 28-32
IMP_FULL_DATABASE role, 25-5
implementing database design, 1-7
Import utility
 about, 1-26
 restricted mode and, 4-7
importing jobs, 10-6
index clusters. *See* clusters.
indexes
 altering, 16-19 to 16-21
 analyzing, 21-3 to 21-6
 choosing columns to index, 16-4
 cluster indexes, 18-8, 18-9, 18-10
 coalescing, 16-8, 16-21
 column order for performance, 16-5
 creating, 16-9 to 16-19
 disabling and dropping constraints cost, 16-9
 dropping, 16-5, 16-22
 estimating size, 16-6
 explicitly creating a unique index, 16-11
 function-based, 16-14 to 16-18
 guidelines for managing, 16-2 to 16-9
 keeping when disabling constraint, 21-18
 keeping when dropping constraint, 21-18
 key compression, 16-18
 limiting per table, 16-5
 monitoring space use of, 16-21
 monitoring usage, 16-21
 parallelizing index creation, 16-7
 partitioned, 17-2
 see also partitioned indexes
 PCTFREE for, 16-5
 PCTUSED for, 16-5
 privileges
 for altering, 16-19
 for dropping, 16-22
 rebuilding, 16-8, 16-20
 rebuilding online, 16-20

 separating from a table, 15-6
 setting storage parameters for, 16-6
 space used by, 16-21
 specifying PCTFREE for, 14-4
 statement for creating, 16-10
 tablespace for, 16-6
 temporary segments and, 16-3
 updating global indexes, 17-26
 validating structure, 21-6
 when to create, 16-4
index-organized tables
 analyzing, 15-32
 AS subquery, 15-27
 converting to heap, 15-33
 creating, 15-25
 described, 15-24
 hash partitioning, 17-21
 INCLUDING clause, 15-28
 key compression, 15-29
 maintaining, 15-30
 ORDER BY clause, using, 15-33
 overflow clause, 15-27
 partitioning, 17-10, 17-19 to 17-21
 partitioning secondary indexes, 17-20
 range partitioning, 17-20
 rebuilding with MOVE clause, 15-30
 threshold value, 15-28
 updating key column, 15-31
in-doubt transaction
 manually committing, example, 32-15
in-doubt transactions, 31-14
 after a system failure, 32-8
 automatic resolution, 31-16
 failure during commit phase, 31-17
 failure during prepare phase, 31-16
 deciding how to handle, 32-7
 deciding whether to perform manual
 override, 32-9
 manually committing, 32-11
 manually overriding, 31-18, 32-10
 scenario, 32-15
 manually rolling back, 32-12
 overriding manually, 32-9
 overview, 31-15
 pending transactions table, 32-22

- purging rows from data dictionary, 32-13
 - deciding when necessary, 32-14
- recoverer process, 32-24
- rollback segments, 32-9
- rolling back, 32-11, 32-12, 32-13
- SCNs and, 31-19
- simulating, 32-24
- tracing session tree, 32-5
- viewing information about, 32-3

INITIAL storage parameter

- cannot alter, 14-13, 15-12
- description, 14-10
- rollback segments, 13-18, 13-20
- when deallocating unused space, 14-26

initialization parameter file

- creating, 2-15
- creating for database creation, 2-15
- editing before database creation, 2-35
- individual parameter names, 2-35
- server parameter file, 2-44 to 2-51, 4-4

initialization parameters

- ARCHIVE_LAG_TARGET, 7-11
- buffer cache, 2-40
- DB_BLOCK_CHECKSUM, 7-18
- DB_CREATE_FILE_DEST, 3-5
- DB_CREATE_ONLINE_LOG_DEST_*n*, 3-5
- FILE_MAPPING, 12-22
- LOG_ARCHIVE_DEST_*n*, 8-10
- LOG_ARCHIVE_DEST_STATE_*n*, 8-14
- LOG_ARCHIVE_MAX_PROCESSES, 8-7, 8-19
- LOG_ARCHIVE_MIN_SUCCEED_DEST, 8-16
- LOG_ARCHIVE_START, 8-7, 8-8, 8-13
- LOG_ARCHIVE_TRACE, 8-21
- MAX_ROLLBACK_SEGMENTS, 13-15
- RESOURCE_MANAGER_PLAN, 27-24
- ROLLBACK_SEGMENTS, 13-15
- shared server and, 5-5
- SPFILE, 2-47
- TRANSACTIONS, 13-15
- TRANSACTIONS_PER_ROLLBACK_SEGMENT, 13-15
- UNDO_MANAGEMENT, 2-24, 13-3
- UNDO_RETENTION, 13-9
- UNDO_SUPPRESS_ERROR, 13-4
- UNDO_TABLESPACE, 13-3

INITRANS storage parameter

- altering, 15-11
- guidelines for setting, 14-8

INSERT privilege

- granting, 25-15
- revoking, 25-18

installation

- Oracle9i, 1-5

instances

- aborting, 4-13
- shutting down immediately, 4-12
- shutting down normally, 4-12
- starting up, 4-2 to 4-9
- transactional shutdown, 4-13

integrity constraints

- See also* constraints
- cost of disabling, 16-9
- cost of dropping, 16-9
- creating indexes associated with, 16-11
- dropping tablespaces and, 11-30
- ORA-02055
 - constraint violation, 30-3

INTERNAL

- security for, 23-8

INTERNAL username

- connecting for shutdown, 4-11

I/O

- distributing, 2-2

IOT. *See* index-organized tables.

J

Jnnn processes

- managing job queues, 10-3 to 10-14

job queues

- altering jobs, 10-11
- broken jobs, 10-12
- CJQ background process, 10-2
- DBMS_JOB package, 10-3
- executing jobs in, 10-9
- Jnnn* processes, 10-2
- locks, 10-9
- removing jobs from, 10-10
- submitting jobs to, 10-4 to 10-9
- terminating jobs, 10-14

- viewing information, 10-15
- JOB_QUEUE_PROCESSES initialization
 - parameter, 10-2
- jobs
 - altering, 10-11
 - broken, 10-12
 - database links and, 10-9
 - environment, recording when submitted, 10-6
 - executing, 10-9
 - exporting, 10-6
 - forcing to execute, 10-14
 - importing, 10-6
 - job definition, 10-7
 - job execution interval, 10-8
 - job number, 10-7
 - ownership of, 10-7
 - removing from job queue, 10-10
 - running broken jobs, 10-13
 - submitting to job queue, 10-4
 - terminating, 10-14
 - trace files for job failures, 10-10
 - troubleshooting, 10-10
- join views
 - definition, 20-3
 - DELETE statements, 20-8
 - key-preserved tables in, 20-6
 - modifying, 20-5
 - rule for, 20-7
 - updating, 20-5
- joins
 - distributed databases
 - managing statement transparency, 29-33
- JQ locks, 10-9

K

- key compression, 15-29
 - indexes, 16-18
- key-preserved tables
 - in join views, 20-6
- keys
 - cluster, 18-2, 18-4, 18-5

L

- LIST CHAINED ROWS clause
 - of ANALYZE statement, 21-7
- list partitioning
 - adding values to value list, 17-45
 - creating tables using, 17-13
 - DEFAULT keyword, 17-13
 - dropping values from value-list, 17-46
 - when to use, 17-5
- listing database links, 29-21, 32-3, 32-5
- loading data
 - using external tables, 15-35
- LOBs
 - storage parameters for, 14-12
- local coordinators, 31-6
 - distributed transactions, 31-6
- locally managed tablespaces, 11-5 to 11-10
 - automatic segment space management, 11-8
 - DBMS_SPACE_ADMIN package, 11-30
 - detecting and repairing defects, 11-30
 - migrating SYSTEM from
 - dictionary-managed, 11-34
 - tempfiles, 11-13
 - temporary, creating, 11-13
- location transparency
 - distributed databases
 - creating using procedures, 29-30
 - creating using synonyms, 29-28
 - creating using views, 29-26
 - using procedures, 29-30, 29-31, 29-32
- lock timeout interval
 - distributed transactions, 32-23
- locks
 - in-doubt distributed transactions, 32-23, 32-24
 - job queue, 10-9
 - monitoring, 5-15
- log sequence number
 - control files, 7-5
- log switches
 - description, 7-5
 - forcing, 7-18
 - log sequence numbers, 7-5
 - multiplexed redo log files and, 7-7
 - privileges, 7-18

- using ARCHIVE_LAG_TIME, 7-10
 - waiting for archiving to complete, 7-7
- log writer process (LGWR), 5-12
 - multiplexed redo log files and, 7-6
 - online redo logs available for use, 7-3
 - trace file monitoring, 5-16
 - trace files and, 7-6
 - writing to online redo log files, 7-3
- LOG_ARCHIVE_DEST initialization parameter
 - specifying destinations using, 8-10
- LOG_ARCHIVE_DEST_n initialization parameter, 8-10
 - REOPEN option, 8-18
- LOG_ARCHIVE_DEST_STATE_n initialization parameter, 8-14
- LOG_ARCHIVE_DUPLEX_DEST initialization parameter
 - specifying destinations using, 8-10
- LOG_ARCHIVE_MAX_PROCESSES initialization parameter, 8-7, 8-19
- LOG_ARCHIVE_MIN_SUCCEED_DEST initialization parameter, 8-16
- LOG_ARCHIVE_START initialization parameter, 8-7, 8-13
 - setting, 8-8
- LOG_ARCHIVE_TRACE initialization parameter, 8-21
- LOGGING clause
 - CREATE TABLESPACE, 11-20
- logical structure of a database, 1-5
- logical volume manager
 - used for Oracle-managed files, 3-2
- logical volume managers
 - mapping files to physical devices, 12-15 to 12-27
- LogMiner
 - continuous mining, 9-25
 - formatting returned data, 9-17
- LogMiner utility
 - analyzing output, 9-16
 - dbmslmd.sql script, 9-6
 - dictionary options, 9-5
 - ending a session, 9-28
 - executing reconstructed SQL, 9-17
 - extracting a dictionary file, 9-6
 - extracting data values from redo logs, 9-18

- graphical user interface, 9-1
 - re-creating LogMiner tables in alternate tablespace, 9-11
 - redo log files, 9-4
 - specifying redo logs for analysis, 9-24
 - starting, 9-26
 - steps in a typical session, 9-23
 - supplemental logging, 9-19
 - identification keys, 9-20
 - log groups, 9-22
 - suppressing delimiters in SQL_REDO and SQL_UNDO, 9-17
 - tracking DDL statements, 9-9
 - using the online catalog, 9-8
 - using to analyze redo log files, 9-1
 - VSLOGMNR_CONTENTS view, 9-16
 - views, 9-15
- LogMiner Viewer, 9-1
- LOGON trigger
 - setting resumable mode, 14-20
- LONG columns, 29-33
- LONG RAW columns, 29-33
- LOW_GROUP for Database Resource Manager, 27-17, 27-29

M

- managing datafiles, 12-1 to 12-29
- managing job queues, 10-3 to 10-14
- managing roles, 25-6
- managing sequences, 20-11 to 20-13
- managing synonyms, 20-13 to 20-15
- managing tables, 15-1 to 15-41
- managing views, 20-2 to 20-11
- manual archiving
 - in ARCHIVELOG mode, 8-9
- manual overrides
 - in-doubt transactions, 32-10
- MAX_DUMP_FILE_SIZE initialization parameter, 5-16
- MAX_ENABLED_ROLES initialization parameter
 - enabling roles and, 25-22
- MAX_ROLLBACK_SEGMENTS initialization parameter, 13-15
- MAXDATAFILES parameter

- changing, 6-6
- MAXEXTENTS storage parameter
 - description, 14-10
 - rollback segments, 13-17, 13-20
 - setting for the data dictionary, 21-28
- MAXINSTANCES parameter
 - changing, 6-6
- MAXLOGFILES option
 - CREATE DATABASE statement, 7-10
- MAXLOGFILES parameter
 - changing, 6-6
- MAXLOGHISTORY parameter
 - changing, 6-6
- MAXLOGMEMBERS parameter
 - changing, 6-6
 - CREATE DATABASE statement, 7-10
- MAXTRANS storage parameter
 - altering, 15-11
 - guidelines for setting, 14-8
- media recovery
 - effects of archiving on, 8-3
- memory
 - viewing per user, 24-26
- MERGE PARTITIONS clause, 17-38
- messages
 - error
 - trapping, 30-11
- migrated rows
 - eliminating from table, procedure, 21-8
- MINEXTENTS storage parameter
 - cannot alter, 14-13, 15-12
 - deallocating unused space, 14-26
 - description, 14-10
 - rollback segments, 13-18, 13-20
- mirrored control files, 6-3
- mirrored files
 - online redo log, 7-6
 - location, 7-9
 - size, 7-9
- mirroring
 - control files, 2-36
- MISSING datafiles, 6-9
- MODIFY DEFAULT ATTRIBUTES clause, 17-44
 - using for partitioned tables, 17-43
- MODIFY DEFAULT ATTRIBUTES FOR

- PARTITION clause
 - of ALTER TABLE, 17-43
- MODIFY PARTITION clause, 17-44, 17-48, 17-51
- MODIFY SUBPARTITION clause, 17-45
- MONITORING clause
 - CREATE TABLE, 15-9
 - monitoring datafiles, 12-28
- MONITORING USAGE clause
 - of ALTER INDEX statement, 16-21
- MOUNT option
 - STARTUP command, 4-7
 - mounting a database, 4-6
- MOVE PARTITION clause, 17-44, 17-48
- MOVE SUBPARTITION clause, 17-44, 17-49
- moving control files, 6-5
- multiplexed control files
 - importance of, 6-3
- multiplexing
 - archived redo logs, 8-10
 - control files, 6-3
 - redo log files, 7-5
 - groups, 7-6
- multi-tier environments
 - auditing clients, 26-13

N

- name resolution
 - distributed databases, 28-22
 - impact of global name changes, 28-42
 - schema objects, 28-38
 - when global database name is
 - complete, 28-37
 - when global database name is partial, 28-37
 - when no global database name is
 - specified, 28-37
- named user limits
 - setting initially, 2-43
- nested tables
 - storage parameters for, 14-12
- network
 - authentication, 24-13
- network authentication, 24-13
- network connections
 - minimizing, 29-14

- networks
 - distributed databases use of, 28-2
 - new features, xviii to lxi
 - NEXT storage parameter
 - altering, 14-13, 15-12
 - description, 14-10
 - rollback segments, 13-18, 13-20
 - setting for the data dictionary, 21-28
 - NO_DATA_FOUND keyword, 30-11
 - NO_MERGE hint, 30-8
 - NOARCHIVELOG MODE
 - dropping datafiles, 12-9
 - NOARCHIVELOG mode
 - archiving, 8-2
 - definition, 8-2
 - media failure, 8-3
 - no hot backups, 8-3
 - running in, 8-2
 - switching to, 8-5
 - taking datafiles offline in, 12-9
 - NOAUDIT statement
 - disabling audit options, 26-13
 - disabling default object audit options, 26-15
 - disabling object auditing, 26-14
 - disabling statement and privilege auditing, 26-14
 - NOLOGGING CLAUSE
 - CREATE TABLESPACE, 11-20
 - NOMOUNT option
 - STARTUP command, 4-6
 - normal transmission mode
 - definition, 8-14
 - Novell's NetWare Management System, 28-32
- O**
-
- O7_DICTIONARY_ACCESSIBILITY initialization parameter, 25-3
 - object privileges
 - for external tables, 15-39
 - granting on behalf of the owner, 25-14
 - revoking, 25-16
 - revoking on behalf of owner, 25-17
 - objects
 - See also* schema objects
 - referencing with synonyms, 29-28
 - See also* schema objects
 - offline tablespaces
 - priorities, 11-21
 - rollback segments and, 13-22
 - taking offline, 11-21
 - online redefinition of tables
 - abort and cleanup, 15-20
 - example, 15-20
 - features of, 15-16
 - intermediate synchronization, 15-19
 - restrictions, 15-22
 - steps, 15-17
 - online redo log, 7-2
 - See also* redo logs
 - creating
 - groups and members, 7-12
 - creating members, 7-13
 - dropping groups, 7-16
 - dropping members, 7-16
 - forcing a log switch, 7-18
 - guidelines for configuring, 7-5
 - INVALID members, 7-17
 - location of, 7-9
 - managing, 7-1
 - moving files, 7-14
 - number of files in the, 7-10
 - optimum configuration for the, 7-10
 - privileges
 - adding groups, 7-12
 - dropping groups, 7-16
 - dropping members, 7-17
 - forcing a log switch, 7-18
 - renaming files, 7-14
 - renaming members, 7-14
 - specifying ARCHIVE_LAG_TIME, 7-10
 - STALE members, 7-17
 - viewing information about, 7-20
 - online redo log files
 - creating as Oracle-managed files, 3-19
 - OPEN_LINKS initialization parameter, 29-20
 - opening a database
 - after creation, 1-6
 - operating systems
 - accounts, 25-24

- authentication, 24-12, 25-22
- database administrators requirements for, 1-10
- enabling and disabling roles, 25-25
- renaming and relocating files, 12-10
- role identification, 25-23
- roles and, 25-22
- security in, 23-3
- Optimal Flexible Architecture (OFA), 2-7
- OPTIMAL storage parameter
 - description, 14-11
 - rollback segments, 13-17, 13-18, 13-20
- optional destinations
 - for archived redo logs
 - destinations
 - archived redo logs
 - mandatory, 8-16
- ORA_TZFILE environment variable
 - specifying time zone file for database, 2-29
- ORA-00900 error, 30-11
- ORA-02015 error, 30-11
- ORA-02055 error
 - integrity constraint violation, 30-3
- ORA-02067 error
 - rollback required, 30-3
- ORA-06510 error
 - PL/SQL error, 30-12
- Oracle
 - installing, 1-5
 - release numbers, 1-8
- Oracle Call Interface. *See* OCI
- Oracle Enterprise Manager, 4-2
- Oracle Managed Files feature
 - See also* Oracle-managed files
- Oracle Net
 - service names in, 8-15
 - transmitting archived logs via, 8-15
- Oracle Universal Installer, 2-5
- Oracle9i Real Application Clusters
 - allocating extents for cluster, 18-9
 - sequence numbers and, 20-13
 - threads of online redo log, 7-2
- Oracle-managed files
 - behavior, 3-21
 - benefits, 3-3

- CREATE DATABASE statement, 3-8
- creating, 3-6 to 3-21
- creating control files, 3-17
- creating datafiles, 3-14
- creating online redo log files, 3-19
- creating tempfiles, 3-16
- described, 3-2
- dropping datafile, 3-21
- dropping online redo log files, 3-22
- dropping tempfile, 3-21
- initialization parameters, 3-4
- introduction, 2-25
- naming, 3-7
- renaming, 3-22
- scenarios for using, 3-22 to 3-28
- ORAPWD utility, 1-20
- ORGANIZATION EXTERNAL clause
 - of CREATE TABLE, 15-35
- OS authentication, 1-17
- OS_ROLES parameter
 - operating-system authorization and, 25-9
 - REMOTE_OS_ROLES and, 25-25
 - using, 25-23
- OSDBA group, 1-18
- OSOPER group, 1-18
- OTHER_GROUPS for Database Resource
 - Manager, 27-6, 27-13, 27-16, 27-19, 27-28

P

- packages
 - DBMS_JOB, 10-3
 - DBMS_METADATA, 21-31
 - DBMS_REDEFINITION, 15-17
 - DBMS_REPAIR, 22-2 to 22-15
 - DBMS_RESOURCE_MANAGER, 27-4, 27-8, 27-9, 27-20, 27-21
 - DBMS_RESOURCE_MANAGER_PRIVS, 27-9, 27-20
 - DBMS_RESUMABLE, 14-22
 - DBMS_SESSION, 27-23
 - DBMS_SPACE, 14-25, 21-31
 - DBMS_STATS, 15-9, 21-4
 - DBMS_STORAGE_MAPPING, 12-22, 12-23
 - DBMS_UTILITY

- used for computing statistics, 21-6
 - privileges for recompiling, 21-25
 - recompiling, 21-25
- parallel execution
 - managing, 5-18
 - parallel hints, 5-18
 - parallelizing index creation, 16-7
 - resumable space allocation, 14-18
- parallel hints, 5-18
- PARALLEL_DEGREE_LIMIT_ABSOLUTE resource
 - allocation method, 27-15
- parallelizing table creation, 15-4, 15-8
- parameter files
 - See also* initialization parameter file.
- PARTITION BY HASH clause, 17-12
- PARTITION BY LIST clause, 17-13
- PARTITION BY RANGE clause, 17-11
 - for composite-partitioned tables, 17-14, 17-15
- PARTITION clause
 - for composite-partitioned tables, 17-14, 17-15
 - for hash partitions, 17-12
 - for list partitions, 17-13
 - for range partitions, 17-11
- partition views
 - converting to partitioned table, 17-62
- partitioned indexes, 17-1 to 17-65
 - adding partitions, 17-31
 - creating local index on composite partitioned table, 17-15
 - creating local index on hash partitioned table, 17-13
 - creating range partitions, 17-12
 - description, 17-2
 - dropping partitions, 17-35
 - global, 17-3
 - local, 17-3
 - maintenance operations, 17-22 to 17-61
 - table of, 17-24
 - modifying partition default attributes, 17-43
 - modifying real attributes of partitions, 17-45
 - moving partitions, 17-49
 - rebuilding index partitions, 17-50
 - renaming index partitions/subpartitions, 17-52
 - secondary indexes on index-organized tables, 17-20
 - splitting partitions, 17-57
- partitioned tables, 17-1 to 17-65
 - adding partitions, 17-27
 - adding subpartitions, 17-29, 17-30
 - coalescing partitions, 17-31
 - converting partition views, 17-62
 - creating composite partitions and subpartitions, 17-14
 - creating hash partitions, 17-12
 - creating list partitions, 17-13
 - creating range partitions, 17-11, 17-12
 - description, 17-2
 - DISABLE ROW MOVEMENT, 17-10
 - dropping partitions, 17-32
 - ENABLE ROW MOVEMENT, 17-10
 - exchanging partitions, 17-35
 - exchanging subpartitions, 17-37, 17-38
 - global indexes on, 17-3
 - index-organized tables, 17-10, 17-20, 17-21
 - local indexes on, 17-3
 - maintenance operations, 17-22 to 17-61
 - table of, 17-23
 - marking indexes UNUSABLE, 17-28, 17-30, 17-31, 17-33, 17-35, 17-36, 17-38, 17-44, 17-45, 17-48, 17-52, 17-59
 - merging partitions, 17-38
 - modifying default attributes, 17-43
 - modifying real attributes of partitions, 17-44
 - modifying real attributes of subpartitions, 17-45
 - moving partitions, 17-48
 - moving subpartitions, 17-49
 - rebuilding index partitions, 17-50
 - renaming partitions, 17-51
 - renaming subpartitions, 17-51
 - splitting partitions, 17-52
 - truncating partitions, 17-59
 - truncating subpartitions, 17-61
 - updating global indexes automatically, 17-26
- partitioning
 - composite, 17-7
 - creating partitions, 17-10 to 17-21
 - default partition, 17-7
 - default subpartition, 17-10
 - hash, 17-5
 - indexes, 17-2

- See also* partitioned indexes
 - index-organized tables, 17-10, 17-20, 17-21
 - list, 17-5, 17-45, 17-46
 - maintaining partitions, 17-22 to 17-61
 - methods, 17-3
 - range, 17-4
 - range-list, 17-8, 17-15
 - subpartition templates, 17-17
 - tables, 17-2
 - See also* partitioned tables
- partitions
 - See also* partitioned tables.
 - See also* partitioned indexes.
- PARTITIONS clause
 - for hash partitions, 17-12
- password file authentication, 1-19
- passwords
 - changing for roles, 25-7
 - default for SYS and SYSTEM, 1-11
 - encrypted
 - database, 23-5
 - encryption, 24-9
 - password file, 1-23
 - creating, 1-20
 - OS authentication, 1-16
 - removing, 1-25
 - state of, 1-25
 - privileges for changing for roles, 25-8
 - privileges to alter, 24-6
 - roles, 25-8
 - security policy for users, 23-5
 - setting REMOTE_LOGIN_PASSWORD
 - parameter, 1-22
 - user authentication, 24-9
 - viewing for database links, 29-22
- PCTFREE parameter
 - clustered tables, 14-4
 - clusters, used in, 18-5
 - guidelines for setting, 14-4
 - indexes, 14-4
 - non-clustered tables, 14-4
 - PCTUSED, use with, 14-7
 - usage, 14-2
- PCTFREE storage parameter
 - altering, 15-11
 - table creation, 15-2
- PCTINCREASE parameter
 - altering, 14-13
 - rollback segments, 13-18, 13-20
 - setting for the data dictionary, 21-28
- PCTINCREASE storage parameter
 - altering, 15-12
 - description, 14-10
- PCTUSED parameter, 11-8, 11-9
 - clusters, used in, 18-5
 - guidelines for setting, 14-6
 - PCTFREE, use with, 14-7
 - usage, 14-5
- PCTUSED storage parameter
 - altering, 15-11
 - table creation, 15-2
- pending area for Database Resource Manager
 - plans, 27-12 to 27-14
 - validating plan schema changes, 27-12
- pending transaction tables, 32-22
- performance
 - index column order, 16-5
 - location of datafiles and, 12-4
 - tuning archiving, 8-19
- physical structure of a database, 1-6
- plan schemas for Database Resource
 - Manager, 27-6, 27-12, 27-16, 27-25, 27-32
 - examples, 27-25
 - validating plan changes, 27-12
- planning
 - database creation, 2-2
 - relational design, 1-6
 - the database, 1-5
- PL/SQL
 - errors
 - ORA-06510, 30-12
 - program units
 - replaced views and, 20-11
 - user-defined exceptions, 30-11
- PRAGMA_EXCEPTION_INIT procedure
 - assigning exception names, 30-12
- predefined roles, 1-13
- prepare phase, 31-11
 - recognizing read-only nodes, 31-12
 - two-phase commit, 31-11

- prepare/commit phases
 - effects of failure, 32-23
 - failures during, 32-8
 - locked resources, 32-23
 - pending transaction table, 32-22
- prepared response
 - two-phase commit, 31-12
- prerequisites
 - for creating a database, 2-4
- PRIMARY KEY constraints
 - associated indexes, 16-11
 - dropping associated indexes, 16-23
 - enabling on creation, 16-11
 - foreign key references when dropped, 21-19
 - indexes associated with, 16-11
- private database links, 28-15
- private rollback segments, 13-15, 13-19
 - taking offline, 13-24
- private synonyms, 20-13
- granting privileges and roles
 - specifying ALL, 25-4
- revoking privileges and roles
 - specifying ALL, 25-4
- privileges, 25-2
 - See also* system privileges.
 - adding redo log groups, 7-12
 - altering
 - indexes, 16-19
 - passwords, 24-7
 - sequences, 20-12
 - users, 24-6
 - altering role authentication method, 25-8
 - altering tables, 15-10
 - application developers and, 23-10
 - audit object, 26-12
 - auditing system, 26-12
 - auditing use of, 26-11
 - cascading revokes, 25-19
 - closing a database link, 30-2
 - column, 25-15
 - creating
 - sequences, 20-12
 - synonyms, 20-14
 - views, 20-2
 - creating database links, 29-8
 - creating roles, 25-7
 - creating rollback segments, 13-19
 - creating tables, 15-6
 - creating tablespaces, 11-4
 - creating users, 24-2
 - database administrator, 1-10
 - disabling automatic archiving, 8-8
 - dropping
 - indexes, 16-22
 - online redo log members, 7-17
 - redo log groups, 7-16
 - sequences, 20-13
 - synonyms, 20-14
 - views, 20-10
 - dropping profiles, 24-23
 - dropping roles, 25-10
 - dropping rollback segments, 13-25
 - dropping tables, 15-23
 - enabling and disabling resource limits, 24-19
 - enabling and disabling triggers, 21-13
 - enabling automatic archiving, 8-6
 - for external tables, 15-39
 - forcing a log switch, 7-18
 - granting, 25-11
 - granting object privileges, 25-12
 - granting system privileges, 25-11
 - granting, about, 25-11
 - grouping with roles, 25-6
 - individual privilege names, 25-2
 - listing grants, 25-27
 - managing with procedures, 29-32
 - managing with synonyms, 29-30
 - managing with views, 29-28
 - manually archiving, 8-9
 - object, 25-4
 - on selected columns, 25-18
 - policies for managing, 23-6
 - recompiling packages, 21-25
 - recompiling procedures, 21-25
 - recompiling views, 21-25
 - renaming
 - objects, 21-3
 - redo log members, 7-14
 - replacing views, 20-10
 - RESTRICTED SESSION system privilege, 4-7

- revoking, 25-16
- revoking object, 25-16
- revoking object privileges, 25-16, 25-19
- revoking system privileges, 25-16
- setting resource costs, 24-22
- system, 25-2
- taking tablespaces offline, 11-21
- truncating, 21-11
- See also* system privileges.

procedures

- external, 5-20
- location transparency using, 29-30, 29-31, 29-32
- recompiling, 21-25
- remote calls, 28-46

process monitor, 5-12

processes

- See also* server processes

PROCESSES initialization parameter

- setting before database creation, 2-41

PRODUCT_COMPONENT_VERSION view, 1-10

profiles, 24-18

- altering, 24-21
- assigning to users, 24-20
- creating, 24-20
- default, 24-20
- disabling resource limits, 24-19
- dropping, 24-23
- enabling resource limits, 24-19
- listing, 24-23
- managing, 24-18
- privileges for dropping, 24-23
- privileges to alter, 24-21
- privileges to set resource costs, 24-22
- PUBLIC_DEFAULT, 24-20
- setting a limit to null, 24-21
- viewing, 24-25

program global area (PGA)

- effect of MAX_ENABLED_ROLES on, 25-22

proxies

- auditing clients of, 26-13
- proxy authentication and authorization, 24-16

proxy authentication, 24-16

proxy authorization, 24-16

proxy servers

- auditing clients, 26-13

PROXY_USERS view, 24-16

public database links

- connected user, 29-35
- fixed user, 29-34

public fixed user database links, 29-34

public rollback segments, 13-19

- taking offline, 13-24

public synonyms, 20-13

PUBLIC user group

- granting and revoking privileges to, 25-20
- procedures and, 25-20

PUBLIC_DEFAULT profile

- dropping profiles and, 24-23
- using, 24-20

PURGE_LOST_DB_ENTRY procedure

- DBMS_TRANSACTION package, 32-13

purging pending rows

- from data dictionary, 32-13
- when necessary, 32-14

Q

queries

- distributed, 28-34
 - application development issues, 30-3
- distributed or remote, 28-33
- location transparency and, 28-45
- post-processing, 30-4
- remote, 30-4
- transparency, 29-32

quiescing a database, 4-14

quotas

- listing, 24-23
- revoking from users, 24-4
- setting to zero, 24-4
- tablespace, 24-4
- tablespace quotas, 11-3
- temporary segments and, 24-4
- unlimited, 24-4
- viewing, 24-25

R

range partitioning

- creating tables using, 17-11

- index-organized tables, 17-20
 - when to use, 17-4
- range-hash partitioning
 - subpartitioning template, 17-17
- range-list partitioning, 17-8, 17-15
 - subpartitioning template, 17-19
- read consistency
 - managing in distributed databases, 32-25
- read-only database
 - opening, 4-10
- read-only response
 - two-phase commit, 31-12
- read-only tablespaces
 - datafiles, 12-8
- read-only tablespaces, *see* tablespaces, read-only
- REBUILD PARTITION clause, 17-49, 17-51
- REBUILD SUBPARTITION clause, 17-51
- REBUILD UNUSABLE LOCAL INDEXES
 - clause, 17-51
- REBUILD_FREELISTS procedure, 22-3, 22-6, 22-8
 - example, 22-13
- rebuilding indexes, 16-20
 - costs, 16-8
 - online, 16-20
- RECOVER option
 - STARTUP command, 4-8
- recoverer process, 5-13
- recoverer process (RECO)
 - disabling, 32-24, 32-25
 - distributed transaction recovery, 32-24
 - enabling, 32-24, 32-25
 - pending transaction table, 32-24
- recovery
 - creating new control files, 6-6
- Recovery Manager
 - starting a database, 4-2
 - starting an instance, 4-2
- RECOVERY_CATALOG_OWNER role, 25-6
- redefining tables
 - online, 15-16 to 15-22
- redo log files
 - active (current), 7-4
 - analyzing, 9-1
 - archived
 - advantages of, 8-2
 - contents of, 8-2
 - log switches and, 7-5
 - archived redo log files, 8-5
 - archived redo logs, 8-2
 - available for use, 7-3
 - circular use of, 7-3
 - clearing, 7-7, 7-19
 - restrictions, 7-19
 - contents of, 7-2
 - creating
 - groups and members, 7-12
 - creating members, 7-13
 - distributed transaction information in, 7-3
 - groups, 7-6
 - creating, 7-12
 - dropping, 7-16
 - members, 7-6
 - threads, 7-2
 - how many in redo log, 7-10
 - inactive, 7-4
 - legal and illegal configurations, 7-7
 - LGWR and the, 7-3
 - log sequence numbers of, 7-5
 - log switches, 7-5
 - members, 7-6
 - creating, 7-12
 - dropping, 7-16
 - maximum number of, 7-10
 - mirrored
 - log switches and, 7-7
 - multiplexed
 - diagrammed, 7-6
 - if all inaccessible, 7-7
 - multiplexing, 7-5
 - groups, 7-6
 - if some members inaccessible, 7-7
 - online, 7-2
 - recovery use of, 7-2
 - requirement of two, 7-3
 - threads of, 7-2
 - online redo log, 7-1
 - planning the, 7-5, 7-10
 - privileges
 - adding groups and members, 7-12
 - redo entries, 7-2

- requirements, 7-7
- verifying blocks, 7-18
- viewing, 2-31
- redo logs
 - See also* online redo log
 - storing separately from datafiles, 12-4
 - unavailable when database is opened, 4-6
- redo records, 7-2
 - LOGGING and NOLOGGING, 11-20
- REFERENCES privilege
 - CASCADE CONSTRAINTS option, 25-18
 - revoking, 25-18
- referential integrity
 - distributed database systems
 - application development, 30-3
- relational design
 - planning, 1-6
- release number format, 1-8
- releases, 1-8
 - checking the Oracle database release
 - number, 1-10
- relocating control files, 6-5
- remote connections, 1-25
 - connecting as SYSOPER/SYSDBA, 1-13
 - password files, 1-20
- remote data
 - querying, 29-33
 - updating, 29-33
- remote procedure calls, 28-46
 - distributed databases and, 28-46
- remote queries, 30-4
 - distributed databases and, 28-33
 - execution, 30-4
 - post-processing, 30-4
- remote transactions, 28-35
 - defined, 28-35
- REMOTE_LOGIN_PASSWORDFILE initialization
 - parameter, 1-22
- REMOTE_OS_AUTHENT initialization
 - parameter, 28-17
 - setting, 24-12
- REMOTE_OS_ROLES initialization parameter
 - setting, 25-9, 25-26
- RENAME PARTITION clause, 17-51
- RENAME statement, 21-3
- RENAME SUBPARTITION clause, 17-51
- renaming control files, 6-5
- renaming files
 - Oracle-managed files, 3-22
- REOPEN option
 - LOG_ARCHIVE_DEST_*n* initialization
 - parameter, 8-18
- repairing data block corruption
 - DBMS_REPAIR, 22-2 to 22-15
- resource allocation methods, 27-4
 - active session pool, 27-15
 - CPU resource, 27-14
 - EMPHASIS, 27-14
 - limit on degree of parallelism, 27-15
 - limiting degree of parallelism, 27-15
 - PARALLEL_DEGREE_LIMIT_
 - ABSOLUTE, 27-15
 - queueing resource allocation method, 27-15
 - ROUND-ROBIN, 27-16
- resource consumer groups, 27-3
 - creating, 27-16 to 27-17
 - DEFAULT_CONSUMER_GROUP, 27-16, 27-17, 27-21, 27-23
 - deleting, 27-17
 - LOW_GROUP, 27-17, 27-29
 - managing, 27-20 to 27-23
 - OTHER_GROUPS, 27-6, 27-13, 27-16, 27-19, 27-28
 - parameters, 27-16
 - SYS_GROUP, 27-17, 27-28
 - updating, 27-17
- resource limits
 - altering in profiles, 24-21
 - assigning with profiles, 24-20
 - costs and, 24-22
 - creating profiles and, 24-20
 - disabling, 24-19
 - enabling, 24-19
 - privileges to enable and disable, 24-19
 - privileges to set costs, 24-22
 - profiles, 24-18
 - PUBLIC_DEFAULT profile and, 24-20
 - setting to null, 24-21
- resource plan directives, 27-4, 27-12
 - deleting, 27-19

- specifying, 27-17 to 27-20
- updating, 27-19
- resource plans, 27-3
 - creating, 27-10 to 27-16
 - DELETE_PLAN_CASCADE, 27-16
 - deleting, 27-15
 - examples, 27-4, 27-25
 - parameters, 27-14
 - plan schemas, 27-6, 27-12, 27-16, 27-25, 27-32
 - subplans, 27-5, 27-6, 27-16
 - SYSTEM_PLAN, 27-15, 27-17, 27-28
 - top plan, 27-6, 27-13, 27-24
 - updating, 27-15
 - validating, 27-12
- RESOURCE role, 25-5
- RESOURCE_LIMIT initialization parameter
 - enabling and disabling limits, 24-19
- RESOURCE_MANAGER_PLAN initialization parameter, 27-24
- resources
 - profiles, 24-18
- responsibilities
 - database administrator, 1-2
 - of database users, 1-4
- RESTRICT OPTION
 - STARTUP command, 4-7
- RESTRICTED SESSION system privilege
 - connecting to database, 4-7
 - connecting to database., 4-7
 - restricted mode and, 4-7
- resumable space allocation
 - correctable errors, 14-17
 - detecting suspended statements, 14-21
 - disabling, 14-19
 - distributed databases, 14-18
 - enabling, 14-19
 - example, 14-23
 - how resumable statements work, 14-15
 - naming statements, 14-20
 - parallel execution and, 14-18
 - resumable operations, 14-16
 - setting as default for session, 14-20
 - timeout interval, 14-19, 14-20
- REVOKE CONNECT THROUGH clause
 - revoking proxy authorization, 24-16
- REVOKE statement, 25-16
 - when takes effect, 25-20
- revoking privileges and roles
 - on selected columns, 25-18
 - REVOKE statement, 25-16
 - when using operating-system roles, 25-25
- RMAN. *See* Recovery Manager.
- role identification
 - operating system accounts, 25-24
- roles
 - ADMIN OPTION and, 25-12
 - application developers and, 23-11
 - AQ_ADMINISTRATOR_ROLE, 25-6
 - AQ_USER_ROLE, 25-6
 - authorization, 25-8
 - authorized by enterprise directory service, 25-10
 - changing authorization for, 25-7
 - changing passwords, 25-7
 - CONNECT role, 25-5
 - database authorization, 25-8
 - DBA role, 1-13, 25-5
 - default, 24-7, 25-21
 - definition, 25-5
 - DELETE_CATALOG_ROLE, 25-6
 - disabling, 25-21
 - dropping, 25-10
 - enabling, 25-21
 - enterprise, 24-14, 25-10
 - EXECUTE_CATALOG_ROLE, 25-6
 - EXP_FULL_DATABASE, 25-5
 - global, 24-13, 25-10
 - global authorization, 25-10
 - GRANT statement, 25-25
 - granting, 25-11
 - granting, about, 25-11
 - HS_ADMIN_ROLE, 25-6
 - IMP_FULL_DATABASE, 25-5
 - listing, 25-30
 - listing grants, 25-28
 - listing privileges and roles in, 25-30
 - management using the operating system, 25-22
 - managing, 25-6
 - maximum, 25-22
 - multibyte characters in names, 25-7

- multibyte characters in passwords, 25-8
- network authorization, 25-9
- obtained through database links, 28-23
- operating system, 25-24
- operating system granting of, 25-23, 25-25
- operating-system authorization, 25-9
- OS management and the shared server, 25-25
- passwords for enabling, 25-8
- predefined, 1-13, 25-5
- privileges for creating, 25-7
- privileges for dropping, 25-10
- privileges, changing authorization method for, 25-8
- privileges, changing passwords, 25-8
- RECOVERY_CATALOG_OWNER, 25-6
- RESOURCE role, 25-5
- REVOKE statement, 25-25
- revoking, 25-16
- revoking ADMIN OPTION, 25-16
- security and, 23-6
- SELECT_CATALOG_ROLE, 25-6
- SET ROLE statement, 25-25
- SNMPAGENT, 25-6
- unique names for, 25-7
- WITH GRANT OPTION and, 25-13
- without authorization, 25-7
- rollback segments
 - acquiring automatically, 13-15, 13-23
 - acquiring on startup, 2-43
 - altering storage parameters, 13-21
 - AVAILABLE, 13-22
 - bringing online, 13-22
 - bringing online when new, 13-19
 - bringing PARTLY AVAILABLE segment online, 13-23
 - checking if offline, 13-23
 - creating, 13-19 to 13-21
 - displaying information about, 13-26
 - displaying names of all, 13-28
 - displaying PENDING OFFLINE segments, 13-29
 - dropping, 13-22, 13-25
 - equally sized extents, 13-17
 - explicitly assigning transactions to, 13-24
 - guidelines for managing, 13-13 to 13-18
 - in-doubt distributed transactions, 32-9
 - initial creation of SYSTEM, 13-14
 - INITIAL storage parameter, 13-18, 13-20
 - initialization parameters used with, 13-5
 - invalid status, 13-25
 - listing extents in, 21-35
 - location of, 13-18
 - making available for use, 13-22
 - maximum number of, 13-15
 - MINEXTENTS, 13-17, 13-18, 13-20
 - NEXT, 13-18, 13-20
 - OFFLINE, 13-22
 - OPTIMAL, 13-17, 13-18, 13-20
 - PARTLY AVAILABLE, 13-22
 - PCTINCREASE, 13-18, 13-20
 - PENDING OFFLINE, 13-24
 - private, 13-15, 13-19
 - privileges for dropping, 13-25
 - privileges required to create, 13-19
 - public, 13-19
 - public vs. private, 13-15
 - setting size of, 13-16
 - shrinking size of, 13-21
 - starting an instance using, 13-4
 - status for dropping, 13-25
 - status or state, 13-22
 - storage parameters, 13-20
 - taking offline, 13-23
 - taking tablespaces offline and, 11-23
 - using multiple, 13-14
- ROLLBACK statement
 - FORCE clause, 32-10, 32-11, 32-12
 - forcing, 32-9
- ROLLBACK_SEGMENTS initialization parameter, 13-15
 - adding rollback segments to, 13-19, 13-23
 - dropping rollback segments, 13-25
 - online at instance startup, 13-16
 - setting before database creation, 2-43
- rollbacks
 - ORA-02067 error, 30-3
- ROUND-ROBIN resource allocation method, 27-16
- row movement clause for partitioned tables, 17-10
- rows
 - chaining across blocks, 14-4

listing chained or migrated, 21-7

S

Sample Schemas

description, 2-33

savepoints

in-doubt transactions, 32-11, 32-13

schema objects

analyzing, 21-3 to 21-6

cascading effects on revoking, 25-19

creating multiple objects, 21-2

default audit options, 26-13

default tablespace for, 24-3

defining using DBMS_METADATA
package, 21-31

dependencies between, 21-23

disabling audit options, 26-14

distributed database naming conventions
for, 28-22

enabling audit options on, 26-12

global names, 28-22

granting privileges, 25-12

in a revoked tablespace, 24-4

listing by type, 21-33

name resolution in SQL statements, 21-25

obtaining metadata about, 21-31

owned by dropped users, 24-7

privileges to access, 25-4

privileges to rename, 21-3

privileges with, 25-4

renaming, 21-3

revoking privileges, 25-16

validating structure, 21-6

viewing information, 21-30

schema-independent users, 24-14

SCN. *See* system change number.

SCOPE clause

ALTER SYSTEM SET, 2-47

Secure Sockets Layer, 23-2, 24-8, 24-14

security

accessing a database, 23-2

administrator of, 23-2

application developers and, 23-10

auditing policies, 23-20

authentication of users, 23-2

data, 23-3

database security, 23-2

database users and, 23-2

distributed databases, 28-24

centralized user management, 28-27

establishing policies, 23-1

general users, 23-4

level of, 23-3

multibyte characters in role names, 25-7

multibyte characters in role passwords, 25-8

operating-system security and the
database, 23-3

policies for database administrators, 23-8

privilege management policies, 23-6

privileges, 23-2

protecting the audit trail, 26-18

remote objects, 29-28

REMOTE_OS_ROLES parameter, 25-26

roles to force security, 23-6

security officer, 1-3

test databases, 23-10

using synonyms, 29-30

SEGMENT_FIX_STATUS procedure, 22-3

segments

available space, 21-31

data dictionary, 21-28

data dictionary views for, 21-33

deallocating unused space, 14-25

displaying information on, 21-35

monitoring rollback, 13-27

rollback. *See* rollback segments.

temporary

storage parameters, 14-13

SELECT statement

FOR UPDATE clause, 29-33

SELECT_CATALOG_ROLE role, 25-6

SELECT_CATALOG_ROLE roll, 25-3

sequences

altering, 20-13

creating, 20-12

dropping, 20-13

managing, 20-11

Oracle Real Applications Clusters and, 20-13

privileges for altering, 20-12

- privileges for creating, 20-12
- privileges for dropping, 20-13
- SERVER parameter
 - net service name, 29-16
- server parameter file
 - creating, 2-46
 - defined, 2-44
 - error recovery, 2-50
 - exporting, 2-49
 - migrating to, 2-45
 - RMAN backup, 2-50
 - setting initialization parameter values, 2-47
 - SPFILE initialization parameter, 2-47
 - STARTUP command behavior, 2-45, 4-3
 - viewing parameter settings, 2-51
- server processes
 - archiver (ARCn), 5-12
 - background, 5-11 to 5-13
 - checkpoint (CKPT), 5-12
 - database writer (DBWn), 5-12
 - dedicated, 5-2
 - dispatcher (Dnnn), 5-13
 - dispatchers, 5-6 to 5-10
 - global cache service (LMS), 5-13
 - job queue coordinator process (CJQ0), 5-13, 10-2
 - log writer (LGWR), 5-12
 - monitoring, 5-13
 - monitoring locks, 5-15
 - process monitor (PMON), 5-12
 - recoverer (RECO), 5-13
 - shared server, 5-3 to 5-10
 - system monitor (SMON), 5-12
 - trace files for, 5-15
- servers
 - role in two-phase commit, 31-5
- service names
 - database links and, 29-13
- session trees
 - distributed transactions, 31-4
 - clients, 31-5
 - commit point site, 31-6, 31-8
 - database servers, 31-5
 - global coordinators, 31-6
 - local coordinators, 31-6
 - tracing, 32-5
- sessions
 - auditing connections and disconnections, 26-11
 - listing privilege domain of, 25-29
 - setting advice for transactions, 32-10
 - terminating, 5-21 to 5-23
 - viewing memory use, 24-26
- sessions, user
 - active, 5-22
 - inactive, 5-23
 - marked to be terminated, 5-22
 - terminating, 5-21
 - viewing terminated sessions, 5-23
- SET ROLE statement
 - how password is set, 25-8
 - used to enable/disable roles, 25-21
 - when using operating-system roles, 25-25
- SET TIME_ZONE clause
 - CREATE DATABASE, 2-28
- SET TIME_ZONE clause
 - ALTER SESSION, 2-28
 - time zone files, 2-28
- SET TRANSACTION statement
 - naming transactions, 32-2
 - USE ROLLBACK SEGMENT option, 13-24
- SGA. *See* system global area.
- SGA_MAX_SIZE initialization parameter, 2-38
 - setting size, 2-40
- shared database links
 - configuring, 29-16
 - creating links, 29-14, 29-15
 - to dedicated servers, 29-16
 - to shared servers, 29-17
 - determining whether to use, 29-14
 - example, 28-20
- SHARED keyword
 - CREATE DATABASE LINK statement, 29-15
- shared server, 5-3
 - adjusting number of dispatchers, 5-8
 - enabling and disabling, 5-10
 - initialization parameters, 5-5
 - OS role management restrictions, 25-25
 - setting initial number of dispatchers, 5-6
 - setting initial number of servers, 5-7
 - setting minimum number of servers, 5-10
 - views, 5-10

- shared server processes
 - trace files for, 5-15
- shared SQL
 - for remote and distributed statements, 28-34
- SHARED_SERVERS initialization parameter
 - initial setting, 5-7
- SHUTDOWN command
 - ABORT option, 4-13
 - IMMEDIATE option, 4-12
 - NORMAL option, 4-12
 - TRANSACTIONAL option, 4-13
- Simple Network Management Protocol (SNMP)
 - support
 - database management, 28-32
- single-process systems
 - enabling distributed recovery, 32-25
- single-table hash clusters, 19-5
- site autonomy
 - distributed databases, 28-24
- SKIP_CORRUPT_BLOCKS procedure, 22-3, 22-7
 - example, 22-14
- snapshot too old
 - OPTIMAL storage parameter and, 13-18
 - undo retention and, 13-9
- SNMPAGENT role, 25-6
- SORT_AREA_SIZE initialization parameter
 - index creation and, 16-3
- space allocation
 - resumable, 14-14 to 14-25
- space management
 - data blocks, 14-2 to 14-7
 - datatypes, space requirements, 14-29
 - deallocating unused space, 14-25
 - setting storage parameters, 14-9 to 14-13
- SPACE_ERROR_INFO procedure, 14-21
- specifying destinations
 - for archived redo logs, 8-10
- specifying multiple ARCH processes, 8-19
- SPFILE initialization parameter, 2-47
 - specifying from client machine, 4-5
- SPLIT PARTITION clause, 17-27, 17-52
- SQL errors
 - ORA-00900, 30-11
 - ORA-02015, 30-11
- SQL statements
 - disabling audit options, 26-14
 - distributed databases and, 28-33
 - enabling audit options on, 26-11
- SQL*Loader
 - about, 1-26
- SQL*Plus
 - starting, 4-3
 - starting a database, 4-2
 - starting an instance, 4-2
- SQL_TRACE initialization parameter
 - trace files and, 5-15
- SSL. *See* Secure Sockets Layer.
- STALE status
 - of redo log members, 7-17
- standby transmission mode
 - definition of, 8-14
 - Oracle Net and, 8-15
 - RFS processes and, 8-15
- starting a database
 - forcing, 4-8
 - Oracle Enterprise Manager, 4-2
 - recovery and, 4-8
 - Recovery Manage, 4-2
 - restricted mode, 4-7
 - SQL*Plus, 4-2
 - when control files unavailable, 4-6
 - when redo logs unavailable, 4-6
- starting an instance
 - automatically at system startup, 4-9
 - database closed and mounted, 4-6
 - database name conflicts and, 2-36
 - enabling automatic archiving, 8-7
 - forcing, 4-8
 - mounting and opening the database, 4-6
 - normally, 4-6
 - Oracle Enterprise Manager, 4-2
 - recovery and, 4-8
 - Recovery Manager, 4-2
 - remote instance startup, 4-9
 - restricted mode, 4-7
 - SQL*Plus, 4-2
 - when control files unavailable, 4-6
 - when redo logs unavailable, 4-6
 - without mounting a database, 4-6
- STARTUP command

- default behavior, 2-45
- MOUNT option, 4-7
- NOMOUNT option, 2-17, 4-6
- RECOVER option, 4-8
- RESTRICT option, 4-7
- starting a database, 4-2, 4-3
- statistics
 - automatically collecting for tables, 15-9
- storage
 - quotas and, 24-4
 - revoking tablespaces and, 24-4
 - unlimited quotas, 24-4
- STORAGE clause
 - See also* storage parameters
- storage parameters
 - altering, 15-11
 - altering defaults for tablespaces, 11-11
 - applicable objects, 14-8
 - BUFFER POOL, 14-11
 - changing for data dictionary objects, 21-27
 - data dictionary, 21-27
 - default, 14-9
 - example, 14-14
 - for the data dictionary, 21-28
 - FREELIST GROUPS, 14-11
 - FREELISTS, 14-11
 - INITIAL, 14-10
 - INITIAL, cannot alter, 15-12
 - INTRANS, altering, 15-11
 - MAXEXTENTS, 14-10
 - MAXTRANS, altering, 15-11
 - MINEXTENTS, 14-10
 - MINEXTENTS, cannot alter, 15-12
 - NEXT, 14-10
 - NEXT, altering, 15-12
 - OPTIMAL, 14-11
 - OPTIMAL (in rollback segments), 13-18
 - PCTFREE, altering, 15-11
 - PCTFREE, specifying, 15-2
 - PCTINCREASE, 14-10
 - PCTINCREASE, altering, 15-12
 - PCTUSED, altering, 15-11
 - PCTUSED, specifying, 15-2
 - precedence of, 14-13
 - rollback segments, 13-20
 - setting, 14-9 to 14-13
 - SYSTEM rollback segment, 13-21
 - temporary segments, 14-13
- storage subsystems
 - mapping files to physical devices, 12-15 to 12-27
- STORE IN clause, 17-14
- stored procedures
 - distributed query creation, 30-3
 - managing privileges, 29-32
 - privileges for recompiling, 21-25
 - remote object security, 29-32
 - using privileges granted to PUBLIC, 25-20
- SUBPARTITION BY HASH clause
 - for composite-partitioned tables, 17-14
- SUBPARTITION BY LIST clause
 - for composite-partitioned tables, 17-15
- SUBPARTITION clause, 17-29, 17-30, 17-54
 - for composite-partitioned tables, 17-14, 17-15
- subpartition template
 - modifying, 17-48
- subpartition templates, 17-17
- subpartitions, 17-2
- SUBPARTITIONS clause, 17-29, 17-54
 - for composite-partitioned tables, 17-14
- subqueries, 29-33
 - in remote updates, 28-34
- SunSoft's SunNet Manager, 28-32
- supplemental logging
 - for LogMiner utility, 9-19
 - LogMiner utility
 - identification keys, 9-20
 - log groups, 9-22
- SWITCH LOGFILE option
 - ALTER SYSTEM statement, 7-18
- synonyms
 - CREATE statement, 29-28
 - creating, 20-14
 - definition and creation, 29-28
 - displaying dependencies of, 21-34
 - dropping, 20-15
 - examples, 29-29
 - location transparency using, 29-28
 - managing, 20-13 to 20-15
 - managing privileges, 29-30
 - name resolution, 28-42

- name resolution in distributed databases, 28-42
- private, 20-13
- privileges for creating, 20-14
- privileges for dropping, 20-14
- public, 20-13
- remote object security, 29-30

SYS

- specifying password for CREATE DATABASE statement, 2-23

SYS account

- default password, 1-11
- objects owned, 1-12
- policies for protecting, 23-8
- privileges, 1-12
- user, 1-12

SYS_GROUP for Database Resource Manager, 27-17, 27-28

SYS.AUD\$ table

- audit trail, 26-2
- creating and deleting, 26-19

SYSDBA system privilege

- connecting to database, 1-14

SYSOPER system privilege

- connecting to database, 1-14

SYSOPER/SYSDBA privileges

- adding users to the password file, 1-23
- connecting with, 1-13
- determining who has privileges, 1-24
- granting and revoking, 1-24

SYSTEM

- specifying password for CREATE DATABASE, 2-23

SYSTEM account

- default password, 1-11
- objects owned, 1-12
- policies for protecting, 23-8

system change number

- using VSDATAFILE to view information about, 12-29
- when assigned, 7-2

system change numbers (SCN)

- coordination in a distributed database system, 31-14

- in-doubt transactions, 32-12

system global area

- initialization parameters affecting size, 2-38
- specifying buffer cache sizes, 2-40

system monitor, 5-12

system privileges, 25-2

- ADMINISTER_RESOURCE_MANAGER, 27-8
- described, 25-2
- for external tables, 15-39
- GRANT ANY OBJECT PRIVILEGE, 25-14, 25-17

- granting, 25-11

SYSTEM rollback segment

- altering storage parameters of, 13-21

SYSTEM Tablespace

- creating locally managed, 2-26

SYSTEM tablespace

- cannot drop, 11-29
- initial rollback segment, 13-14
- restrictions on taking offline, 12-8
- when created, 11-3

SYSTEM_PLAN for Database Resource Manager, 27-15, 27-17, 27-28

T

tables

- adding columns, 15-13
- allocating extents, 15-12
- altering, 15-10
- altering physical attributes, 15-11
- analyzing, 21-3 to 21-6
- clustered (hash). *See* hash clusters
- clustered (index). *See* clusters.
- creating, 15-7
- creating in locally managed tablespaces, 15-3
- data block space, specifying, 15-2
- designing before creating, 15-2
- dropping, 15-23
- dropping columns, 15-14 to 15-16
- estimating size, 15-4
- external, 15-33 to 15-39
- guidelines for managing, 15-2
- hash clustered. *See* hash clusters
- historical
 - moving time windows, 17-61
- increasing column length, 15-13

- index-organized, 15-24 to 15-33
 - partitioning, 17-19 to 17-21
- key-preserved, 20-6
- limiting indexes on, 16-5
- managing, 15-1 to 15-41
- modifying column definition, 15-13
- moving, 15-12
- parallelizing creation, 15-4, 15-8
- partitioned, 17-2 to 17-65
 - see also* partitioned tables
- planning large tables, 15-5
- privileges for creation, 15-6
- privileges to alter, 15-10
- redefining online, 15-16 to 15-22
- renaming columns, 15-14
- restrictions when creating, 15-6
- separating from indexes, 15-6
- setting storage parameters, 15-4
- specifying location, 15-3
- specifying PCTFREE for, 14-4
- statistics collection, automatic, 15-9
- temporary, 15-8
- temporary space for creation, 15-6
- truncating, 21-9
- unrecoverable (NOLOGGING), 15-4
- validating structure, 21-6
- views, 15-40

tablespace set, 11-37

tablespaces

- adding datafiles, 12-5
- altering storage parameters, 11-11
- assigning defaults for users, 24-3
- assigning user quotas, 11-3
- automatic segment space management, 11-8
- checking default storage parameters, 11-51
- coalescing free space, 11-16
- creating a default temporary tablespace, 2-24
- creating undo tablespace at database creation, 2-24
- DBMS_SPACE_ADMIN package, 11-30
- default quota, 24-4
- detecting and repairing defects, 11-30
- dictionary managed, 11-10 to 11-12
- dropping, 11-29
- guidelines for managing, 11-2
- listing files of, 11-51
- listing free space in, 11-51
- locally managed, 11-5 to 11-10
- locally managed SYSTEM, 2-26
- locally managed temporary, 11-13
- location, 12-4
- making read-only, 11-25
- making read-only writable, 11-27
- migrating SYSTEM to locally managed, 11-34
- multiple block sizes, 11-41
- on a WORM device, 11-27
- privileges for creating, 11-4
- privileges to take offline, 11-21
- quotas for users, 24-4
- quotas, assigning, 11-3
- revoking from users, 24-4
- setting default storage parameters, 14-11
- setting default storage parameters for, 11-3
- specifying non-standard block sizes, 11-19
- SYSTEM tablespace, 11-3
- taking offline normal, 11-21
- taking offline temporarily, 11-22
- tempfiles in locally managed, 11-13
- temporary
 - assigning to users, 24-5
 - for creating large indexes, 16-13
- temporary, creating, 11-12
- transportable, 11-35 to 11-49
- undo, 13-2 to 13-13
- unlimited quotas, 24-4
- using multiple, 11-2
- viewing quotas, 24-25

tempfiles, 11-13

- creating as Oracle-managed files, 3-16
- dropping Oracle-managed files, 3-21
- taking offline, 11-23

templates

- for databases (DBCA), 2-11

temporary segments

- index creation and, 16-3

temporary tables

- creating, 15-8

temporary tablespaces, *see* tablespaces, temporary

terminating user sessions

- active sessions, 5-22

- identifying sessions, 5-22
- inactive session, example, 5-23
- inactive sessions, 5-23
- threads
 - online redo log, 7-2
- time zone
 - files, 2-28
 - setting for database, 2-28
- TNSNAMES.ORA file, 8-11
- trace files
 - job failures and, 10-10
 - location of, 5-16
 - log writer, 5-16
 - log writer process and, 7-6
 - size of, 5-16
 - using, 5-15, 5-16
 - when written, 5-17
- tracing
 - archivelog process, 8-21
- transaction control statements
 - distributed transactions and, 31-4
- transaction failures
 - simulating, 32-24
- transaction management
 - overview, 31-10
- transaction processing
 - distributed systems, 28-33
- transactions
 - assigning to specific rollback segment, 13-24
 - closing database links, 30-2
 - distributed
 - two-phase commit and, 28-36
 - in-doubt, 31-14
 - after a system failure, 32-8
 - pending transactions table, 32-22
 - recoverer process (RECO) and, 32-24
 - manually overriding in-doubt, 32-9
 - naming distributed, 32-2, 32-10
 - remote, 28-35
 - rollback segments and, 13-24
- TRANSACTIONS initialization parameter, 13-15
- TRANSACTIONS_PER_ROLLBACK_SEGMENT
 - initialization parameter, 13-15
- transmitting archived redo logs, 8-14
 - in normal transmission mode, 8-14
 - in standby transmission mode, 8-14
- transparency
 - location
 - using procedures, 29-30, 29-31, 29-32
 - query, 29-32
 - update, 29-32
- transportable tablespaces, 11-35 to 11-49
 - multiple block sizes, 11-41
- transporting tablespaces between
 - databases, 11-34 to 11-49
- triggers
 - disabling, 21-13
 - distributed query creation, 30-3
 - enabling, 21-13
 - privileges for enabling and disabling, 21-13
- TRUNCATE PARTITION clause, 17-59
- TRUNCATE statement, 21-10
 - DROP STORAGE clause, 21-11
 - REUSE STORAGE clause, 21-11
 - vs. dropping table, 15-23
- TRUNCATE SUBPARTITION clause, 17-61
- tuning
 - analyzing tables, 30-7
 - archiving, 8-19
 - cost-based optimization, 30-5
 - databases, 1-8
- two-phase commit
 - case study, 31-19
 - commit phase, 31-14, 31-23
 - steps in, 31-14
 - described, 28-35
 - distributed transactions, 31-10
 - tracing session tree, 32-5
 - viewing information about, 32-3
 - forget phase, 31-15
 - in-doubt transactions, 31-15
 - automatic resolution, 31-16
 - manual resolution, 31-18
 - SCNs and, 31-19
 - phases, 31-10
 - prepare phase, 31-11
 - abort response, 31-13
 - prepared response, 31-12
 - read-only response, 31-12
 - responses, 31-11

- steps, 31-13
- problems, 32-8
- recognizing read-only nodes, 31-12
- specifying commit point strength, 32-2

U

- undo space management
 - automatic undo management
 - mode, 13-3 to 13-13
 - described, 13-2
 - rollback segment undo mode, 13-13 to 13-29
 - specifying mode, 13-3
- undo tablespaces
 - altering, 13-7
 - creating, 13-6
 - dropping, 13-7
 - estimating space requirements, 13-11
 - initialization parameters for, 13-3
 - monitoring, 13-12
 - PENDING OFFLINE status, 13-8
 - specifying at database creation, 2-24
 - specifying retention period, 13-9
 - starting an instance using, 13-3
 - statistics for, 13-12
 - switching, 13-8
 - used with flashback queries, 13-10
 - user quotas, 13-9
 - viewing information about, 13-11
- UNDO_MANAGEMENT initialization
 - parameter, 2-24
 - starting instance as AUTO, 13-3
- UNDO_RETENTION initialization parameter
 - for undo tablespaces, 13-9
- UNDO_SUPPRESS_ERROR initialization parameter
 - for undo tablespaces, 13-4
- UNDO_TABLESPACE initialization parameter
 - starting an instance using, 13-3
- UNIQUE key constraints
 - associated indexes, 16-11
 - dropping associated indexes, 16-23
 - enabling on creation, 16-11
 - foreign key references when dropped, 21-19
 - indexes associated with, 16-11
- UNLIMITED TABLESPACE privilege, 24-5
- UNRECOVERABLE DATAFILE option
 - ALTER DATABASE statement, 7-20
- UPDATE GLOBAL INDEX clause
 - of ALTER TABLE, 17-26
- UPDATE privilege
 - revoking, 25-18
- updates
 - location transparency and, 28-45
 - transparency, 29-32
- upgrading a database, 2-5
- USER_DB_LINKS view, 29-21
- USER_DUMP_DEST initialization parameter, 5-16
- USER_JOBS view
 - jobs in system, viewing, 10-15
- USER_RESUMABLE view, 14-21
- USER_SEGMENTS view, 11-50
- usernames
 - SYS and SYSTEM, 1-11
- users
 - altering, 24-6
 - assigning profiles to, 24-20
 - assigning tablespace quotas, 11-3
 - assigning unlimited quotas for, 24-4
 - authentication
 - about, 23-2, 24-8
 - changing default roles, 24-7
 - database authentication, 24-9
 - default tablespaces, 24-3
 - dropping, 24-7
 - dropping profiles and, 24-23
 - dropping roles and, 25-10
 - end-user security policies, 23-6
 - enrolling, 1-7
 - enterprise, 24-14, 25-10
 - external authentication, 24-11
 - global, 24-13
 - in a newly created database, 2-32
 - limiting number of, 2-43
 - listing, 24-23
 - listing privileges granted to, 25-27
 - listing roles granted to, 25-28
 - managing, 24-2
 - network authentication, 24-13
 - objects after dropping, 24-7
 - operating system authentication, 24-12

- password security, 23-5
- policies for managing privileges, 23-6
- privileges for changing passwords, 24-6
- privileges for creating, 24-2
- privileges for dropping, 24-8
- proxy authentication and authorization, 24-16
- PUBLIC group, 25-20
- schema-independent, 24-14
- security and, 23-2
- security for general users, 23-4
- session, terminating, 5-23
- specifying user names, 24-3
- tablespace quotas, 24-4
- unique user names, 2-43
- viewing information on, 24-25
- viewing memory use, 24-26
- viewing tablespace quotas, 24-25
- utilities
 - Export, 1-26
 - for the database administrator, 1-26
 - Import, 1-26
 - SQL*Loader, 1-26
- UTLCHAIN.SQL script
 - listing chained rows, 21-7
- UTLCHN1.SQL script
 - listing chained rows, 21-7
- UTLLOCKT.SQL script, 5-15

V

- V\$ARCHIVE view, 8-23
- V\$ARCHIVE_DEST view
 - obtaining destination status, 8-13
- V\$DATABASE view, 8-24
- V\$DATAFILE view, 11-50
- V\$DBFILE view, 2-31
- V\$DBLINK view, 29-25
- V\$DISPATCHER view
 - monitoring shared server dispatchers, 5-8
- V\$DISPATCHER_RATE view
 - monitoring shared server dispatchers, 5-8
- V\$INSTANCE view
 - for database quiesce state, 4-16
- V\$LOG view, 8-23
 - displaying archiving status, 8-23
 - online redo log, 7-20
 - viewing redo data with, 7-20
- V\$LOG_HISTORY view
 - viewing redo data, 7-20
- V\$LOGFILE view, 2-31
 - log file status, 7-17
 - viewing redo data, 7-20
- V\$LOGMNR_CONTENTS view, 9-16
- V\$OBJECT_USAGE view
 - for monitoring index usage, 16-21
- V\$PWFILERS view, 1-24
- V\$QUEUE view
 - monitoring shared server dispatchers, 5-8
- V\$ROLLNAME view
 - finding PENDING OFFLINE segments, 13-29
- V\$ROLLSTAT view
 - finding PENDING OFFLINE segments, 13-29
 - undo segments, 13-12
- V\$SESSION view, 5-23
- V\$SORT_SEGMENT view, 11-50
- V\$SORT_USER view, 11-50
- V\$TEMP_EXTENT_MAP view, 11-50
- V\$TEMP_EXTENT_POOL view, 11-50
- V\$TEMP_SPACE_HEADER view, 11-50
- V\$TEMPFILE view, 11-50
- V\$THREAD view, 7-20
- V\$TIMEZONE_NAMES view
 - time zone table information, 2-29
- V\$TRANSACTION view
 - undo tablespaces information, 13-12
- V\$UNDOSTAT view
 - statistics for undo tablespaces, 13-12
- V\$VERSION view, 1-10
- varrays
 - storage parameters for, 14-12
- verifying blocks
 - redo log files, 7-18
- views
 - creating, 20-2
 - creating with errors, 20-4
 - Database Resource Manager, 27-31
 - DATABASE_PROPERTIES, 2-25
 - DBA_RESUMABLE, 14-21
 - displaying dependencies of, 21-34
 - dropping, 20-10

- file mapping views, 12-23
- for monitoring datafiles, 12-28
- FOR UPDATE clause and, 20-3
- join. *See* join views.
- location transparency using, 29-26
- managing, 20-2, 20-11
- managing privileges with, 29-28
- name resolution in distributed databases, 28-42
- ORDER BY clause and, 20-3
- privileges, 20-2
- privileges for dropping, 20-10
- privileges for recompiling, 21-25
- privileges to replace, 20-10
- recompiling, 21-25
- remote object security, 29-28
- tables, 15-40
- USER_RESUMABLE, 14-21
- VSARCHIVE, 8-23
- VSARCHIVE_DEST, 8-13
- VSDATABASE, 8-24
- VSLOG, 7-20, 8-23
- VSLOG_HISTORY, 7-20
- VSLOGFILE, 7-17, 7-20
- VSOBJECT_USAGE, 16-21
- V\$THREAD, 7-20
- wildcards in, 20-4
- WITH CHECK OPTION, 20-3

W

- wildcards
 - in views, 20-4
- Windows operating system
 - OS audit trail, 26-2, 26-9
- WORM devices
 - and read-only tablespaces, 11-27

