Oracle® Database

Administrator's Guide

11*g* Release 2 (11.2)

E17120-11

August 2011

ORACLE®

Oracle Database Administrator's Guide, 11*g* Release 2 (11.2)

E17120-11

Copyright © 2001, 2011, Oracle and/or its affiliates. All rights reserved.

Primary Author: Steve Fogel

Contributing Author: Caroline Johnston, Sheila Moore, Tony Morales, Padmaja Potineni, Randy Urbano, Lance Ashdown, Janis Greenberg

Contributors: D. Austin, B. Baddepudi, H. Baer, C. Baird, M. Bauer, E. Belden, A. Chaudhry, J. Creighton, S. Datta, M. Dilman, J. Draaijer, M. Fallen, A. Ganesh, GP Gongloor, V. Goorah, S. Gupta, W. Hodak, P. Huey, C. Iyer, B. Khaladkar, B. Krishnan, V. Krishnaswamy, B. Kuchibhotla, S. Kumar, V. Kumar, P. Lane, A. Lee, B. Lee, S. K. Lee, C. Lei, Y. Li, I. Listvinsky, B. Llewellyn, B. Lundhild, S. Lynn, R. Mani, V. Marwah, C. McGregor, M. Minhas, K. Mohan, V. Moore, N. Mukherjee, S. Muthulingam, G. Ngai, A. Raghavan, M. Ramacher, R. Ramkissoon, Y. Sarig, S. Shankar, B. Sinha, V. Srihari, D. Steiner, J. Stern, M. Stewart, M. Subramaniam, A. Tran, A. Tsukerman, K. Umamageswaran, E. Voss, D. M. Wong, P. Youn, W. Zhang

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface xxxv

Audience xxxv
Documentation Accessibility xxxv
Related Documents xxxvi
Conventions xxxvi

What's New in Oracle Database Administrator's Guide? xxxvii

Oracle Database 11*g* Release 2 (11.2.0.2) New Features in the Administrator's Guide xxxvii
Oracle Database 11*g* Release 2 (11.2.0.1) New Features in the Administrator's Guide xxxix

Part I Basic Database Administration

1 Getting Started with Database Administration

Types of Oracle Database Users 1-1
Database Administrators 1-1
Security Officers 1-2
Network Administrators 1-2
Application Developers 1-2
Application Administrators 1-3
Database Users 1-3
Tasks of a Database Administrator 1-3
Task 1: Evaluate the Database Server Hardware 1-3
Task 2: Install the Oracle Database Software 1-4
Task 3: Plan the Database 1-4
Task 4: Create and Open the Database 1-5
Task 5: Back Up the Database 1-5
Task 6: Enroll System Users 1-5
Task 7: Implement the Database Design 1-5
Task 8: Back Up the Fully Functional Database 1-5
Task 9: Tune Database Performance 1-5
Task 10: Download and Install Patches 1-6
Task 11: Roll Out to Additional Hosts 1-6
Submitting Commands and SQL to the Database 1-6
About SQL*Plus 1-7
Connecting to the Database with SQL*Plus 1-7

Identifying Your Oracle Database Software Release 1-12
Release Number Format 1-13
Checking Your Current Release Number 1-13
About Database Administrator Security and Privileges 1-14
The Database Administrator's Operating System Account 1-14
Administrative User Accounts 1-14
Database Administrator Authentication 1-16
Administrative Privileges 1-16
Selecting an Authentication Method for Database Administrators 1-18
Using Operating System Authentication 1-20
Using Password File Authentication 1-21
Creating and Maintaining a Password File 1-22
Creating a Password File with ORAPWD 1-23
Sharing and Disabling the Password File 1-24
Adding Users to a Password File 1-26
Maintaining a Password File 1-27
Data Utilities 1-28

2 Creating and Configuring an Oracle Database

About Creating an Oracle Database 2-1
Considerations Before Creating the Database 2-2
Creating a Database with DBCA 2-5
Creating a Database with Interactive DBCA 2-5
Creating a Database with Noninteractive/Silent DBCA 2-5
Creating a Database with the CREATE DATABASE Statement 2-6
Step 1: Specify an Instance Identifier (SID) 2-7
Step 2: Ensure That the Required Environment Variables Are Set 2-7
Step 3: Choose a Database Administrator Authentication Method 2-7
Step 4: Create the Initialization Parameter File 2-8
Step 5: (Windows Only) Create an Instance 2-9
Step 6: Connect to the Instance 2-9
Step 7: Create a Server Parameter File 2-10
Step 8: Start the Instance 2-10
Step 9: Issue the CREATE DATABASE Statement 2-11
Step 10: Create Additional Tablespaces 2-14
Step 11: Run Scripts to Build Data Dictionary Views 2-14
Step 12: (Optional) Run Scripts to Install Additional Options 2-15
Step 13: Back Up the Database. 2-15
Step 14: (Optional) Enable Automatic Instance Startup 2-15
Specifying CREATE DATABASE Statement Clauses 2-15
Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM 2-16
Creating a Locally Managed SYSTEM Tablespace 2-17
About the SYSAUX Tablespace 2-17
Using Automatic Undo Management: Creating an Undo Tablespace 2-18
Creating a Default Permanent Tablespace 2-18
Creating a Default Temporary Tablespace 2-18
Specifying Oracle Managed Files at Database Creation 2-19

Supporting Bigfile Tablespaces During Database Creation 2-20
Specifying the Database Time Zone and Time Zone File 2-22
Specifying FORCE LOGGING Mode 2-23
Specifying Initialization Parameters 2-24
About Initialization Parameters and Initialization Parameter Files 2-24
Determining the Global Database Name 2-26
Specifying a Fast Recovery Area 2-27
Specifying Control Files 2-28
Specifying Database Block Sizes 2-28
Specifying the Maximum Number of Processes 2-29
Specifying the DDL Lock Timeout 2-30
Specifying the Method of Undo Space Management 2-30
About The COMPATIBLE Initialization Parameter 2-31
Setting the License Parameter 2-31
Managing Initialization Parameters Using a Server Parameter File 2-32
What Is a Server Parameter File? 2-32
Migrating to a Server Parameter File 2-33
Creating a Server Parameter File 2-33
The SPFILE Initialization Parameter 2-35
Changing Initialization Parameter Values 2-35
Clearing Initialization Parameter Values 2-37
Exporting the Server Parameter File 2-37
Backing Up the Server Parameter File 2-37
Recovering a Lost or Damaged Server Parameter File 2-38
Viewing Parameter Settings 2-38
Managing Application Workloads with Database Services 2-39
About Database Services 2-39
Creating Database Services 2-41
Database Service Data Dictionary Views 2-42
Considerations After Creating a Database 2-42
Some Security Considerations 2-42
Enabling Transparent Data Encryption 2-43
Creating a Secure External Password Store 2-44
Installing the Oracle Database Sample Schemas 2-44
Dropping a Database 2-44
Database Data Dictionary Views 2-45

3 Starting Up and Shutting Down

Starting Up a Database 3-1
About Database Startup Options 3-1
Specifying Initialization Parameters at Startup 3-2
About Automatic Startup of Database Services 3-5
Preparing to Start Up an Instance 3-5
Starting Up an Instance 3-6
Altering Database Availability 3-9
Mounting a Database to an Instance 3-9
Opening a Closed Database 3-10

Opening a Database in Read-Only Mode 3-10
Restricting Access to an Open Database 3-11
Shutting Down a Database 3-11
Shutting Down with the Normal Mode 3-12
Shutting Down with the Immediate Mode 3-12
Shutting Down with the Transactional Mode 3-13
Shutting Down with the Abort Mode 3-13
Shutdown Timeout 3-14
Quiescing a Database 3-14
Placing a Database into a Quiesced State 3-15
Restoring the System to Normal Operation 3-16
Viewing the Quiesce State of an Instance 3-16
Suspending and Resuming a Database 3-16

4 Configuring Automatic Restart of an Oracle Database

About Oracle Restart 4-1
Oracle Restart Overview 4-1
About Startup Dependencies 4-2
About Starting and Stopping Components with Oracle Restart 4-3
About Starting and Stopping Oracle Restart 4-3
Oracle Restart Configuration 4-4
Oracle Restart Integration with Oracle Data Guard 4-5
Fast Application Notification with Oracle Restart 4-6
Configuring Oracle Restart 4-9
Preparing to Run SRVCTL 4-10
Obtaining Help for SRVCTL 4-11
Adding Components to the Oracle Restart Configuration 4-11
Removing Components from the Oracle Restart Configuration 4-14
Disabling and Enabling Oracle Restart Management for a Component 4-14
Viewing Component Status 4-15
Viewing the Oracle Restart Configuration for a Component 4-16
Modifying the Oracle Restart Configuration for a Component 4-16
Managing Environment Variables in the Oracle Restart Configuration 4-17
Creating and Deleting Database Services with SRVCTL 4-19
Enabling FAN Events in an Oracle Restart Environment 4-19
Automating the Failover of Connections Between Primary and Standby Databases 4-20
Enabling Clients for Fast Connection Failover 4-21
Starting and Stopping Components Managed by Oracle Restart 4-25
Starting and Stopping Components Managed by Oracle Restart with SRVCTL 4-25
Starting a Database Managed by Oracle Restart with Oracle Enterprise Manager 4-26
Stopping and Restarting Oracle Restart for Maintenance Operations 4-27
SRVCTL Command Reference for Oracle Restart 4-30
add 4-32
config 4-38
disable 4-41
enable 4-44
getenv 4-47

modify 4-49
remove 4-53
setenv 4-56
start 4-58
status 4-62
stop 4-65
unsetenv 4-69
CRSCTL Command Reference 4-71
check 4-72
config 4-73
disable 4-74
enable 4-75
start 4-76
stop 4-77

5 Managing Processes

About Dedicated and Shared Server Processes 5-1
Dedicated Server Processes 5-1
Shared Server Processes 5-2
About Database Resident Connection Pooling 5-4
Comparing DRCP to Dedicated Server and Shared Server 5-5
Restrictions on Using Database Resident Connection Pooling 5-6
Configuring Oracle Database for Shared Server 5-6
Initialization Parameters for Shared Server 5-6
Memory Management for Shared Server 5-7
Enabling Shared Server 5-7
Configuring Dispatchers 5-10
Disabling Shared Server 5-14
Shared Server Data Dictionary Views 5-15
Configuring Database Resident Connection Pooling 5-15
Enabling Database Resident Connection Pooling 5-16
Configuring the Connection Pool for Database Resident Connection Pooling 5-17
Data Dictionary Views for Database Resident Connection Pooling 5-19
About Oracle Database Background Processes 5-19
Managing Processes for Parallel SQL Execution 5-20
About Parallel Execution Servers 5-21
Altering Parallel Execution for a Session 5-21
Managing Processes for External Procedures 5-22
About External Procedures 5-22
DBA Tasks to Enable External Procedure Calls 5-23
Terminating Sessions 5-23
Identifying Which Session to Terminate 5-24
Terminating an Active Session 5-24
Terminating an Inactive Session 5-25
Process and Session Data Dictionary Views 5-25

6 Managing Memory

About Memory Management ... 6-1
Memory Architecture Overview ... 6-2
Using Automatic Memory Management ... 6-3
About Automatic Memory Management ... 6-4
Enabling Automatic Memory Management ... 6-4
Monitoring and Tuning Automatic Memory Management ... 6-6
Configuring Memory Manually ... 6-7
Using Automatic Shared Memory Management ... 6-8
Using Manual Shared Memory Management ... 6-14
Using Automatic PGA Memory Management ... 6-19
Using Manual PGA Memory Management ... 6-20
Configuring Database Smart Flash Cache ... 6-21
When to Configure the Flash Cache ... 6-21
Sizing the Flash Cache ... 6-21
Tuning Memory for the Flash Cache ... 6-22
Flash Cache Initialization Parameters ... 6-22
Flash Cache in an Oracle Real Applications Clusters Environment ... 6-23
Memory Management Reference ... 6-23
Platforms That Support Automatic Memory Management ... 6-23
Memory Management Data Dictionary Views ... 6-23

7 Managing Users and Securing the Database

The Importance of Establishing a Security Policy for Your Database ... 7-1
Managing Users and Resources ... 7-1
Managing User Privileges and Roles ... 7-2
Auditing Database Use ... 7-2
Predefined User Accounts ... 7-2

8 Monitoring Database Operations

Monitoring Errors and Alerts ... 8-1
Monitoring Errors with Trace Files and the Alert Log ... 8-1
Monitoring Database Operations with Server-Generated Alerts ... 8-4
Monitoring Performance ... 8-6
Monitoring Locks ... 8-7
Monitoring Wait Events ... 8-7
Performance Monitoring Data Dictionary Views ... 8-7

9 Managing Diagnostic Data

About the Oracle Database Fault Diagnosability Infrastructure ... 9-1
Fault Diagnosability Infrastructure Overview ... 9-1
About Incidents and Problems ... 9-3
Fault Diagnosability Infrastructure Components ... 9-5
Structure, Contents, and Location of the Automatic Diagnostic Repository ... 9-7
Investigating, Reporting, and Resolving a Problem ... 9-11
Roadmap—Investigating, Reporting, and Resolving a Problem ... 9-11

Task 1 – View Critical Error Alerts in Enterprise Manager 9-13
Task 2 –View Problem Details 9-14
Task 3 – (Optional) Gather Additional Diagnostic Information 9-14
Task 4 – (Optional) Create a Service Request 9-14
Task 5 – Package and Upload Diagnostic Data to Oracle Support 9-15
Task 6 – Track the Service Request and Implement Any Repairs 9-16
Task 7 – Close Incidents 9-17
Viewing Problems with the Enterprise Manager Support Workbench 9-18
Creating a User-Reported Problem 9-19
Viewing the Alert Log 9-20
Finding Trace Files 9-21
Running Health Checks with Health Monitor 9-21
About Health Monitor 9-22
Running Health Checks Manually 9-23
Viewing Checker Reports 9-24
Health Monitor Views 9-27
Health Check Parameters Reference 9-28
Repairing SQL Failures with the SQL Repair Advisor 9-28
About the SQL Repair Advisor 9-29
Running the SQL Repair Advisor 9-29
Viewing, Disabling, or Removing a SQL Patch 9-30
Repairing Data Corruptions with the Data Recovery Advisor 9-31
Creating, Editing, and Uploading Custom Incident Packages 9-32
About Incident Packages 9-33
Packaging and Uploading Problems with Custom Packaging 9-35
Viewing and Modifying Incident Packages 9-39
Creating, Editing, and Uploading Correlated Packages 9-45
Deleting Correlated Packages 9-45
Setting Incident Packaging Preferences 9-46

10 Managing Oracle Enterprise Manager Database Control

Database Control Overview 10-1
Understanding the Enterprise Manager Directories 10-1
Configuring Database Control During and After Installation 10-3
Overview of Configuration Options 10-3
Configuring Database Control During Installation 10-4
Configuring Database Control with DBCA 10-4
Configuring Database Control Using EMCA 10-5
Using an Input File for EMCA Parameters 10-10
Specifying the Ports Used by Database Control 10-11
Using EMCA With Oracle RAC 10-12
Administering Database Control 10-15
Starting and Stopping the Database Control Process 10-15
Accessing the Database Home Page 10-16
Creating Database Control Administrative Users 10-17
Granting Access to Database Control for Non-Administrative Users 10-18
Setting Database Control Preferences 10-18

Managing Database Control in an Oracle RAC Environment 10-21
Verifying that Database Control is Running in an Oracle RAC Environment 10-21
Backing Up the emkey.ora File 10-22
Deleting and Creating Database Control Objects 10-22
Deleting Database Control Objects 10-22
Creating Database Control Objects 10-24
Re-Creating Or Reconfiguring Database Control 10-25
EMCA Troubleshooting Tips 10-25
Using EMCA After Changing the Database Listener Port 10-25
Upgrading 11*g* Release 2 Grid Control Agents 10-26
Using EMCA When the Database Host Name or IP Address Changes 10-26
Using EMCA When the TNS Configuration Is Changed 10-26
Using EMCA for Desktop Class Installations 10-26
Handling EMCA Error Messages 10-27

Part II Oracle Database Structure and Storage

11 Managing Control Files

What Is a Control File? 11-1
Guidelines for Control Files 11-2
Provide Filenames for the Control Files 11-2
Multiplex Control Files on Different Disks 11-2
Back Up Control Files 11-3
Manage the Size of Control Files 11-3
Creating Control Files 11-3
Creating Initial Control Files 11-3
Creating Additional Copies, Renaming, and Relocating Control Files 11-4
Creating New Control Files 11-4
Troubleshooting After Creating Control Files 11-7
Checking for Missing or Extra Files 11-7
Handling Errors During CREATE CONTROLFILE 11-7
Backing Up Control Files 11-8
Recovering a Control File Using a Current Copy 11-8
Recovering from Control File Corruption Using a Control File Copy 11-8
Recovering from Permanent Media Failure Using a Control File Copy 11-8
Dropping Control Files 11-9
Control Files Data Dictionary Views 11-9

12 Managing the Redo Log

What Is the Redo Log? 12-1
Redo Threads 12-1
Redo Log Contents 12-2
How Oracle Database Writes to the Redo Log 12-2
Planning the Redo Log 12-4
Multiplexing Redo Log Files 12-4
Placing Redo Log Members on Different Disks 12-6

Planning the Size of Redo Log Files 12-7
Planning the Block Size of Redo Log Files 12-7
Choosing the Number of Redo Log Files 12-8
Controlling Archive Lag 12-9
Creating Redo Log Groups and Members 12-10
Creating Redo Log Groups 12-10
Creating Redo Log Members 12-11
Relocating and Renaming Redo Log Members 12-11
Dropping Redo Log Groups and Members 12-12
Dropping Log Groups 12-13
Dropping Redo Log Members 12-13
Forcing Log Switches 12-14
Verifying Blocks in Redo Log Files 12-14
Clearing a Redo Log File 12-15
Redo Log Data Dictionary Views 12-16

13 Managing Archived Redo Logs

What Is the Archived Redo Log? 13-1
Choosing Between NOARCHIVELOG and ARCHIVELOG Mode 13-2
Running a Database in NOARCHIVELOG Mode 13-2
Running a Database in ARCHIVELOG Mode 13-3
Controlling Archiving 13-4
Setting the Initial Database Archiving Mode 13-4
Changing the Database Archiving Mode 13-4
Performing Manual Archiving 13-5
Adjusting the Number of Archiver Processes 13-5
Specifying Archive Destinations 13-6
Setting Initialization Parameters for Archive Destinations 13-6
Understanding Archive Destination Status 13-9
Specifying Alternate Destinations 13-10
About Log Transmission Modes 13-10
Normal Transmission Mode 13-10
Standby Transmission Mode 13-10
Managing Archive Destination Failure 13-11
Specifying the Minimum Number of Successful Destinations 13-11
Rearchiving to a Failed Destination 13-12
Controlling Trace Output Generated by the Archivelog Process 13-13
Viewing Information About the Archived Redo Log 13-14
Archived Redo Logs Views 13-14
The ARCHIVE LOG LIST Command 13-15

14 Managing Tablespaces

Guidelines for Managing Tablespaces 14-1
Using Multiple Tablespaces 14-2
Assigning Tablespace Quotas to Users 14-2
Creating Tablespaces 14-2

Locally Managed Tablespaces 14-3
Bigfile Tablespaces 14-6
Compressed Tablespaces 14-8
Encrypted Tablespaces 14-8
Temporary Tablespaces 14-10
Multiple Temporary Tablespaces: Using Tablespace Groups 14-13
Specifying Nonstandard Block Sizes for Tablespaces 14-14
Controlling the Writing of Redo Records 14-15
Altering Tablespace Availability 14-15
Taking Tablespaces Offline 14-15
Bringing Tablespaces Online 14-17
Using Read-Only Tablespaces 14-17
Making a Tablespace Read-Only 14-18
Making a Read-Only Tablespace Writable 14-19
Creating a Read-Only Tablespace on a WORM Device 14-20
Delaying the Opening of Datafiles in Read-Only Tablespaces 14-20
Altering and Maintaining Tablespaces 14-21
Increasing the Size of a Tablespace 14-21
Altering a Locally Managed Tablespace 14-21
Altering a Bigfile Tablespace 14-21
Altering a Locally Managed Temporary Tablespace 14-22
Shrinking a Locally Managed Temporary Tablespace 14-23
Renaming Tablespaces 14-23
Dropping Tablespaces 14-24
Managing the SYSAUX Tablespace 14-25
Monitoring Occupants of the SYSAUX Tablespace 14-25
Moving Occupants Out Of or Into the SYSAUX Tablespace 14-25
Controlling the Size of the SYSAUX Tablespace 14-25
Diagnosing and Repairing Locally Managed Tablespace Problems 14-26
Scenario 1: Fixing Bitmap When Allocated Blocks are Marked Free (No Overlap) 14-28
Scenario 2: Dropping a Corrupted Segment 14-28
Scenario 3: Fixing Bitmap Where Overlap is Reported 14-28
Scenario 4: Correcting Media Corruption of Bitmap Blocks 14-28
Scenario 5: Migrating from a Dictionary-Managed to a Locally Managed Tablespace 14-29
Migrating the SYSTEM Tablespace to a Locally Managed Tablespace 14-29
Transporting Tablespaces Between Databases 14-30
Introduction to Transportable Tablespaces 14-30
About Transporting Tablespaces Across Platforms 14-31
Limitations on Transportable Tablespace Use 14-32
Compatibility Considerations for Transportable Tablespaces 14-34
Transporting Tablespaces Between Databases: A Procedure and Example 14-35
Using Transportable Tablespaces: Scenarios 14-44
Moving Databases Across Platforms Using Transportable Tablespaces 14-47
Tablespace Data Dictionary Views 14-47
Example 1: Listing Tablespaces and Default Storage Parameters 14-48
Example 2: Listing the Datafiles and Associated Tablespaces of a Database 14-48
Example 3: Displaying Statistics for Free Space (Extents) of Each Tablespace 14-49

15 Managing Datafiles and Tempfiles

Guidelines for Managing Datafiles 15-1
Determine the Number of Datafiles 15-2
Determine the Size of Datafiles 15-3
Place Datafiles Appropriately 15-4
Store Datafiles Separate from Redo Log Files 15-4
Creating Datafiles and Adding Datafiles to a Tablespace 15-4
Changing Datafile Size 15-5
Enabling and Disabling Automatic Extension for a Datafile 15-5
Manually Resizing a Datafile 15-6
Altering Datafile Availability 15-6
Bringing Datafiles Online or Taking Offline in ARCHIVELOG Mode 15-7
Taking Datafiles Offline in NOARCHIVELOG Mode 15-7
Altering the Availability of All Datafiles or Tempfiles in a Tablespace 15-8
Renaming and Relocating Datafiles 15-8
Procedures for Renaming and Relocating Datafiles in a Single Tablespace 15-9
Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces 15-10
Dropping Datafiles 15-11
Verifying Data Blocks in Datafiles 15-12
Copying Files Using the Database Server 15-12
Copying a File on a Local File System 15-13
Third-Party File Transfer 15-14
File Transfer and the DBMS_SCHEDULER Package 15-14
Advanced File Transfer Mechanisms 15-15
Mapping Files to Physical Devices 15-15
Overview of Oracle Database File Mapping Interface 15-16
How the Oracle Database File Mapping Interface Works 15-16
Using the Oracle Database File Mapping Interface 15-19
File Mapping Examples 15-22
Datafiles Data Dictionary Views 15-24

16 Managing Undo

What Is Undo? 16-1
Introduction to Automatic Undo Management 16-2
Overview of Automatic Undo Management 16-2
About the Undo Retention Period 16-3
Setting the Minimum Undo Retention Period 16-5
Sizing a Fixed-Size Undo Tablespace 16-6
The Undo Advisor PL/SQL Interface 16-7
Managing Undo Tablespaces 16-7
Creating an Undo Tablespace 16-8
Altering an Undo Tablespace 16-9
Dropping an Undo Tablespace 16-9
Switching Undo Tablespaces 16-10
Establishing User Quotas for Undo Space 16-10
Managing Space Threshold Alerts for the Undo Tablespace 16-11

Migrating to Automatic Undo Management 16-11
Undo Space Data Dictionary Views 16-11

17 Using Oracle Managed Files

What Are Oracle Managed Files? 17-1
Who Can Use Oracle Managed Files? 17-2
Benefits of Using Oracle Managed Files 17-3
Oracle Managed Files and Existing Functionality 17-3
Enabling the Creation and Use of Oracle Managed Files 17-3
Setting the DB_CREATE_FILE_DEST Initialization Parameter 17-4
Setting the DB_RECOVERY_FILE_DEST Parameter 17-5
Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameters 17-5
Creating Oracle Managed Files 17-5
How Oracle Managed Files Are Named 17-6
Creating Oracle Managed Files at Database Creation 17-7
Creating Datafiles for Tablespaces Using Oracle Managed Files 17-12
Creating Tempfiles for Temporary Tablespaces Using Oracle Managed Files 17-14
Creating Control Files Using Oracle Managed Files 17-15
Creating Redo Log Files Using Oracle Managed Files 17-16
Creating Archived Logs Using Oracle Managed Files 17-17
Operation of Oracle Managed Files 17-17
Dropping Datafiles and Tempfiles 17-18
Dropping Redo Log Files 17-18
Renaming Files 17-18
Managing Standby Databases 17-19
Scenarios for Using Oracle Managed Files 17-19
Scenario 1: Create and Manage a Database with Multiplexed Redo Logs 17-19
Scenario 2: Create and Manage a Database with Database and Fast Recovery Areas 17-22
Scenario 3: Adding Oracle Managed Files to an Existing Database 17-23

Part III Schema Objects

18 Managing Schema Objects

Creating Multiple Tables and Views in a Single Operation 18-1
Analyzing Tables, Indexes, and Clusters 18-2
Using DBMS_STATS to Collect Table and Index Statistics 18-2
Validating Tables, Indexes, Clusters, and Materialized Views 18-3
Listing Chained Rows of Tables and Clusters 18-4
Truncating Tables and Clusters 18-5
Using DELETE 18-6
Using DROP and CREATE 18-6
Using TRUNCATE 18-6
Enabling and Disabling Triggers 18-7
Enabling Triggers 18-9
Disabling Triggers 18-9
Managing Integrity Constraints 18-9

Integrity Constraint States 18-10
Setting Integrity Constraints Upon Definition 18-11
Modifying, Renaming, or Dropping Existing Integrity Constraints 18-12
Deferring Constraint Checks 18-14
Reporting Constraint Exceptions 18-14
Viewing Constraint Information 18-16
Renaming Schema Objects 18-16
Managing Object Dependencies 18-17
About Object Dependencies and Object Invalidation 18-17
Manually Recompiling Invalid Objects with DDL 18-18
Manually Recompiling Invalid Objects with PL/SQL Package Procedures 18-19
Managing Object Name Resolution 18-19
Switching to a Different Schema 18-21
Managing Editions 18-21
About Editions and Edition-Based Redefinition 18-21
DBA Tasks for Edition-Based Redefinition 18-22
Setting the Database Default Edition 18-22
Querying the Database Default Edition 18-22
Setting the Edition Attribute of a Database Service 18-23
Using an Edition 18-24
Editions Data Dictionary Views 18-24
Displaying Information About Schema Objects 18-24
Using a PL/SQL Package to Display Information About Schema Objects 18-25
Schema Objects Data Dictionary Views 18-25

19 Managing Space for Schema Objects

Managing Tablespace Alerts 19-1
Setting Alert Thresholds 19-2
Viewing Alerts 19-3
Limitations 19-3
Managing Resumable Space Allocation 19-4
Resumable Space Allocation Overview 19-4
Enabling and Disabling Resumable Space Allocation 19-7
Using a LOGON Trigger to Set Default Resumable Mode 19-8
Detecting Suspended Statements 19-8
Operation-Suspended Alert 19-10
Resumable Space Allocation Example: Registering an AFTER SUSPEND Trigger 19-10
Reclaiming Wasted Space 19-12
Understanding Reclaimable Unused Space 19-12
Using the Segment Advisor 19-12
Shrinking Database Segments Online 19-26
Deallocating Unused Space 19-28
Dropping Unused Object Storage 19-28
Understanding Space Usage of Data Types 19-29
Displaying Information About Space Usage for Schema Objects 19-29
Using PL/SQL Packages to Display Information About Schema Object Space Usage 19-29
Schema Objects Space Usage Data Dictionary Views 19-30

Capacity Planning for Database Objects 19-32
Estimating the Space Use of a Table 19-32
Estimating the Space Use of an Index 19-33
Obtaining Object Growth Trends 19-33

20 Managing Tables

About Tables 20-1
Guidelines for Managing Tables 20-2
Design Tables Before Creating Them 20-2
Specify the Type of Table to Create 20-3
Specify the Location of Each Table 20-4
Consider Parallelizing Table Creation 20-4
Consider Using NOLOGGING When Creating Tables 20-4
Consider Using Table Compression 20-5
Consider Encrypting Columns That Contain Sensitive Data 20-8
Understand Deferred Segment Creation 20-9
Materializing Segments 20-11
Estimate Table Size and Plan Accordingly 20-12
Restrictions to Consider When Creating Tables 20-12
Creating Tables 20-12
Example: Creating a Table 20-13
Creating a Temporary Table 20-14
Parallelizing Table Creation 20-15
Loading Tables 20-16
Methods for Loading Tables 20-16
Improving INSERT Performance with Direct-Path INSERT 20-17
Using Conventional Inserts to Load Tables 20-22
Avoiding Bulk INSERT Failures with DML Error Logging 20-22
Automatically Collecting Statistics on Tables 20-26
Altering Tables 20-26
Reasons for Using the ALTER TABLE Statement 20-27
Altering Physical Attributes of a Table 20-28
Moving a Table to a New Segment or Tablespace 20-28
Manually Allocating Storage for a Table 20-29
Modifying an Existing Column Definition 20-29
Adding Table Columns 20-29
Renaming Table Columns 20-30
Dropping Table Columns 20-30
Placing a Table in Read-Only Mode 20-32
Redefining Tables Online 20-33
Features of Online Table Redefinition 20-34
Performing Online Redefinition with DBMS_REDEFINITION 20-34
Results of the Redefinition Process 20-38
Performing Intermediate Synchronization 20-39
Aborting Online Table Redefinition and Cleaning Up After Errors 20-39
Restrictions for Online Redefinition of Tables 20-39
Online Redefinition of a Single Partition 20-41

Online Table Redefinition Examples 20-42
Privileges Required for the DBMS_REDEFINITION Package 20-48
Researching and Reversing Erroneous Table Changes 20-48
Recovering Tables Using Oracle Flashback Table 20-49
Dropping Tables 20-50
Using Flashback Drop and Managing the Recycle Bin 20-50
What Is the Recycle Bin? 20-51
Enabling and Disabling the Recycle Bin 20-52
Viewing and Querying Objects in the Recycle Bin 20-52
Purging Objects in the Recycle Bin 20-53
Restoring Tables from the Recycle Bin 20-54
Managing Index-Organized Tables 20-55
What Are Index-Organized Tables? 20-55
Creating Index-Organized Tables 20-56
Maintaining Index-Organized Tables 20-60
Creating Secondary Indexes on Index-Organized Tables 20-62
Analyzing Index-Organized Tables 20-63
Using the ORDER BY Clause with Index-Organized Tables 20-64
Converting Index-Organized Tables to Regular Tables 20-64
Managing External Tables 20-64
About External Tables 20-65
Creating External Tables 20-66
Altering External Tables 20-68
Preprocessing External Tables 20-69
Dropping External Tables 20-70
System and Object Privileges for External Tables 20-70
Tables Data Dictionary Views 20-71

21 Managing Indexes

About Indexes 21-1
Guidelines for Managing Indexes 21-2
Create Indexes After Inserting Table Data 21-2
Index the Correct Tables and Columns 21-3
Order Index Columns for Performance 21-3
Limit the Number of Indexes for Each Table 21-4
Drop Indexes That Are No Longer Required 21-4
Indexes and Deferred Segment Creation 21-4
Estimate Index Size and Set Storage Parameters 21-4
Specify the Tablespace for Each Index 21-5
Consider Parallelizing Index Creation 21-5
Consider Creating Indexes with NOLOGGING 21-5
Understand When to Use Unusable or Invisible Indexes 21-6
Consider Costs and Benefits of Coalescing or Rebuilding Indexes 21-7
Consider Cost Before Disabling or Dropping Constraints 21-8
Creating Indexes 21-8
Creating an Index Explicitly 21-9
Creating a Unique Index Explicitly 21-9

Creating an Index Associated with a Constraint 21-9
Creating a Large Index 21-11
Creating an Index Online 21-11
Creating a Function-Based Index 21-11
Creating a Key-Compressed Index 21-12
Creating an Unusable Index 21-13
Creating an Invisible Index 21-14
Altering Indexes 21-15
Altering Storage Characteristics of an Index 21-16
Rebuilding an Existing Index 21-16
Making an Index Unusable 21-17
Making an Index Invisible 21-18
Renaming an Index 21-19
Monitoring Index Usage 21-19
Monitoring Space Use of Indexes 21-20
Dropping Indexes 21-20
Indexes Data Dictionary Views 21-21

22 Managing Clusters

About Clusters 22-1
Guidelines for Managing Clusters 22-2
Choose Appropriate Tables for the Cluster 22-3
Choose Appropriate Columns for the Cluster Key 22-3
Specify the Space Required by an Average Cluster Key and Its Associated Rows 22-3
Specify the Location of Each Cluster and Cluster Index Rows 22-4
Estimate Cluster Size and Set Storage Parameters 22-4
Creating Clusters 22-4
Creating Clustered Tables 22-5
Creating Cluster Indexes 22-5
Altering Clusters 22-6
Altering Clustered Tables 22-6
Altering Cluster Indexes 22-7
Dropping Clusters 22-7
Dropping Clustered Tables 22-8
Dropping Cluster Indexes 22-8
Clusters Data Dictionary Views 22-8

23 Managing Hash Clusters

About Hash Clusters 23-1
When to Use Hash Clusters 23-2
Situations Where Hashing Is Useful 23-2
Situations Where Hashing Is Not Advantageous 23-2
Creating Hash Clusters 23-2
Creating a Sorted Hash Cluster 23-3
Creating Single-Table Hash Clusters 23-6
Controlling Space Use Within a Hash Cluster 23-6
Estimating Size Required by Hash Clusters 23-8

Altering Hash Clusters ... 23-9
Dropping Hash Clusters ... 23-9
Hash Clusters Data Dictionary Views ... 23-9

24 Managing Views, Sequences, and Synonyms

Managing Views ... 24-1
About Views ... 24-1
Creating Views ... 24-2
Replacing Views ... 24-4
Using Views in Queries ... 24-4
Updating a Join View ... 24-6
Altering Views ... 24-12
Dropping Views ... 24-12
Managing Sequences ... 24-12
About Sequences ... 24-12
Creating Sequences ... 24-13
Altering Sequences ... 24-13
Using Sequences ... 24-14
Dropping Sequences ... 24-16
Managing Synonyms ... 24-17
About Synonyms ... 24-17
Creating Synonyms ... 24-17
Using Synonyms in DML Statements ... 24-18
Dropping Synonyms ... 24-18
Views, Synonyms, and Sequences Data Dictionary Views ... 24-19

25 Repairing Corrupted Data

Options for Repairing Data Block Corruption ... 25-1
About the DBMS_REPAIR Package ... 25-1
DBMS_REPAIR Procedures ... 25-2
Limitations and Restrictions ... 25-2
Using the DBMS_REPAIR Package ... 25-2
Task 1: Detect and Report Corruptions ... 25-3
Task 2: Evaluate the Costs and Benefits of Using DBMS_REPAIR ... 25-4
Task 3: Make Objects Usable ... 25-5
Task 4: Repair Corruptions and Rebuild Lost Data ... 25-5
DBMS_REPAIR Examples ... 25-5
Examples: Building a Repair Table or Orphan Key Table ... 25-6
Example: Detecting Corruption ... 25-7
Example: Fixing Corrupt Blocks ... 25-8
Example: Finding Index Entries Pointing to Corrupt Data Blocks ... 25-9
Example: Skipping Corrupt Blocks ... 25-9

Part IV Database Resource Management and Task Scheduling

26 Managing Automated Database Maintenance Tasks

About Automated Maintenance Tasks 26-1
About Maintenance Windows 26-2
Configuring Automated Maintenance Tasks 26-3
Enabling and Disabling Maintenance Tasks for all Maintenance Windows 26-3
Enabling and Disabling Maintenance Tasks for Specific Maintenance Windows 26-4
Configuring Maintenance Windows 26-4
Modifying a Maintenance Window 26-4
Creating a New Maintenance Window 26-4
Removing a Maintenance Window 26-5
Configuring Resource Allocations for Automated Maintenance Tasks 26-5
About Resource Allocations for Automated Maintenance Tasks 26-6
Changing Resource Allocations for Automated Maintenance Tasks 26-6
Automated Maintenance Tasks Reference 26-7
Predefined Maintenance Windows 26-7
Automated Maintenance Tasks Database Dictionary Views 26-7

27 Managing Resources with Oracle Database Resource Manager

About Oracle Database Resource Manager 27-1
What Solutions Does the Resource Manager Provide for Workload Management? 27-2
Elements of the Resource Manager 27-3
About Resource Manager Administration Privileges 27-6
Assigning Sessions to Resource Consumer Groups 27-7
Overview of Assigning Sessions to Resource Consumer Groups 27-8
Assigning an Initial Resource Consumer Group 27-8
Specifying Session-to–Consumer Group Mapping Rules 27-8
Switching Resource Consumer Groups 27-12
Specifying Automatic Resource Consumer Group Switching 27-14
Granting and Revoking the Switch Privilege 27-16
The Types of Resources Managed by the Resource Manager 27-17
CPU 27-18
Degree of Parallelism Limit 27-20
Parallel Target Percentage 27-21
Parallel Queue Timeout 27-22
Active Session Pool with Queuing 27-23
Automatic Consumer Group Switching 27-23
Canceling SQL and Terminating Sessions 27-23
Execution Time Limit 27-24
Undo Pool 27-24
Idle Time Limit 27-24
Creating a Simple Resource Plan 27-24
Creating a Complex Resource Plan 27-25
About the Pending Area 27-26
Creating a Pending Area 27-27
Creating Resource Consumer Groups 27-27
Creating a Resource Plan 27-28
Creating Resource Plan Directives 27-29

Validating the Pending Area ... 27-33
Submitting the Pending Area ... 27-35
Clearing the Pending Area ... 27-35
Enabling Oracle Database Resource Manager and Switching Plans ... 27-35
Putting It All Together: Oracle Database Resource Manager Examples ... 27-37
Multilevel Plan Example ... 27-37
Examples of Using the Maximum Utilization Limit Attribute ... 27-39
Example of Using Several Resource Allocation Methods ... 27-45
Example of Managing Parallel Statements Using Directive Attributes ... 27-45
An Oracle-Supplied Mixed Workload Plan ... 27-47
Managing Multiple Database Instances on a Single Server ... 27-48
About Instance Caging ... 27-48
Enabling Instance Caging ... 27-49
Maintaining Consumer Groups, Plans, and Directives ... 27-49
Updating a Consumer Group ... 27-50
Deleting a Consumer Group ... 27-50
Updating a Plan ... 27-50
Deleting a Plan ... 27-50
Updating a Resource Plan Directive ... 27-51
Deleting a Resource Plan Directive ... 27-51
Viewing Database Resource Manager Configuration and Status ... 27-51
Viewing Consumer Groups Granted to Users or Roles ... 27-52
Viewing Plan Information ... 27-52
Viewing Current Consumer Groups for Sessions ... 27-52
Viewing the Currently Active Plans ... 27-53
Monitoring Oracle Database Resource Manager ... 27-53
Interacting with Operating-System Resource Control ... 27-57
Guidelines for Using Operating-System Resource Control ... 27-57
Oracle Database Resource Manager Reference ... 27-58
Predefined Resource Plans and Consumer Groups ... 27-58
Predefined Consumer Group Mapping Rules ... 27-60
Resource Manager Data Dictionary Views ... 27-60

28 Oracle Scheduler Concepts

Overview of Oracle Scheduler ... 28-1
About Jobs and Supporting Scheduler Objects ... 28-3
Programs ... 28-4
Schedules ... 28-4
Jobs ... 28-5
Destinations ... 28-6
File Watchers ... 28-7
Credentials ... 28-7
Chains ... 28-8
Job Classes ... 28-10
Windows ... 28-11
Groups ... 28-14
More About Jobs ... 28-15

Job Categories 28-15
Job Instances 28-21
Job Arguments 28-22
How Programs, Jobs, and Schedules are Related 28-22
Scheduler Architecture 28-23
The Job Table 28-23
The Job Coordinator 28-23
How Jobs Execute 28-24
After Jobs Complete 28-25
Using the Scheduler in Real Application Clusters Environments 28-25
Scheduler Support for Oracle Data Guard 28-27
Oracle Scheduler and Editions 28-27

29 Scheduling Jobs with Oracle Scheduler

About Scheduler Objects and Their Naming 29-1
Creating, Running, and Managing Jobs 29-2
Job Tasks and Their Procedures 29-2
Creating Jobs 29-2
Altering Jobs 29-15
Running Jobs 29-15
Stopping Jobs 29-16
Dropping Jobs 29-17
Disabling Jobs 29-19
Enabling Jobs 29-20
Copying Jobs 29-20
Viewing stdout and stderr for External Jobs 29-20
Creating and Managing Programs to Define Jobs 29-21
Program Tasks and Their Procedures 29-21
Creating Programs 29-21
Altering Programs 29-23
Dropping Programs 29-23
Disabling Programs 29-24
Enabling Programs 29-24
Creating and Managing Schedules to Define Jobs 29-24
Schedule Tasks and Their Procedures 29-25
Creating Schedules 29-25
Altering Schedules 29-25
Dropping Schedules 29-26
Setting the Repeat Interval 29-26
Using Events to Start Jobs 29-30
About Events 29-30
Starting Jobs with Events Raised by Your Application 29-31
Starting a Job When a File Arrives on a System 29-35
Creating and Managing Job Chains 29-41
Chain Tasks and Their Procedures 29-42
Creating Chains 29-42
Defining Chain Steps 29-43

Adding Rules to a Chain 29-44
Enabling Chains 29-47
Creating Jobs for Chains 29-48
Dropping Chains 29-48
Running Chains 29-49
Dropping Chain Rules 29-49
Disabling Chains 29-50
Dropping Chain Steps 29-50
Stopping Chains 29-50
Stopping Individual Chain Steps 29-50
Pausing Chains 29-51
Skipping Chain Steps 29-52
Running Part of a Chain 29-52
Monitoring Running Chains 29-52
Handling Stalled Chains 29-53
Prioritizing Jobs 29-53
Managing Job Priorities with Job Classes 29-53
Setting Relative Job Priorities Within a Job Class 29-55
Managing Job Scheduling and Job Priorities with Windows 29-55
Managing Job Scheduling and Job Priorities with Window Groups 29-60
Allocating Resources Among Jobs Using Resource Manager 29-63
Example of Resource Allocation for Jobs 29-64
Monitoring Jobs 29-64
Viewing the Job Log 29-65
Monitoring Multiple Destination Jobs 29-67
Monitoring Job State with Events Raised by the Scheduler 29-68
Monitoring Job State with E-mail Notifications 29-70

30 Administering Oracle Scheduler

Configuring Oracle Scheduler 30-1
Setting Oracle Scheduler Privileges 30-1
Setting Scheduler Preferences 30-2
Using the Oracle Scheduler Agent to Run Remote Jobs 30-4
Monitoring and Managing the Scheduler 30-10
Viewing the Currently Active Window and Resource Plan 30-10
Finding Information About Currently Running Jobs 30-10
Monitoring and Managing Window and Job Logs 30-11
Managing Scheduler Security 30-14
Import/Export and the Scheduler 30-14
Troubleshooting the Scheduler 30-14
A Job Does Not Run 30-15
A Program Becomes Disabled 30-17
A Window Fails to Take Effect 30-17
Examples of Using the Scheduler 30-17
Examples of Creating Job Classes 30-17
Examples of Setting Attributes 30-18
Examples of Creating Chains 30-19

Examples of Creating Jobs and Schedules Based on Events 30-21
Example of Creating a Job In an Oracle Data Guard Environment 30-22
Scheduler Reference 30-23
Scheduler Privileges 30-23
Scheduler Data Dictionary Views 30-24

Part V Distributed Database Management

31 Distributed Database Concepts

Distributed Database Architecture 31-1
Homogenous Distributed Database Systems 31-1
Heterogeneous Distributed Database Systems 31-3
Client/Server Database Architecture 31-4
Database Links 31-5
What Are Database Links? 31-6
What Are Shared Database Links? 31-7
Why Use Database Links? 31-8
Global Database Names in Database Links 31-8
Names for Database Links 31-10
Types of Database Links 31-10
Users of Database Links 31-11
Creation of Database Links: Examples 31-14
Schema Objects and Database Links 31-14
Database Link Restrictions 31-16
Distributed Database Administration 31-17
Site Autonomy 31-17
Distributed Database Security 31-17
Auditing Database Links 31-22
Administration Tools 31-23
Transaction Processing in a Distributed System 31-24
Remote SQL Statements 31-24
Distributed SQL Statements 31-24
Shared SQL for Remote and Distributed Statements 31-25
Remote Transactions 31-25
Distributed Transactions 31-25
Two-Phase Commit Mechanism 31-26
Database Link Name Resolution 31-26
Schema Object Name Resolution 31-28
Global Name Resolution in Views, Synonyms, and Procedures 31-30
Distributed Database Application Development 31-32
Transparency in a Distributed Database System 31-32
Remote Procedure Calls (RPCs) 31-33
Distributed Query Optimization 31-34
Character Set Support for Distributed Environments 31-34
Client/Server Environment 31-35
Homogeneous Distributed Environment 31-35
Heterogeneous Distributed Environment 31-36

32 Managing a Distributed Database

Managing Global Names in a Distributed System 32-1
Understanding How Global Database Names Are Formed 32-1
Determining Whether Global Naming Is Enforced 32-2
Viewing a Global Database Name 32-3
Changing the Domain in a Global Database Name 32-3
Changing a Global Database Name: Scenario 32-3
Creating Database Links 32-6
Obtaining Privileges Necessary for Creating Database Links 32-6
Specifying Link Types 32-7
Specifying Link Users 32-8
Using Connection Qualifiers to Specify Service Names Within Link Names 32-10
Using Shared Database Links 32-10
Determining Whether to Use Shared Database Links 32-11
Creating Shared Database Links 32-12
Configuring Shared Database Links 32-12
Managing Database Links 32-14
Closing Database Links 32-14
Dropping Database Links 32-15
Limiting the Number of Active Database Link Connections 32-16
Viewing Information About Database Links 32-16
Determining Which Links Are in the Database 32-16
Determining Which Link Connections Are Open 32-17
Creating Location Transparency 32-18
Using Views to Create Location Transparency 32-19
Using Synonyms to Create Location Transparency 32-20
Using Procedures to Create Location Transparency 32-21
Managing Statement Transparency 32-23
Managing a Distributed Database: Examples 32-24
Example 1: Creating a Public Fixed User Database Link 32-25
Example 2: Creating a Public Fixed User Shared Database Link 32-25
Example 3: Creating a Public Connected User Database Link 32-25
Example 4: Creating a Public Connected User Shared Database Link 32-26
Example 5: Creating a Public Current User Database Link 32-26

33 Developing Applications for a Distributed Database System

Managing the Distribution of Application Data 33-1
Controlling Connections Established by Database Links 33-1
Maintaining Referential Integrity in a Distributed System 33-2
Tuning Distributed Queries 33-2
Using Collocated Inline Views 33-3
Using Cost-Based Optimization 33-3
Using Hints 33-5
Analyzing the Execution Plan 33-6
Handling Errors in Remote Procedures 33-8

34 Distributed Transactions Concepts

What Are Distributed Transactions? 34-1
DML and DDL Transactions 34-2
Transaction Control Statements 34-2
Session Trees for Distributed Transactions 34-3
Clients 34-4
Database Servers 34-4
Local Coordinators 34-4
Global Coordinator 34-4
Commit Point Site 34-5
Two-Phase Commit Mechanism 34-7
Prepare Phase 34-8
Commit Phase 34-10
Forget Phase 34-11
In-Doubt Transactions 34-11
Automatic Resolution of In-Doubt Transactions 34-12
Manual Resolution of In-Doubt Transactions 34-13
Relevance of System Change Numbers for In-Doubt Transactions 34-14
Distributed Transaction Processing: Case Study 34-14
Stage 1: Client Application Issues DML Statements 34-14
Stage 2: Oracle Database Determines Commit Point Site 34-15
Stage 3: Global Coordinator Sends Prepare Response 34-16
Stage 4: Commit Point Site Commits 34-17
Stage 5: Commit Point Site Informs Global Coordinator of Commit 34-17
Stage 6: Global and Local Coordinators Tell All Nodes to Commit 34-17
Stage 7: Global Coordinator and Commit Point Site Complete the Commit 34-18

35 Managing Distributed Transactions

Specifying the Commit Point Strength of a Node 35-1
Naming Transactions 35-2
Viewing Information About Distributed Transactions 35-2
Determining the ID Number and Status of Prepared Transactions 35-2
Tracing the Session Tree of In-Doubt Transactions 35-4
Deciding How to Handle In-Doubt Transactions 35-5
Discovering Problems with a Two-Phase Commit 35-6
Determining Whether to Perform a Manual Override 35-6
Analyzing the Transaction Data 35-7
Manually Overriding In-Doubt Transactions 35-8
Manually Committing an In-Doubt Transaction 35-8
Manually Rolling Back an In-Doubt Transaction 35-9
Purging Pending Rows from the Data Dictionary 35-9
Executing the PURGE_LOST_DB_ENTRY Procedure 35-10
Determining When to Use DBMS_TRANSACTION 35-10
Manually Committing an In-Doubt Transaction: Example 35-11
Step 1: Record User Feedback 35-11
Step 2: Query DBA_2PC_PENDING 35-11
Step 3: Query DBA_2PC_NEIGHBORS on Local Node 35-13

Step 4: Querying Data Dictionary Views on All Nodes 35-14
Step 5: Commit the In-Doubt Transaction 35-16
Step 6: Check for Mixed Outcome Using DBA_2PC_PENDING 35-16
Data Access Failures Due to Locks 35-17
Transaction Timeouts 35-17
Locks from In-Doubt Transactions 35-17
Simulating Distributed Transaction Failure 35-17
Forcing a Distributed Transaction to Fail 35-18
Disabling and Enabling RECO 35-18
Managing Read Consistency 35-19

Part VI Appendixes

A Support for DBMS_JOB in Release 11gR2

About DBMS_JOB A-1
Configuring DBMS_JOB A-1
Using Both DBMS_JOB and Oracle Scheduler A-1
Moving from DBMS_JOB to Oracle Scheduler A-2
Creating a Job A-2
Altering a Job A-2
Removing a Job from the Job Queue A-3

Index

List of Figures

1–1 Example of an Oracle Database Release Number ... 1-13
1–2 Database Administrator Authentication Methods ... 1-19
5–1 Oracle Database Dedicated Server Processes ... 5-2
5–2 Oracle Database Shared Server Processes ... 5-3
6–1 Oracle Database Memory Structures ... 6-3
9–1 Product/Component Type Subdirectories in the ADR ... 9-7
9–2 ADR Directory Structure for a Database Instance ... 9-9
9–3 Workflow for Investigating, Reporting, and Resolving a Problem ... 9-12
9–4 Alerts Section of the Database Home Page ... 9-13
9–5 Enterprise Manager Support Workbench Home Page ... 9-18
9–6 Select Package Page ... 9-36
9–7 Customize Package Page ... 9-37
9–8 Copy Out Files Page ... 9-41
9–9 Files Subpage of Customize Package Page ... 9-43
10–1 Directory Structure in the Management Service Home for Release 11g or Later ... 10-3
10–2 Directory Structure in the Management Home for Release 10g ... 10-3
12–1 Reuse of Redo Log Files by LGWR ... 12-3
12–2 Multiplexed Redo Log Files ... 12-4
12–3 Legal and Illegal Multiplexed Redo Log Configuration ... 12-6
13–1 Redo Log File Use in ARCHIVELOG Mode ... 13-3
15–1 Components of File Mapping ... 15-16
15–2 Illustration of Mapping Structures ... 15-19
19–1 Tables page ... 19-15
19–2 Advisor Central page ... 19-16
19–3 Segment Advisor Recommendations page ... 19-20
19–4 Recommendation Details page ... 19-21
21–1 Coalescing Indexes ... 21-8
22–1 Clustered Table Data ... 22-2
27–1 A Simple Resource Plan ... 27-5
27–2 A Resource Plan With Subplans ... 27-6
27–3 Multilevel Plan Schema ... 27-39
27–4 Resource Plan with Maximum Utilization for Subplan and Consumer Groups ... 27-43
28–1 Chain with Multiple Branches ... 28-9
28–2 Windows help define the resources that are allocated to jobs ... 28-11
28–3 Windows and Resource Plans (Example 1) ... 28-12
28–4 Windows and Resource Plans (Example 2) ... 28-13
28–5 Relationships Among Programs, Jobs, and Schedules ... 28-22
28–6 Scheduler Components ... 28-23
28–7 Oracle RAC Architecture and the Scheduler ... 28-26
28–8 Service Affinity and the Scheduler ... 28-26
29–1 Chain with Step 3 Paused ... 29-51
29–2 Sample Resource Plan ... 29-64
31–1 Homogeneous Distributed Database ... 31-2
31–2 An Oracle Database Distributed Database System ... 31-5
31–3 Database Link ... 31-6
31–4 Hierarchical Arrangement of Networked Databases ... 31-9
31–5 Global User Security ... 31-21
31–6 NLS Parameter Settings in a Client/Server Environment ... 31-35
31–7 NLS Parameter Settings in a Homogeneous Environment ... 31-36
31–8 NLS Parameter Settings in a Heterogeneous Environment ... 31-36
32–1 A Shared Database Link to Dedicated Server Processes ... 32-13
32–2 Shared Database Link to Shared Server ... 32-14
32–3 Views and Location Transparency ... 32-19
34–1 Distributed System ... 34-1

34–2 Example of a Session Tree 34-3
34–3 Commit Point Site 34-5
34–4 Commit Point Strengths and Determination of the Commit Point Site 34-6
34–5 Failure During Prepare Phase 34-12
34–6 Failure During Commit Phase 34-13
34–7 Defining the Session Tree 34-15
34–8 Determining the Commit Point Site 34-16
34–9 Sending and Acknowledging the Prepare Message 34-17
34–10 Instructing Nodes to Commit 34-18
35–1 Example of an In-Doubt Distributed Transaction 35-11

List of Tables

1–1 Required Password File Name and Location on UNIX, Linux, and Windows 1-24
2–1 Database Planning Tasks 2-2
2–2 Recommended Minimum Initialization Parameters 2-8
2–3 PFILE and SPFILE Default Names and Locations on UNIX, LInux, and Windows..... 2-34
4–1 Oracle Components Automatically Restarted by Oracle Restart..... 4-2
4–2 Create Operations and the Oracle Restart Configuration..... 4-4
4–3 Delete/Drop/Remove Operations and the Oracle Restart Configuration 4-5
4–4 Event Record Parameters and Descriptions..... 4-8
4–5 FAN Parameters and Matching Database Signatures 4-8
4–6 Determining the Oracle Home from which to Start SRVCTL 4-10
4–7 Summary of SRVCTL Commands..... 4-30
4–8 Component Keywords and Abbreviations 4-31
4–9 srvctl add Summary 4-32
4–10 srvctl add asm Options 4-32
4–11 srvctl add database Options 4-33
4–12 srvctl add listener Options..... 4-35
4–13 srvctl add ons Options 4-35
4–14 srvctl add service Options 4-36
4–15 srvctl config Summary 4-38
4–16 srvctl config asm Options 4-38
4–17 srvctl config database Options 4-39
4–18 srvctl config listener Options 4-39
4–19 srvctl config service Options 4-40
4–20 srvctl disable Summary..... 4-41
4–21 srvctl disable database Options 4-41
4–22 srvctl disable diskgroup Options..... 4-42
4–23 srvctl disable listener Options..... 4-42
4–24 srvctl disable ons Options..... 4-42
4–25 srvctl disable service Options..... 4-43
4–26 srvctl enable Summary..... 4-44
4–27 srvctl enable database Options 4-45
4–28 srvctl enable diskgroup Options..... 4-45
4–29 srvctl enable listener Options..... 4-45
4–30 srvctl enable ons Options..... 4-46
4–31 srvctl enable service Options..... 4-46
4–32 srvctl getenv Summary 4-47
4–33 srvctl getenv asm Options 4-47
4–34 srvctl getenv database Options 4-48
4–35 srvctl getenv listener Options 4-48
4–36 srvctl modify Summary 4-49
4–37 srvctl modify asm Options 4-49
4–38 srvctl modify database Options 4-50
4–39 srvctl modify listener Options..... 4-50
4–40 srvctl modify ons Options 4-51
4–41 srvctl modify service Options 4-51
4–42 srvctl remove Summary..... 4-53
4–43 srvctl remove asm Options..... 4-53
4–44 srvctl remove database Options..... 4-54
4–45 srvctl remove diskgroup Options..... 4-54
4–46 srvctl remove listener Options..... 4-54
4–47 srvctl remove ons Options..... 4-55
4–48 srvctl remove service Options..... 4-55
4–49 srvctl setenv and unsetenv Summary 4-56

4–50 srvctl setenv database Options 4-56
4–51 srvctl setenv database Options 4-57
4–52 srvctl setenv listener Options 4-57
4–53 srvctl start Summary 4-58
4–54 srvctl start asm Option 4-58
4–55 srvctl start database Options 4-59
4–56 srvctl start diskgroup Options 4-59
4–57 srvctl start home Options 4-60
4–58 srvctl start listener Options 4-60
4–59 srvctl start ons Options 4-60
4–60 srvctl start service Options 4-61
4–61 srvctl status Summary 4-62
4–62 srvctl status asm Options 4-62
4–63 srvctl status database Options 4-63
4–64 srvctl status diskgroup Options 4-63
4–65 srvctl status home Options 4-63
4–66 srvctl status listener Options 4-64
4–67 srvctl status service Options 4-64
4–68 srvctl stop Summary 4-65
4–69 srvctl stop asm Option 4-65
4–70 srvctl stop database Options 4-66
4–71 srvctl stop diskgroup Options 4-66
4–72 srvctl stop home Options 4-67
4–73 srvctl stop listener Options 4-67
4–74 srvctl stop ons Options 4-68
4–75 srvctl stop service Options 4-68
4–76 srvctl unsetenv Command Summary 4-69
4–77 srvctl unsetenv asm Options 4-69
4–78 srvctl unsetenv database Options 4-70
4–79 srvctl unsetenv listener Options 4-70
4–80 Summary of CRSCTL Commands 4-71
4–81 crsctl stop has Options 4-77
5–1 Dedicated Servers, Shared Servers, and Database Resident Connection Pooling 5-5
5–2 Configuration Parameters for Database Resident Connection Pooling 5-17
5–3 Data Dictionary Views for Database Resident Connection Pooling 5-19
5–4 Oracle Database Background Processes 5-20
6–1 Automatically Sized SGA Components and Corresponding Parameters 6-9
6–2 Manually Sized SGA Components that Use SGA_TARGET Space 6-9
6–3 Flash Cache Initialization Parameters 6-22
9–1 ADR Home Path Components for Oracle Database 9-8
9–2 ADR Home Subdirectories 9-8
9–3 Data in the V$DIAG_INFO View 9-10
9–4 Data in the V$DIAG_CRITICAL_ERROR View 9-10
9–5 Oracle Advisors that Help Repair Critical Errors 9-17
9–6 Parameters for Data Block Integrity Check 9-28
9–7 Parameters for Redo Integrity Check 9-28
9–8 Parameters for Undo Segment Integrity Check 9-28
9–9 Parameters for Transaction Integrity Check 9-28
9–10 Parameters for Dictionary Integrity Check 9-28
10–1 EMCA Command-Line Operations 10-6
10–2 EMCA Command-Line Flags 10-8
10–3 EMCA Command-Line Parameters 10-9
13–1 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 1 13-12
13–2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2 13-12
14–1 Minimum Compatibility Requirements 14-35

18–1 DBA Tasks for Edition-Based Redefinition 18-22
19–1 DBMS_ADVISOR package procedures relevant to the Segment Advisor 19-16
19–2 Input for DBMS_ADVISOR.CREATE_OBJECT 19-17
19–3 Input for DBMS_ADVISOR.SET_TASK_PARAMETER 19-17
19–4 Segment Advisor Result Types 19-19
19–5 Segment Advisor Outcomes: Summary 19-23
20–1 Table Compression Methods 20-5
20–2 Mandatory Error Description Columns 20-24
20–3 Error Logging Table Column Data Types 20-24
20–4 ALTER TABLE Clauses for External Tables 20-69
21–1 Costs and Benefits of Coalescing or Rebuilding Indexes 21-7
25–1 Comparison of Corruption Detection Methods 25-3
26–1 Predefined Maintenance Windows 26-7
26–2 Automated Maintenance Tasks Database Dictionary Views 26-7
27–1 A Simple Three-Level Resource Plan 27-19
27–2 A Three-Level Resource Plan with Maximum Utilization Limits 27-20
27–3 Resource Plan with Parallel Statement Directives 27-46
27–4 Predefined Resource Plans 27-59
27–5 Predefined Resource Consumer Groups 27-59
27–6 Predefined Consumer Group Mapping Rules 27-60
27–7 Resource Manager Data Dictionary Views 27-61
28–1 Default Credentials for Local External Jobs 28-17
29–1 Job Tasks and Their Procedures 29-2
29–2 Credential and Destination Tasks and Their Procedures 29-6
29–3 Program Tasks and Their Procedures 29-21
29–4 Schedule Tasks and Their Procedures 29-25
29–5 Event Tasks and Their Procedures for Events Raised by an Application 29-32
29–6 Chain Tasks and Their Procedures 29-42
29–7 Values for the State Attribute of a Chain Step 29-45
29–8 Job Class Tasks and Their Procedures 29-54
29–9 Window Tasks and Their Procedures 29-56
29–10 Window Group Tasks and Their Procedures 29-61
29–11 Job Logging Levels 29-65
29–12 Scheduler Data Dictionary View Contents for Multiple-Destination Jobs 29-67
29–13 Job State Event Types Raised by the Scheduler 29-68
30–1 schagent options 30-8
30–2 Job States 30-11
30–3 Scheduler System Privileges 30-23
30–4 Scheduler Object Privileges 30-24
30–5 Scheduler Views 30-25
35–1 DBA_2PC_PENDING 35-3
35–2 DBA_2PC_NEIGHBORS 35-4

Preface

This document describes how to create, configure, and administer an Oracle database.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This document is intended for database administrators who perform the following tasks:

- Create and configure one or more Oracle databases
- Monitor and tune Oracle databases
- Oversee routine maintenance operations for Oracle databases
- Create and maintain schema objects, such as tables, indexes, and views
- Schedule system and user jobs
- Diagnose, repair, and report problems

To use this document, you should be familiar with relational database concepts. You should also be familiar with the operating system environment under which you are running Oracle Database.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

- *Oracle Database 2 Day DBA*
- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*
- *Oracle Database Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Automatic Storage Management Administrator's Guide*
- *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database Error Messages*
- *Oracle Database Net Services Administrator's Guide*
- *Oracle Database Backup and Recovery User's Guide*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database Advanced Application Developer's Guide*
- *Oracle Database PL/SQL Language Reference*
- *SQL*Plus User's Guide and Reference*

Many of the examples in this book use the sample schemas, which are installed by default when you select the Basic Installation option with an Oracle Database installation. See *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Database Administrator's Guide?

This section describes new features of Oracle Database 11*g* Release 2 that are documented in this guide, and provides pointers to additional information.

This section contains the following topics:

- Oracle Database 11g Release 2 (11.2.0.2) New Features in the Administrator's Guide
- Oracle Database 11g Release 2 (11.2.0.1) New Features in the Administrator's Guide

Oracle Database 11*g* Release 2 (11.2.0.2) New Features in the Administrator's Guide

- Deferred segment creation for partitioned objects
  - Deferred segment creation now applies to partitioned objects as well as nonpartitioned objects.

    See "Understand Deferred Segment Creation" on page 20-9.
  - You can drop segments from empty tables that were migrated from previous releases, using the `DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS` procedure.

    See "Dropping Unused Object Storage" on page 19-28.
  - The `DROP ALL STORAGE` clause of the `TRUNCATE` statement enables you to deallocate segments for a table.

    See "Using TRUNCATE" on page 18-6.
  - You can materialize segments for tables, partitions, and dependent objects for which segment creation was deferred, using the `DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS` procedure.

    See "Materializing Segments" on page 20-11.
- New default first extent size for partitioned tables improves performance

  The default size of the first extent of any new segment for a partitioned table is now 8 MB instead of 64 KB. This helps improve performance of inserts and queries on partitioned tables. Although partitioned tables will start with a larger initial size, once sufficient data is inserted, the space consumption will be the same as in previous releases. You can override this default by setting the `INITIAL` size in the

storage clause for the table. This new default only applies to table partitions and LOB partitions.

- New initialization parameter to improve the performance of `NOLOGGING` direct-path inserts

  You can now significantly improve the performance of unrecoverable direct path inserts by disabling the periodic update of the control files. You do so by setting the new initialization parameter `DB_UNRECOVERABLE_SCN_TRACKING` to `FALSE`. However, if you perform an unrecoverable direct path insert with these control file updates disabled, you will no longer be able to accurately query the database to determine if any datafiles are currently unrecoverable.

  See "Direct-Path INSERT without Logging" on page 20-20

- Improved e-mail notification capabilities in Oracle Scheduler

  Oracle Scheduler e-mail notification preferences now include options to specify authentication and SSL or TLS protocols for the designated SMTP Server.

  See "Setting Scheduler Preferences" on page 30-2.

- Edition attributes of database services

  You can set the edition attribute of a database service when you create the service, or you can modify an existing database service to set its edition attribute.

  See "Setting the Edition Attribute of a Database Service" on page 18-23.

- Enhancements to Oracle Database Resource Manager

  – You can optimize the performance of parallel statements by limiting the parallel statement activity per consumer group, prioritizing parallel statements in the parallel statement queue, and limiting the wait time for parallel statement in the parallel statement queue.

    Parallel statement queuing enables efficient management of parallel statements when the database does not have sufficient resources to run a statement at the specified degree of parallelism. A submitted parallel statement is added to the parallel statement queue if the resources required to run the statement exceed the parallel statement activity limit specified for the database or the consumer group to which the parallel statement is assigned. Parallel statement queuing can also manage multiple parallel statement workloads according to the priorities and resource allocations specified by a resource plan.

    See Also:

    - "Parallel Target Percentage" on page 27-21
    - "Parallel Queue Timeout" on page 27-22
    - "Example of Managing Parallel Statements Using Directive Attributes" on page 27-45

  – You can track CPU utilization using per-minute metrics for each resource plan. You can also monitor resource utilization for consumer groups, even if Resource Manager is not enabled, thus allowing you to view the potential effect of Resource Manager.

    See "V$RSRCMGRMETRIC" on page 27-56.

Oracle Database 11*g* Release 2 (11.2.0.1) New Features in the Administrator's Guide

- Oracle Restart improves database availability by automatically restarting the database after a failure.

  If you configure Oracle Restart, then the database, the listener, the Oracle Automatic Storage Management instance, and other Oracle components can be automatically restarted after a hardware or software failure or after a restart of the database host computer.

  See Chapter 4, "Configuring Automatic Restart of an Oracle Database".

- Edition-based redefinition enables application developers and DBAs to upgrade an application with little or no application down time.

  A new database construct called an edition provides a privacy mechanism for installing new code and for making data changes so that the running production application does not see the changes. When all the required changes have been made in private, they can be made available to users. In support of edition-based redefinition, a new kind of view called an editioning view and a new kind of trigger called a crossedition trigger are introduced.

  See "Managing Editions" on page 18-21.

- Database Smart Flash Cache

  Database Smart Flash Cache is an optional memory component that you can add if your database is running on Solaris or Oracle Linux. It is an extension of the SGA-resident buffer cache, providing a level 2 cache for database blocks. It can improve response time and overall throughput.

  See "Memory Architecture Overview" on page 6-2.

- The Automatic Segment Advisor can now return a recommendation to use OLTP compression for a table.

  See "Reclaiming Wasted Space" on page 19-12.

- Deferred segment creation

  When creating a nonpartitioned heap-organized table in a locally managed tablespace, table segment creation is deferred until the first row is inserted.

  See "Understand Deferred Segment Creation" on page 20-9.

- Oracle Database File System

  The Oracle Database File System creates a standard file system interface on top of files and directories that are stored in database tables.

  See *Oracle Database SecureFiles and Large Objects Developer's Guide* for complete information.

- Oracle Scheduler enhancements

  – Remote database jobs—You can now create a job that runs stored procedures and anonymous PL/SQL blocks on another database instance on the same host or on a remote host. The target database can be any release of Oracle Database.

    See "Database Jobs" on page 28-15.

  – Multiple-destination jobs—You can now run a job on multiple locations, and control and monitor all instances of the job from one central database. You do

so by specifying multiple destinations when you create the job. A destination can be the local host or local database; a remote host (for remote external jobs); or a remote database (for remote database jobs).

See "Multiple-Destination Jobs" on page 28-18

- File watchers—A new Scheduler object called a file watcher simplifies the task of configuring the Scheduler to start a job upon the arrival of a file on the local or a remote system.

  See "Starting a Job When a File Arrives on a System" on page 29-35.

- E-mail notifications—You can configure the Scheduler to automatically send an e-mail notification to one or more recipients when a specified job state event occurs. You can now receive an e-mail when a job completes, if it fails or is disabled, if it exceeds its allotted run time, and so on.

  See "Monitoring Job State with E-mail Notifications" on page 29-70.

- Database Resource Manager Enhancements
  - Instance caging

    Oracle Database now provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. Instance caging limits the maximum number of CPUs that any one database instance can use. If an instance then becomes CPU-bound, the Resource Manager begins allocating CPU based on the current resource plan. Thus, instance caging and the Resource Manager work together to support desired levels of service across multiple instances.

    See "Managing Multiple Database Instances on a Single Server" on page 27-48.

  - New `MAX_UTILIZATION_LIMIT` attribute of resource plan directives enables you to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any automatic redistribution of CPU within a plan.
  - New `ORACLE_FUNCTION` consumer group mapping rule type, and new predefined mapping rules for Data Pump and RMAN.

    Sessions performing a data load with Data Pump or performing backup or copy operations with RMAN are now automatically mapped to predefined consumer groups.

    See "Predefined Consumer Group Mapping Rules" on page 27-60.

  - New sample resource plans and resource consumer groups to support data warehousing operations with Oracle Exadata

    See "Predefined Resource Plans and Consumer Groups" on page 27-58.

- New SQL command syntax for specifying table compression for direct load operations only or for all (OLTP) operations.

  See "Consider Using Table Compression" on page 20-5.

- The Flash Recovery Area is renamed to Fast Recovery Area.
- External tables can be preprocessed by user-supplied preprocessor programs.

  By using a preprocessing program, users can use data from a file that is not in a format supported by the access driver. For example, a user may want to access data stored in a compressed format. Specifying a decompression program for the

ORACLE_LOADER access driver allows the data to be decompressed as the access driver processes the data.

See "Preprocessing External Tables" on page 20-69.

- Archive logging now supports up to 30 standby databases.
- IP version 6 is now supported.

  Oracle Database components and utilities now support Internet Protocol version 6 (IPv6) addresses, which are 128 bits in length. You can now specify an IPv6 address with the easy connect method in SQL*Plus.

  See "Connecting to the Database with SQL*Plus" on page 1-7.

- Redo logs can now be stored on disk drives with a sector size of 4K bytes without performance degradation.

  A new redo log file block size of 4K bytes enables online redo logs to be stored on newer high-capacity disks with a 4K byte sectors size without incurring performance degradation. The new block size ensures that log file writes are sector-aligned.

  See "Planning the Block Size of Redo Log Files" on page 12-7.

- The Enterprise Manager Support Workbench, a component of the fault diagnosability infrastructure, now supports investigating, reporting, and resolving critical errors in Oracle Automatic Storage Management instances.

  See Chapter 9, "Managing Diagnostic Data".

Part I

Basic Database Administration

Part I provides an overview of the responsibilities of a database administrator, and describes how to accomplish basic database administration tasks. It contains the following chapters:

- Chapter 1, "Getting Started with Database Administration"
- Chapter 2, "Creating and Configuring an Oracle Database"
- Chapter 3, "Starting Up and Shutting Down"
- Chapter 4, "Configuring Automatic Restart of an Oracle Database"
- Chapter 5, "Managing Processes"
- Chapter 6, "Managing Memory"
- Chapter 7, "Managing Users and Securing the Database"
- Chapter 8, "Monitoring Database Operations"
- Chapter 9, "Managing Diagnostic Data"
- Chapter 10, "Managing Oracle Enterprise Manager Database Control"

1

Getting Started with Database Administration

In this chapter:

- Types of Oracle Database Users
- Tasks of a Database Administrator
- Submitting Commands and SQL to the Database
- Identifying Your Oracle Database Software Release
- About Database Administrator Security and Privileges
- Database Administrator Authentication
- Creating and Maintaining a Password File
- Data Utilities

Types of Oracle Database Users

The types of users and their roles and responsibilities depend on the database site. 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.

Database Administrators

Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.

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

- Installing and upgrading the Oracle Database 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 Oracle license agreements
- 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 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. See *Oracle Database Security Guide* for information about the duties of security officers.

Network Administrators

Some sites have one or more network administrators. A network administrator, for example, administers Oracle networking products, such as Oracle Net Services. See *Oracle Database Net Services Administrator's Guide* for information about the duties of network administrators.

> **See Also:** Part V, "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 this information to a database administrator
- Tuning the application during development
- Establishing security measures for an application during development

Application developers can perform some of these tasks in collaboration with DBAs. See *Oracle Database Advanced Application Developer's Guide* for information about application development tasks.

Application Administrators

An Oracle Database site can assign one or more application administrators to administer 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 Database 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

Task 10: Download and Install Patches

Task 11: Roll Out to Additional Hosts

These tasks are discussed in the sections that follow.

Note: When upgrading to a new release, back up your existing production environment, both software and database, before installation. For information on preserving your existing production database, see *Oracle Database Upgrade Guide*.

Task 1: Evaluate the Database Server Hardware

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

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

Task 2: Install the Oracle Database 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 systems to the computer that executes Oracle Database.

For more information on what software to install, see "Identifying Your Oracle Database Software Release" on page 1-12.

> **See Also:** For specific requirements and instructions for installation, see the following documentation:
>
> - The Oracle documentation specific to your operating system
> - The 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 comprise 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, consider the effects that this structure will have when the database is actually created and running. Consider how the logical storage structure of the database will affect:

- The performance of the computer running Oracle Database
- 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, see accepted industry-standard documentation.

Part II, "Oracle Database Structure and Storage", 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

After 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.

See Chapter 2, "Creating and Configuring an Oracle Database", for information on creating a database and Chapter 3, "Starting Up and Shutting Down" for guidance in starting up the database.

Task 5: Back Up the Database

After you create the database structure, perform 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:** *Oracle Database Backup and Recovery 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, and grant appropriate privileges and roles to these users. See Chapter 7, "Managing Users and Securing the Database" for guidance in this task.

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 have finished creating tablespaces, you can create the database objects.

Part II, "Oracle Database Structure and Storage" and Part III, "Schema Objects" provide information on creating logical storage structures and objects for your database.

Task 8: Back Up the Fully Functional Database

When 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. Oracle Database provides a database resource management feature that helps you to control the allocation of resources among various user groups. The database resource manager is described in Chapter 27, "Managing Resources with Oracle Database Resource Manager".

> **See Also:** *Oracle Database Performance Tuning Guide* for information about tuning your database and applications

Task 10: Download and Install Patches

After installation and on a regular basis, download and install patches. Patches are available as single interim patches and as patchsets (or **patch releases**). Interim patches address individual software bugs and may or may not be needed at your installation. Patch releases are collections of bug fixes that are applicable for all customers. Patch releases have release numbers. For example, if you installed Oracle Database 11.2.0.1, the first patch release will have a release number of 11.2.0.2.

> **See Also:** *Oracle Database Installation Guide* for your platform for instructions on downloading and installing patches.

Task 11: Roll Out to Additional Hosts

After you have an Oracle Database installation properly configured, tuned, patched, and tested, you may want to roll that exact installation out to other hosts. Reasons to do this include the following:

- You have multiple production database systems.
- You want to create development and test systems that are identical to your production system.

Instead of installing, tuning, and patching on each additional host, you can **clone** your tested Oracle Database installation to other hosts, saving time and avoiding inconsistencies. There are two types of cloning available to you:

- Cloning an Oracle home—Just the configured and patched binaries from the Oracle home directory and subdirectories are copied to the destination host and "fixed" to match the new environment. You can then start an instance with this cloned home and create a database.

  You can use the Enterprise Manager Clone Oracle Home tool to clone an Oracle home to one or more destination hosts. You can also manually clone an Oracle home using a set of provided scripts and Oracle Universal Installer.
- Cloning a database—The tuned database, including database files, initialization parameters, and so on, are cloned to an existing Oracle home (possibly a cloned home).

  You can use the Enterprise Manager Clone Database tool to clone an Oracle database instance to an existing Oracle home.

  > **See Also:**
  >
  > - *Oracle Universal Installer and OPatch User's Guide for Windows and UNIX* for information about cloning Oracle software.
  > - Enterprise Manager online help for instructions for cloning a database.

Submitting Commands and SQL to the Database

The primary means of communicating with Oracle Database is by submitting SQL statements. Oracle Database also supports a superset of SQL, which includes commands for starting up and shutting down the database, modifying database configuration, and so on. There are three ways to submit these SQL statements and commands to Oracle Database:

- Directly, using the command-line interface of SQL*Plus
- Indirectly, using the graphical user interface of Oracle Enterprise Manager

With Oracle Enterprise Manager (Enterprise Manager), you use an intuitive graphical interface to administer the database, and Enterprise Manager submits SQL statements and commands behind the scenes.

See *Oracle Database 2 Day DBA* for more information.

- Directly, using SQL Developer

  Developers use SQL Developer to create and test database schemas and applications, although you can also use it for database administration tasks.

  See *Oracle Database 2 Day Developer's Guide* for more information.

This section focuses on using SQL*Plus to submit SQL statements and commands to the database. It includes the following topics:

- About SQL*Plus
- Connecting to the Database with SQL*Plus

About SQL*Plus

SQL*Plus is the primary command-line interface to your Oracle database. You use SQL*Plus to start up and shut down the database, set database initialization parameters, create and manage users, create and alter database objects (such as tables and indexes), insert and update data, run SQL queries, and more.

Before you can submit SQL statements and commands, you must connect to the database. With SQL*Plus, you can connect locally or remotely. **Connecting locally** means connecting to an Oracle database running on the same computer on which you are running SQL*Plus. **Connecting remotely** means connecting over a network to an Oracle database that is running on a remote computer. Such a database is referred to as a **remote database**. The SQL*Plus executable on the local computer is provided by a full Oracle Database installation, an Oracle Client installation, or an Instant Client installation.

See Also: *SQL*Plus User's Guide and Reference*

Connecting to the Database with SQL*Plus

Oracle Database includes the following components:

- The Oracle Database instance, which is a collection of processes and memory
- A set of disk files that contain user data and system data

When you connect with SQL*Plus, you are connecting to the Oracle instance. Each instance has an instance ID, also known as a system ID (SID). Because there can be multiple Oracle instances on a host computer, each with its own set of datafiles, you must identify the instance to which you want to connect. For a local connection, you identify the instance by setting operating system environment variables. For a remote connection, you identify the instance by specifying a network address and a database service name. For both local and remote connections, you must set environment variables to help the operating system find the SQL*Plus executable and to provide the executable with a path to its support files and scripts. To connect to an Oracle instance with SQL*Plus, therefore, you must complete the following steps:

Step 1: Open a Command Window
Step 2: Set Operating System Environment Variables
Step 3: Start SQL*Plus
Step 4: Submit the SQL*Plus CONNECT Statement

See Also: *Oracle Database Concepts* for background information about the Oracle instance

Step 1: Open a Command Window

Take the necessary action on your platform to open a window into which you can enter operating system commands.

Step 2: Set Operating System Environment Variables

Depending on your platform, you may have to set environment variables before starting SQL*Plus, or at least verify that they are set properly.

For example, on most platforms, `ORACLE_SID` and `ORACLE_HOME` must be set. In addition, it is advisable to set the `PATH` environment variable to include the *ORACLE_HOME*/bin directory. Some platforms may require additional environment variables:

- On the UNIX and Linux platforms, you must set environment variables by entering operating system commands.
- On the Windows platform, Oracle Universal Installer (OUI) automatically assigns values to `ORACLE_HOME` and `ORACLE_SID` in the Windows registry.

If you did not create a database upon installation, OUI does not set `ORACLE_SID` in the registry; after you create your database at a later time, you must set the `ORACLE_SID` environment variable from a command window.

UNIX and Linux installations come with two scripts, `oraenv` and `coraenv`, that you can use to easily set environment variables. For more information, see *Administrator's Reference for UNIX Systems*.

For all platforms, when switching between instances with different Oracle homes, you must change the `ORACLE_HOME` environment variable. If multiple instances share the same Oracle home, you must change only `ORACLE_SID` when switching instances.

See the *Oracle Database Installation Guide* or administration guide for your operating system for details on environment variables and for information on switching instances.

Example 1–1 Setting Environment Variables in UNIX (C Shell)

```
setenv ORACLE_SID orcl
setenv ORACLE_HOME /u01/app/oracle/product/11.2.0/db_1
setenv LD_LIBRARY_PATH $ORACLE_HOME/lib:/usr/lib:/usr/dt/lib:/usr/openwin/lib:/usr/ccs/lib
```

Example 1–2 Setting Environment Variables in Windows

```
SET ORACLE_SID=orawin2
```

Example 1–2 assumes that `ORACLE_HOME` and `ORACLE_SID` are set in the registry but that you want to override the registry value of `ORACLE_SID` to connect to a different instance.

On Windows, environment variable values that you set in a command prompt window override the values in the registry.

Step 3: Start SQL*Plus

To start SQL*Plus:

1. Do one of the following:
   - Ensure that the `PATH` environment variable contains *ORACLE_HOME*/bin.

- Change directory to *ORACLE_HOME*/bin.

2. Enter the following command (case-sensitive on UNIX and Linux):

```
sqlplus /nolog
```

Step 4: Submit the SQL*Plus CONNECT Statement

You submit the SQL*Plus `CONNECT` statement to initially connect to the Oracle instance or at any time to reconnect as a different user. The syntax of the `CONNECT` statement is as follows:

```
CONN[ECT] [logon] [AS {SYSOPER | SYSDBA}]
```

The syntax of *`logon`* is as follows:

```
{username | /}[@connect_identifier] [edition={edition_name | DATABASE_DEFAULT}]
```

When you provide *`username`*, SQL*Plus prompts for a password. The password is not echoed as you type it.

The following table describes the syntax components of the `CONNECT` statement.

Syntax Component	Description
`/`	Calls for external authentication of the connection request. A database password is not used in this type of authentication. The most common form of external authentication is operating system authentication, where the database user is authenticated by having logged in to the host operating system with a certain host user account. External authentication can also be performed with an Oracle wallet or by a network service. See *Oracle Database Security Guide* for more information. See also "Using Operating System Authentication" on page 1-20.
`AS {SYSOPER | SYSDBA}`	Indicates that the database user is connecting with either the `SYSOPER` or `SYSDBA` system privilege. Only certain predefined administrative users or users who have been added to the password file may connect with these privileges. See "Administrative Privileges" on page 1-16 for more information.
`username`	A valid database user name. The database authenticates the connection request by matching *`username`* against the data dictionary and prompting for a user password.
`connect_identifier` (1)	An Oracle Net connect identifier, for a remote connection. The exact syntax depends on the Oracle Net configuration. If omitted, SQL*Plus attempts connection to a local instance. A common connect identifier is a *net service name*. This is an alias for an Oracle Net connect descriptor (network address and database service name). The alias is typically resolved in the `tnsnames.ora` file on the local computer, but can be resolved in other ways. See *Oracle Database Net Services Administrator's Guide* for more information on connect identifiers.

Syntax Component	Description
connect_identifier (2)	As an alternative, a connect identifier can use *easy connect* syntax. Easy connect provides out-of-the-box TCP/IP connectivity for remote databases without having to configure Oracle Net Services on the client (local) computer. Easy connect syntax for the connect identifier is as follows (the enclosing double-quotes must be included): `"host[:port][/service_name][:server][/instance_name]"` where: ■ *host* is the host name or IP address of the computer hosting the remote database. Both IP version 4 (IPv4) and IP version 6 (IPv6) addresses are supported. IPv6 addresses must be enclosed in square brackets. See *Oracle Database Net Services Administrator's Guide* for information about IPv6 addressing. ■ *port* is the TCP port on which the Oracle Net listener on *host* listens for database connections. If omitted, 1521 is assumed. ■ *service_name* is the database service name to which to connect. Can be omitted if the Net Services listener configuration on the remote host designates a default service. If no default service is configured, *service_name* must be supplied. Each database typically offers a standard service with a name equal to the global database name, which is made up of the DB_NAME and DB_DOMAIN initialization parameters as follows: DB_NAME.DB_DOMAIN If DB_DOMAIN is null, then the standard service name is just the DB_NAME. For example, if DB_NAME is orcl and DB_DOMAIN is us.example.com, then the standard service name is orcl.us.example.com. See "Managing Application Workloads with Database Services" on page 2-39 for more information. ■ *server* is the type of service handler. Acceptable values are dedicated, shared, and pooled. If omitted, the default type of server is chosen by the listener: shared server if configured, otherwise dedicated server. ■ *instance_name* is the instance to which to connect. You can specify both service name and instance name, which you would typically do only for Oracle Real Application Clusters (Oracle RAC) environments. For Oracle RAC or single instance environments, if you specify only instance name, you connect to the default database service. If there is no default service configured in the listener.ora file, an error is generated.You can obtain the instance name from the instance_name initialization parameter. See *Oracle Database Net Services Administrator's Guide* for more information on easy connect.
edition={*edition_name* | DATABASE_DEFAULT}	Specifies the edition in which the new database session starts. If you specify an edition, it must exist and you must have the USE privilege on it. If this clause is not specified, the database default edition is used for the session. See *Oracle Database Advanced Application Developer's Guide* for information on editions and edition-based redefinition.

Example 1–3

This simple example connects to a local database as user `SYSTEM`. SQL*Plus prompts for the `SYSTEM` user password.

```
connect system
```

Example 1–4

This example connects to a local database as user `SYS` with the `SYSDBA` privilege. SQL*Plus prompts for the `SYS` user password.

```
connect sys as sysdba
```

When connecting as user `SYS`, you must connect `AS SYSDBA`.

Example 1–5

This example connects locally with operating system authentication.

```
connect /
```

Example 1–6

This example connects locally with the `SYSDBA` privilege with operating system authentication.

```
connect / as sysdba
```

Example 1–7

This example uses easy connect syntax to connect as user `salesadmin` to a remote database running on the host `dbhost.example.com`. The Oracle Net listener (the listener) is listening on the default port (1521). The database service is `sales.example.com`. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@"dbhost.example.com/sales.example.com"
```

Example 1–8

This example is identical to Example 1–7, except that the service handler type is indicated.

```
connect salesadmin@"dbhost.example.com/sales.example.com:dedicated"
```

Example 1–9

This example is identical to Example 1–7, except that the listener is listening on the nondefault port number 1522.

```
connect salesadmin@"dbhost.example.com:1522/sales.example.com"
```

Example 1–10

This example is identical to Example 1–7, except that the host IP address is substituted for the host name.

```
connect salesadmin@"192.0.2.5/sales.example.com"
```

Example 1–11

This example connects using an IPv6 address. Note the enclosing square brackets.

```
connect salesadmin@"[2001:0DB8:0:0::200C:417A]/sales.example.com"
```

Example 1–12

This example specifies the instance to which to connect and omits the database service name. A default database service must have been specified, otherwise an error is generated. Note that when you specify the instance only, you cannot specify the service handler type.

```
connect salesadmin@"dbhost.example.com//orcl"
```

Example 1–13

This example connects remotely as user `salesadmin` to the database service designated by the net service name `sales1`. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@sales1
```

Example 1–14

This example connects remotely with external authentication to the database service designated by the net service name `sales1`.

```
connect /@sales1
```

Example 1–15

This example connects remotely with the `SYSDBA` privilege and with external authentication to the database service designated by the net service name `sales1`.

```
connect /@sales1 as sysdba
```

Example 1–16

This example connects remotely as user `salesadmin` to the database service designated by the net service name `sales1`. The database session starts in the `rev21` edition. SQL*Plus prompts for the `salesadmin` user password.

```
connect salesadmin@sales1 edition=rev21
```

See Also:

- "Using Operating System Authentication" on page 1-20
- "Managing Application Workloads with Database Services" on page 2-39 for information about database services
- *SQL*Plus User's Guide and Reference* for more information on the `CONNECT` statement
- *Oracle Database Net Services Administrator's Guide* for more information on net service names
- *Oracle Database Net Services Reference* for information on how to define the default service in `listener.ora`

Identifying Your Oracle Database Software Release

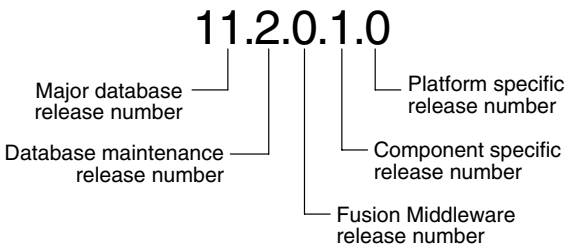
Because Oracle Database continues to evolve and can require maintenance, Oracle periodically produces new releases. Not all customers initially subscribe to a new release or require specific maintenance for their existing release. As a result, multiple releases of the product exist simultaneously.

As many as five numbers may be required to fully identify a release. The significance of these numbers is discussed in the sections that follow.

Release Number Format

To understand the release nomenclature used by Oracle, examine the following example of an Oracle Database release labeled "11.2.0.1.0".

Figure 1–1 Example of an Oracle Database Release Number



Note: Starting with release 9.2, maintenance releases of Oracle Database 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

The first digit is the most general identifier. It represents a major new version of the software that contains significant new functionality.

Database Maintenance Release Number

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

Fusion Middleware Release Number

The third digit reflects the release level of Oracle Fusion Middleware.

Component-Specific Release Number

The fourth 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

The fifth digit identifies a platform-specific release. Usually this is a patch set. When different platforms require the equivalent patch set, this digit will be the same across the affected platforms.

Checking Your Current Release Number

To identify the release of Oracle Database that is currently installed and to see the release levels of other database components you are using, query the data dictionary view `PRODUCT_COMPONENT_VERSION`. A sample query follows. (You can also query the `V$VERSION` view to see component-level information.) Other product release levels may increment independent of the database server.

```
COL PRODUCT FORMAT A40
COL VERSION FORMAT A15
COL STATUS FORMAT A15
```

```
SELECT * FROM PRODUCT_COMPONENT_VERSION;

PRODUCT                                  VERSION     STATUS
---------------------------------------- ----------- -----------
NLSRTL                                   11.2.0.0.1  Production
Oracle Database 11g Enterprise Edition   11.2.0.0.1  Production
PL/SQL                                   11.2.0.0.1  Production
...
```

It is important to convey to Oracle the results of this query when you report problems with the software.

About Database Administrator Security and Privileges

To perform the administrative tasks of an Oracle Database DBA, you need specific privileges 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
- Administrative User Accounts

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 on which Oracle Database is running, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require operating system privileges or access rights that other database users do not require (for example, to perform Oracle Database software installation). Although you do not need the Oracle Database files to be stored in your account, you should have access to them.

> **See Also:** Your operating system-specific Oracle documentation. The method of creating the account of the database administrator is specific to the operating system.

Administrative User Accounts

Two administrative user accounts are automatically created when Oracle Database is installed:

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

> **Note:** Both Oracle Universal Installer (OUI) and Database Configuration Assistant (DBCA) now prompt for SYS and SYSTEM passwords and do not accept the default passwords "change_on_install" or "manager", respectively.
>
> If you create the database manually, Oracle strongly recommends that you specify passwords for SYS and SYSTEM at database creation time, rather than using these default passwords. See "Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM" on page 2-16 for more information.

Create at least one additional administrative user and grant to that user an appropriate administrative role to use when performing daily administrative tasks. Do not use SYS and SYSTEM for these purposes.

> **Note Regarding Security Enhancements:** In this release of Oracle Database and in subsequent releases, several enhancements are being made to ensure the security of default database user accounts. You can find a security checklist for this release in *Oracle Database Security Guide*. Oracle recommends that you read this checklist and configure your database accordingly.

SYS

When you create an Oracle database, the user SYS is automatically created and granted the DBA role.

All of the base tables and views for the database data dictionary are stored in the schema SYS. These base tables and views are critical for the operation of Oracle Database. To maintain the integrity of the data dictionary, tables in the SYS schema are manipulated only by the database. 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 to Oracle Database using the SYS account.

SYSTEM

When you create an Oracle Database, the user SYSTEM is also automatically created and granted the DBA role.

The SYSTEM user name is used to create additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEM schema to store tables of interest to non-administrative users.

The DBA Role

A predefined DBA role is automatically created with every Oracle Database installation. This role contains most database system privileges. Therefore, the DBA role should be granted only to actual 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 1-16.

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 for Database Administrators
- Using Operating System 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.
>
> The SYSDBA and SYSOPER privileges can also be thought of as types of connections that enable you to perform certain database operations for which privileges cannot be granted in any other fashion. For example, if you have the SYSDBA privilege, you can connect to the database by specifying CONNECT AS SYSDBA.

SYSDBA and SYSOPER

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

System Privilege	Operations Authorized
SYSDBA	■ Perform STARTUP and SHUTDOWN operations ■ ALTER DATABASE: open, mount, back up, or change character set ■ CREATE DATABASE ■ DROP DATABASE ■ CREATE SPFILE ■ ALTER DATABASE ARCHIVELOG ■ ALTER DATABASE RECOVER ■ Includes the RESTRICTED SESSION privilege Effectively, this system privilege allows a user to connect as user SYS.
SYSOPER	■ Perform STARTUP and SHUTDOWN operations ■ CREATE SPFILE ■ ALTER DATABASE OPEN/MOUNT/BACKUP ■ ALTER DATABASE ARCHIVELOG ■ ALTER DATABASE RECOVER (Complete recovery only. Any form of incomplete recovery, such as UNTIL TIME|CHANGE|CANCEL|CONTROLFILE requires connecting as SYSDBA.) ■ Includes the RESTRICTED SESSION privilege This privilege allows a user to perform basic operational tasks, but without the ability to look at user data.

The manner in which you are authorized to use these privileges depends upon the method of authentication that you 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 the sample user oe has been granted the SYSDBA system privilege and has issued the following statements:

```
CONNECT oe
CREATE TABLE admin_test(name VARCHAR2(20));
```

Later, user oe issues these statements:

```
CONNECT oe AS SYSDBA
SELECT * FROM admin_test;
```

User oe now receives the following error:

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

Having connected as SYSDBA, user oe now references the SYS schema, but the table was created in the oe schema.

See Also:

- "Using Operating System Authentication" on page 1-20
- "Using Password File Authentication" on page 1-21

Selecting an Authentication Method for Database Administrators

Database Administrators can authenticate database administrators through the data dictionary, (using an account password) like other users. Keep in mind that beginning with Oracle Database 11*g* Release 1, database passwords are case-sensitive. (You can disable case sensitivity and return to pre–Release 11*g* behavior by setting the `SEC_CASE_SENSITIVE_LOGON` initialization parameter to `FALSE`.)

In addition to normal data dictionary authentication, the following methods are available for authenticating database administrators with the `SYSDBA` or `SYSOPER` privilege:

- Operating system (OS) authentication
- Password files
- Strong authentication with a network-based authentication service, such as Oracle Internet Directory

These methods are required to authenticate a database administrator when the database is not started or otherwise unavailable. (They can also be used when the database is available.)

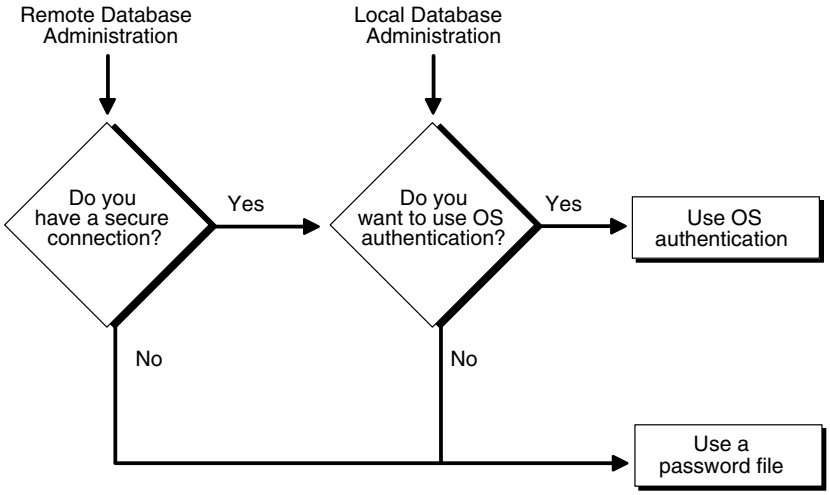
The remainder of this section focuses on operating system authentication and password file authentication. See *Oracle Database Security Guide* for information about authenticating database administrators with network-based authentication services.

Notes:

- These methods replace the `CONNECT INTERNAL` syntax provided with earlier versions of Oracle Database. `CONNECT INTERNAL` is no longer supported.
- Operating system authentication takes precedence over password file authentication. If you meet the requirements for operating system authentication, then even if you use a password file, you will be authenticated by operating system authentication.

Your choice will be influenced by whether you intend to administer your database locally on the same system 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, consult your Oracle Net documentation to determine whether you are using a secure connection. Most popular connection protocols, such as TCP/IP and DECnet, are not secure.

See Also:

- *Oracle Database Security Guide* for information about authenticating database administrators with network-based authentication services.
- *Oracle Database Net Services Administrator's Guide*

Nonsecure Remote Connections

To connect to Oracle Database as a privileged user over a nonsecure 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 user names that have been granted the `SYSDBA` or `SYSOPER` system privilege. This form of authentication is discussed in "Using Password File Authentication" on page 1-21.

Local Connections and Secure Remote Connections

You can connect to Oracle Database as a privileged user over a local connection or a secure remote connection in two ways:

- If the database has a password file and you have been granted the `SYSDBA` or `SYSOPER` system privilege, then you can connect and be authenticated by a password file.
- 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 operating system authentication. On most operating systems, authentication for database administrators involves placing the operating system username of the database administrator in a special group, generically referred to as OSDBA. Users in that group are granted `SYSDBA` privileges. A similar group, OSOPER, is used to grant `SYSOPER` privileges to users.

Using Operating System Authentication

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

OSDBA and OSOPER

Membership in one of two special operating system groups enables a DBA to authenticate to the database through the operating system rather than with a database user name and password. This is known as operating system authentication. These operating system 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 default names vary depending upon your operating system, and are listed in the following table:

Operating System Group	UNIX User Group	Windows User Group
OSDBA	`dba`	`ORA_DBA`
OSOPER	`oper`	`ORA_OPER`

Oracle Universal Installer uses these default names, but you can override them. One reason to override them is if you have multiple instances running on the same host computer. If each instance is to have a different person as the principal DBA, you can improve the security of each instance by creating a different OSDBA group for each instance. For example, for two instances on the same host, the OSDBA group for the first instance could be named `dba1`, and OSDBA for the second instance could be named `dba2`. The first DBA would be a member of `dba1` only, and the second DBA would be a member of `dba2` only. Thus, when using operating system authentication, each DBA would be able to connect only to his assigned instance.

Membership in the OSDBA or OSOPER group affects your connection to the database in the following ways:

- If you are a member of the OSDBA group and you specify `AS SYSDBA` when you connect to the database, then you connect to the database with the `SYSDBA` system privilege.
- If you are a member of the OSOPER group and you specify `AS SYSOPER` when you connect to the database, then you connect to the database with the `SYSOPER` system privilege.
- If you are not a member of either of these operating system groups and you attempt to connect as `SYSDBA` or `SYSOPER`, the `CONNECT` command fails.

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

Preparing to Use Operating System Authentication

To enable operating system authentication of an administrative user:

1. Create an operating system account for the user.
2. Add the account to the `OSDBA` or `OSOPER` operating system defined groups.

Connecting Using Operating System 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 the Windows platform only, remote operating system authentication over a secure connection is supported. You must specify the net service name for the remote database:

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

Both the client computer and database host computer must be on a Windows domain.

See Also:

- "Connecting to the Database with SQL*Plus" on page 1-7
- *SQL*Plus User's Guide and Reference* for syntax of the `CONNECT` command

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. If not already created, create the password file using the `ORAPWD` utility:

   ```
   ORAPWD FILE=filename ENTRIES=max_users
   ```

   See "Creating and Maintaining a Password File" on page 1-22 for details.

   Notes:

   - When you invoke Database Configuration Assistant (DBCA) as part of the Oracle Database installation process, DBCA creates a password file.
   - Beginning with Oracle Database 11*g* Release 1, passwords in the password file are case-sensitive unless you include the `IGNORECASE = Y` command-line argument.

2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE`. (This is the default).

   Note: `REMOTE_LOGIN_PASSWORDFILE` is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect to the database as user `SYS` (or as another user with the administrative privileges).
4. If the user does not already exist in the database, create the user and assign a password.

   Keep in mind that beginning with Oracle Database 11*g* Release 1, database passwords are case-sensitive. (You can disable case sensitivity and return to

pre–Release 11*g* behavior by setting the SEC_CASE_SENSITIVE_LOGON initialization parameter to FALSE.)

5. Grant the SYSDBA or SYSOPER system privilege to the user:

```
GRANT SYSDBA to oe;
```

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

> **See Also:** "Creating and Maintaining a Password File" on page 1-22 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 the AS SYSDBA or AS SYSOPER clause. Note that beginning with Oracle Database 11*g* Release 1, passwords are case-sensitive unless the password file was created with the IGNORECASE = Y option.

For example, user oe has been granted the SYSDBA privilege, so oe can connect as follows:

```
CONNECT oe AS SYSDBA
```

However, user oe has not been granted the SYSOPER privilege, so the following command will fail:

```
CONNECT oe 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 attempting to connect as SYSDBA or as SYSOPER fails.

> **See Also:**
>
> - "Connecting to the Database with SQL*Plus" on page 1-7
> - *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:

- Creating a Password File with ORAPWD
- Sharing and Disabling the Password File

- Adding Users to a Password File
- Maintaining a Password File

See Also:

- "Using Password File Authentication" on page 1-21
- "Selecting an Authentication Method for Database Administrators" on page 1-18

Creating a Password File with ORAPWD

The syntax of the `ORAPWD` command is as follows:

```
ORAPWD FILE=filename [ENTRIES=numusers] [FORCE={Y|N}] [IGNORECASE={Y|N}]
```

Command arguments are summarized in the following table.

Argument	Description
`FILE`	Name to assign to the password file. You must supply a complete path. If you supply only a file name, the file is written to the current directory.
`ENTRIES`	(Optional) Maximum number of entries (user accounts) to permit in the file.
`FORCE`	(Optional) If `y`, permits overwriting an existing password file.
`IGNORECASE`	(Optional) If `y`, passwords are treated as case-insensitive.

There are no spaces permitted around the equal-to (=) character.

The command prompts for the `SYS` password and stores the password in the created password file.

Example

The following command creates a password file named `orapworcl` that allows up to 30 privileged users with different passwords.

```
orapwd FILE=orapworcl ENTRIES=30
```

ORAPWD Command Line Argument Descriptions

The following sections describe the `ORAPWD` command line arguments.

FILE

This argument 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 argument is mandatory.

The file name required for the password file is operating system specific. Some operating systems require the password file to adhere to a specific format and be located in a specific directory. Other operating systems allow the use of environment variables to specify the name and location of the password file.

Table 1–1 lists the required name and location for the password file on the UNIX, Linux, and Windows platforms. For other platforms, consult your platform-specific documentation.

Table 1–1 Required Password File Name and Location on UNIX, Linux, and Windows

Platform	Required Name	Required Location)
UNIX and Linux	orapw*ORACLE_SID*	*ORACLE_HOME*/dbs
Windows	PWD*ORACLE_SID*.ora	*ORACLE_HOME*\database

For example, for a database instance with the SID orcldw, the password file must be named orapworcldw on Linux and PWDorcldw.ora on Windows.

In an Oracle Real Application Clusters environment on a platform that requires an environment variable to be set to the path of the password file, the environment variable for each instance must 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.

ENTRIES

This argument 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 a multiple of four.

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

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

FORCE

This argument, if set to Y, enables you to overwrite an existing password file. An error is returned if a password file of the same name already exists and this argument is omitted or set to N.

IGNORECASE

If this argument is set to y, passwords are case-insensitive. That is, case is ignored when comparing the password that the user supplies during login with the password in the password file.

> **See Also:** *Oracle Database Security Guide* for more information about case-sensitivity in passwords.

Sharing and Disabling the Password File

You use the initialization parameter REMOTE_LOGIN_PASSWORDFILE to control whether a password file is shared among multiple Oracle Database instances. You can also use

this parameter to disable password file authentication. The values recognized for REMOTE_LOGIN_PASSWORDFILE are:

- NONE: Setting this parameter to NONE causes Oracle Database to behave as if the password file does not exist. That is, no privileged connections are allowed over nonsecure connections.
- EXCLUSIVE: (The default) An EXCLUSIVE password file can be used with only one instance of one database. Only an EXCLUSIVE file can be modified. Using an EXCLUSIVE password file enables you to add, modify, and delete users. It also enables you to change the SYS password with the ALTER USER command.
- SHARED: A SHARED password file can be used by multiple databases running on the same server, or multiple instances of an Oracle Real Application Clusters (Oracle RAC) database. A SHARED password file cannot be modified. Therefore, you cannot add users to a SHARED password file. Any attempt to do so or to change the password of SYS or other users with the SYSDBA or SYSOPER privileges generates an error. All users needing SYSDBA or SYSOPER system privileges must be added to the password file when REMOTE_LOGIN_PASSWORDFILE is set to EXCLUSIVE. After all users are added, you can change REMOTE_LOGIN_PASSWORDFILE to SHARED, and then share the file.

  This option is useful if you are administering multiple databases or an Oracle RAC database.

If REMOTE_LOGIN_PASSWORDFILE is set to EXCLUSIVE or SHARED and the password file is missing, this is equivalent to setting REMOTE_LOGIN_PASSWORDFILE to NONE.

Note: You cannot change the password for SYS if REMOTE_LOGIN_PASSWORDFILE is set to SHARED. An error message is issued if you attempt to do so.

Keeping Administrator Passwords Synchronized with the Data Dictionary

If you change the REMOTE_LOGIN_PASSWORDFILE initialization parameter from NONE to EXCLUSIVE or SHARED, or if you re-create the password file with a different SYS password, then you must ensure that the passwords in the data dictionary and password file for the SYS user are the same.

To synchronize the SYS passwords, use the ALTER USER statement to change the SYS password. The ALTER USER statement updates and synchronizes both the dictionary and password file passwords.

To synchronize the passwords for non-SYS users who log in using the SYSDBA or SYSOPER privilege, you must revoke and then regrant the privilege to the user, as follows:

1. Find all users who have been granted the SYSDBA privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSDBA='TRUE';
   ```

2. Revoke and then re-grant the SYSDBA privilege to these users.

   ```
   REVOKE SYSDBA FROM non-SYS-user;
   GRANT SYSDBA TO non-SYS-user;
   ```

3. Find all users who have been granted the SYSOPER privilege.

   ```
   SELECT USERNAME FROM V$PWFILE_USERS WHERE USERNAME != 'SYS' AND SYSOPER='TRUE';
   ```

4. Revoke and regrant the SYSOPER privilege to these users.

```
REVOKE SYSOPER FROM non-SYS-user;
GRANT SYSOPER TO non-SYS-user;
```

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, or the password file is missing), Oracle Database issues an error 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, Oracle Database removes the user from the password file.

Creating a Password File and Adding New Users to It

Use the following procedure to create a password and add new users to it:

1. Follow the instructions for creating a password file as explained in "Creating a Password File with ORAPWD" on page 1-23.
2. Set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE. (This is the default.)

   Note: REMOTE_LOGIN_PASSWORDFILE is a static initialization parameter and therefore cannot be changed without restarting the database.

3. Connect with SYSDBA privileges as shown in the following example, and enter the SYS password when prompted:

   ```
   CONNECT SYS 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", later in this section.

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 oe;
```

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

```
REVOKE SYSDBA FROM oe;
```

Because SYSDBA and SYSOPER are the most powerful database privileges, the WITH ADMIN OPTION is not used in the GRANT statement. That is, the grantee cannot in turn grant the SYSDBA or SYSOPER privilege to another user. Only a user currently connected as SYSDBA can grant or revoke another user's SYSDBA or SYSOPER system privileges. These privileges cannot be granted to roles, because roles are available only after

database startup. Do not confuse the SYSDBA and SYSOPER database privileges with operating system roles.

> **See Also:** *Oracle Database Security Guide* for more information on system privileges

Viewing Password File Members

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

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

> **Note:** SYSASM is valid only for Oracle Automatic Storage Management instances.

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

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 regrant the privileges to the users.

Replacing a Password File

Use the following procedure to replace a password file:

1. Identify the users who have SYSDBA or SYSOPER privileges by querying the V$PWFILE_USERS view.
2. Delete the existing password file.
3. Follow the instructions for creating a new password file using the ORAPWD utility in "Creating a Password File with ORAPWD" on page 1-23. Ensure that the ENTRIES parameter is set to a number larger than you think you will ever need.
4. Follow the instructions in "Adding Users to a Password File" on page 1-26.

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 then optionally 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 `SYSDBA` or `SYSOPER` database administration operations.

Data Utilities

Oracle utilities are available to help you maintain the data in your Oracle Database.

SQL*Loader

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

Export and Import Utilities

The Data Pump utility enables you to archive data and to move data between one Oracle Database and another. Also available are the original Import (IMP) and Export (EXP) utilities for importing and exporting data from and to earlier releases.

> **See Also:** *Oracle Database Utilities* for detailed information about these utilities

2

Creating and Configuring an Oracle Database

In this chapter:

- About Creating an Oracle Database
- Creating a Database with DBCA
- Creating a Database with the CREATE DATABASE Statement
- Specifying CREATE DATABASE Statement Clauses
- Specifying Initialization Parameters
- Managing Initialization Parameters Using a Server Parameter File
- Managing Application Workloads with Database Services
- Considerations After Creating a Database
- Dropping a Database
- Database Data Dictionary Views

See Also:

- Chapter 17, "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
- Your platform-specific Oracle Real Application Clusters (Oracle RAC) installation guide for information about creating a database in an Oracle RAC environment

About Creating an Oracle Database

After you plan your database using some of the guidelines presented in this section, you can create the database with a graphical tool or a SQL command. You typically create a database during Oracle Database software installation. However, you can also create a database after installation. Reasons to create a database after installation are as follows:

- You used Oracle Universal Installer (OUI) to install software only, and did not create a database.
- You want to create another database (and database instance) on the same host computer as an existing Oracle database. In this case, this chapter assumes that the

new database uses the same Oracle home as the existing database. You can also create the database in a new Oracle home by running OUI again.

- You want to make a copy of (clone) a database.

The specific methods for creating a database are:

- With Database Configuration Assistant (DBCA), a graphical tool.

  See "Creating a Database with DBCA" on page 2-5

- With the `CREATE DATABASE` SQL statement.

  See "Creating a Database with the CREATE DATABASE Statement" on page 2-6

Considerations Before Creating the Database

Database creation prepares several operating system files to work together as an Oracle Database. You only need to create a database once, regardless of how many datafiles it has or how many instances access it. You can create a database to 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

Planning for Database Creation

Prepare to create the database by research and careful planning. Table 2–1 lists some recommended actions:

Table 2–1 Database Planning Tasks

Action	Additional Information
Plan the database tables and indexes and estimate the amount of space they will require.	Part II, "Oracle Database Structure and Storage" Part III, "Schema Objects"
Plan the layout of the underlying operating system files your database will comprise. Proper distribution of files can improve database performance dramatically by distributing the I/O during file access. You can distribute I/O in several ways when you install Oracle software and create your database. For example, you can place redo log files on separate disks or use striping. You can situate datafiles to reduce contention. And you can control data density (number of rows to a data block). If you create a Fast Recovery Area, Oracle recommends that you place it on a storage device that is different from that of the datafiles. To greatly simplify this planning task, consider using Oracle Managed Files and Automatic Storage Management to create and manage the operating system files that comprise your database storage.	Chapter 17, "Using Oracle Managed Files" *Oracle Automatic Storage Management Administrator's Guide* *Oracle Database Performance Tuning Guide* *Oracle Database Backup and Recovery User's Guide* Your Oracle operating system–specific documentation, including the appropriate Oracle Database installation guide.
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.	"Determining the Global Database Name" on page 2-26

Table 2–1 (Cont.) Database Planning Tasks

Action	Additional Information
Familiarize yourself with the initialization parameters contained in the initialization parameter file. Become familiar with the concept and operation of a **server parameter file**. A server parameter file lets you store and manage your initialization parameters persistently in a server-side disk file.	"About Initialization Parameters and Initialization Parameter Files" on page 2-24 "What Is a Server Parameter File?" on page 2-32 *Oracle Database Reference*
Select the database character set. All character data, including data in the data dictionary, is stored in the database character set. You specify the database character set when you create the database. See "Selecting a Character Set" on page 2-3 for details.	*Oracle Database Globalization Support Guide*
Consider what time zones your database must support. Oracle Database uses one of two time zone files as the source of valid time zones. The default time zone file is `timezlrg_11.dat`. It contains more time zones than the smaller time zone file, `timezone_11.dat`.	"Specifying the Database Time Zone File" on page 2-22
Select the standard database block size. This is specified at database creation by the `DB_BLOCK_SIZE` initialization parameter and cannot be changed after the database is created. The `SYSTEM` tablespace and most other tablespaces use the standard block size. Additionally, you can specify up to four nonstandard block sizes when creating tablespaces.	"Specifying Database Block Sizes" on page 2-28
If you plan to store online redo log files on disks with a 4K byte sector size, determine whether you must manually specify redo log block size.	"Planning the Block Size of Redo Log Files" on page 12-7
Determine the appropriate initial sizing for the `SYSAUX` tablespace.	"About the SYSAUX Tablespace" on page 2-17
Plan to use a default tablespace for non-`SYSTEM` users to prevent inadvertently saving database objects in the `SYSTEM` tablespace.	"Creating a Default Permanent Tablespace" on page 2-18
Plan to use an undo tablespace to manage your undo data.	Chapter 16, "Managing Undo"
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.	Chapter 12, "Managing the Redo Log" Chapter 13, "Managing Archived Redo Logs" Chapter 11, "Managing Control Files" *Oracle Database Backup and Recovery User's Guide*
Familiarize yourself with the principles and options of starting up and shutting down an instance and mounting and opening a database.	Chapter 3, "Starting Up and Shutting Down"

Selecting a Character Set Oracle recommends AL32UTF8 as the database character set. AL32UTF8 is Oracle's name for the UTF-8 encoding of the Unicode standard. The Unicode standard is the universal character set that supports most of the currently

spoken languages of the world. The use of the Unicode standard is indispensable for any multilingual technology, including database processing.

After a database is created and accumulates production data, changing the database character set is a time consuming and complex project. Therefore, it is very important to select the right character set at installation time. Even if the database does not currently store multilingual data but is expected to store multilingual data within a few years, the choice of AL32UTF8 for the database character set is usually the only good decision.

Even so, the default character set used by Oracle Universal Installer (OUI) and Database Configuration Assistant (DBCA) for the UNIX, Linux, and Microsoft Windows platforms is not AL32UTF8, but a Microsoft Windows character set known as an ANSI code page. The particular character set is selected based on the current language (locale) of the operating system session that started OUI or DBCA. If the language is American English or one of the Western European languages, the default character set is WE8MSWIN1252. Each Microsoft Windows ANSI Code Page is capable of storing data only from one language or a limited group of languages, such as only Western European, or only Eastern European, or only Japanese.

A Microsoft Windows character set is the default even for databases created on UNIX and Linux platforms because Microsoft Windows is the prevalent platform for client workstations. Oracle Client libraries automatically perform the necessary character set conversion between the database character set and the character sets used by non-Windows client applications.

You may also choose to use any other character set from the presented list of character sets. You can use this option to select a particular character set required by an application vendor, or choose a particular character set that is the common character set used by all clients connecting to this database.

As AL32UTF8 is a multibyte character set, database operations on character data may be slightly slower when compared to single-byte database character sets, such as WE8MSWIN1252. Storage space requirements for text in most languages that use characters outside of the ASCII repertoire are higher in AL32UTF8 compared to legacy character sets supporting the language. The increase in storage space concerns only character data and only data that is not in English. The universality and flexibility of Unicode usually outweighs these additional costs.

Caution: Do not use the character set named UTF8 as the database character set unless required for compatibility with Oracle Database clients and servers in version 8.1.7 and earlier, or unless explicitly requested by your application vendor. Despite having a very similar name, UTF8 is not a proper implementation of the Unicode encoding UTF-8. If the UTF8 character set is used where UTF-8 processing is expected, data loss and security issues may occur. This is especially true for Web related data, such as XML and URL addresses.

Meeting Creation Prerequisites

Before you can create a new database, the following prerequisites must be met:

- The desired Oracle software must be installed. This includes setting various environment variables unique to your operating system and establishing the directory structure for software and database files.
- Sufficient memory must be available to start the Oracle Database instance.

- Sufficient disk storage space must be available for the planned database on the computer that runs Oracle Database.

All of these are discussed in the *Oracle Database Installation Guide* specific to your operating system. If you use the Oracle Universal Installer, it will guide you through your installation and provide help in setting environment variables and establishing directory structure and authorizations.

Creating a Database with DBCA

Database Configuration Assistant (DBCA) is the preferred way to create a database, because it is a more automated approach, and your database is ready to use when DBCA completes. DBCA can be launched by the Oracle Universal Installer (OUI), depending upon the type of install that you select. You can also launch DBCA as a standalone tool at any time after Oracle Database installation.

You can run DBCA in interactive mode or noninteractive/silent mode. Interactive mode provides a graphical interface and guided workflow for creating and configuring a database. Noninteractive/silent mode enables you to script database creation. You can run DBCA in noninteractive/silent mode by specifying command-line arguments, a response file, or both.

Creating a Database with Interactive DBCA

See *Oracle Database 2 Day DBA* for detailed information about creating a database interactively with DBCA.

Creating a Database with Noninteractive/Silent DBCA

See Appendix A of the installation guide for your platform for details on using the noninteractive/silent mode of DBCA.

The following example creates a database by passing command-line arguments to DBCA:

```
dbca -silent -createDatabase -templateName General_Purpose.dbc
  -gdbname ora11g -sid ora11g -responseFile NO_VALUE -characterSet AL32UTF8
  -memoryPercentage 30 -emConfiguration LOCAL

Enter SYSTEM user password:
password
Enter SYS user password:
password
Copying database files
1% complete
3% complete
...
```

To ensure completely silent operation, you can redirect stdout to a file. In this case, however, you must supply passwords for the administrative accounts in command-line arguments or the response file.

To view brief help for DBCA command-line arguments, enter the following command:

```
dbca -help
```

For more detailed argument information, including defaults, view the response file template found on your distribution media. Appendix A of your platform installation guide provides the name and location of this file.

Creating a Database with the CREATE DATABASE Statement

Using the `CREATE DATBASE` SQL statement is a more manual approach to creating a database. One advantage of using this statement over using DBCA is that you can create databases from within scripts.

If you use the `CREATE DATABASE` statement, you must complete additional actions before you have an operational database. These actions include building views on the data dictionary tables and installing standard PL/SQL packages. You perform these actions by running the supplied scripts.

If you have existing scripts for creating your database, consider editing those scripts to take advantage of new Oracle Database features.

The instructions in this section apply to *single-instance installations only*. See the Oracle Real Application Clusters (Oracle RAC) installation guide for your platform for instructions for creating an Oracle RAC database.

> **Note:** *Single-instance* does not mean that only one Oracle instance can reside on a single host computer. In fact, multiple Oracle instances (and their associated databases) can run on a single host computer. A **single-instance database** is a database that is accessed by only one Oracle instance, as opposed to an Oracle RAC database, which is accessed concurrently by multiple Oracle instances on multiple nodes. See *Oracle Real Application Clusters Administration and Deployment Guide* for more information on Oracle RAC.

Complete the following steps to create a database with the `CREATE DATABASE` statement. The examples create a database named `mynewdb`.

Step 1: Specify an Instance Identifier (SID)

Step 2: Ensure That the Required Environment Variables Are Set

Step 3: Choose a Database Administrator Authentication Method

Step 4: Create the Initialization Parameter File

Step 5: (Windows Only) Create an Instance

Step 6: Connect to the Instance

Step 7: Create a Server Parameter File

Step 8: Start the Instance

Step 9: Issue the CREATE DATABASE Statement

Step 10: Create Additional Tablespaces

Step 11: Run Scripts to Build Data Dictionary Views

Step 12: (Optional) Run Scripts to Install Additional Options

Step 13: Back Up the Database.

Step 14: (Optional) Enable Automatic Instance Startup

> **Tip:** If you are using Oracle Automatic Storage Management (Oracle ASM) to manage your disk storage, you must start the Oracle ASM instance and configure your disk groups before performing these steps. For information about Automatic Storage Management, see *Oracle Automatic Storage Management Administrator's Guide*.

Step 1: Specify an Instance Identifier (SID)

Decide on a unique Oracle system identifier (SID) for your instance, open a command window, and set the ORACLE_SID environment variable. Use this command window for the subsequent steps.

ORACLE_SID is used to distinguish this instance from other Oracle Database instances that you may create later and run concurrently on the same host computer. The maximum number of characters for ORACLE_SID is 12, and only letters and numeric digits are permitted. On some platforms, the SID is case-sensitive.

> **Note:** It is common practice to set the SID to be equal to the database name. The maximum number of characters for the database name is eight. For more information, see the discussion of the DB_NAME initialization parameter in *Oracle Database Reference*.

The following example for UNIX and Linux operating systems sets the SID for the instance that you will connect to in Step 6: Connect to the Instance:

- Bourne, Bash, or Korn shell:

```
ORACLE_SID=mynewdb
export ORACLE_SID
```

- C shell:

```
setenv ORACLE_SID mynewdb
```

The following example sets the SID for the Windows operating system:

```
set ORACLE_SID=mynewdb
```

> **See Also:** *Oracle Database Concepts* for background information about the Oracle instance

Step 2: Ensure That the Required Environment Variables Are Set

Depending on your platform, before you can start SQL*Plus (as required in Step 6: Connect to the Instance), you may have to set environment variables, or at least verify that they are set properly.

For example, on most platforms, ORACLE_SID and ORACLE_HOME must be set. In addition, it is advisable to set the PATH variable to include the *ORACLE_HOME*/bin directory. On the UNIX and Linux platforms, you must set these environment variables manually. On the Windows platform, OUI automatically assigns values to ORACLE_HOME and ORACLE_SID in the Windows registry. If you did not create a database upon installation, OUI does not set ORACLE_SID in the registry, and you will have to set the ORACLE_SID environment variable when you create your database later.

Step 3: Choose a Database Administrator Authentication Method

You must be authenticated and granted appropriate system privileges in order to create a database. You can be authenticated as an administrator with the required privileges in the following ways:

- With a password file
- With operating system authentication

In this step, you decide on an authentication method.

To be authenticated with a password file, create the password file as described in "Creating and Maintaining a Password File" on page 1-22. To be authenticated with operating system authentication, ensure that you log in to the host computer with a user account that is a member of the appropriate operating system user group. On the UNIX and Linux platforms, for example, this is typically the `dba` user group. On the Windows platform, the user installing the Oracle software is automatically placed in the required user group.

See Also:

- "About Database Administrator Security and Privileges" on page 1-14
- "Database Administrator Authentication" on page 1-16 for information about password files and operating system authentication

Step 4: Create the Initialization Parameter File

When an Oracle instance starts, it reads an initialization parameter file. This file can be a text file, which can be created and modified with a text editor, or a binary file, which is created and dynamically modified by the database. The binary file, which is preferred, is called a **server parameter file**. In this step, you create a text initialization parameter file. In a later step, you create a server parameter file from the text file.

One way to create the text initialization parameter file is to edit the sample presented in "Sample Initialization Parameter File" on page 2-26.

If you create the initialization parameter file manually, ensure that it contains at least the parameters listed in Table 2–2. All other parameters not listed have default values.

Table 2–2 Recommended Minimum Initialization Parameters

Parameter Name	Mandatory	Notes
`DB_NAME`	Yes	Database identifier. Must correspond to the value used in the `CREATE DATABASE` statement. Maximum 8 characters.
`CONTROL_FILES`	No	Strongly recommended. If not provided, then the database instance creates one control file in the same location as the initialization parameter file. Providing this parameter enables you to multiplex control files. See "Creating Initial Control Files" on page 11-3 for more information.
`MEMORY_TARGET`	No	Sets the total amount of memory used by the instance and enables automatic memory management. You can choose other initialization parameters instead of this one for more manual control of memory usage. See "Configuring Memory Manually" on page 6-7.

For convenience, store your initialization parameter file in the Oracle Database default location, using the default file name. Then when you start your database, it will not be necessary to specify the `PFILE` clause of the `STARTUP` command, because Oracle Database automatically looks in the default location for the initialization parameter file.

For more information about initialization parameters and the initialization parameter file, including the default name and location of the initialization parameter file for your platform, see "About Initialization Parameters and Initialization Parameter Files" on page 2-24.

See Also:

- "Specifying Initialization Parameters" on page 2-24
- *Oracle Database Reference* for details on all initialization parameters

Step 5: (Windows Only) Create an Instance

On the Windows platform, before you can connect to an instance, you must manually create it if it does not already exist. The `ORADIM` command creates an Oracle instance by creating a new Windows service.

To create an instance:

- Enter the following command at a Windows command prompt:

  ```
  oradim -NEW -SID sid -STARTMODE MANUAL -PFILE pfile
  ```

  where *`sid`* is the desired SID (for example `mynewdb`) and *`pfile`* is the full path to the text initialization parameter file. This command creates the instance but does not start it.

  Caution: Do not set the `-STARTMODE` argument to `AUTO` at this point, because this causes the new instance to start and attempt to mount the database, which does not exist yet. You can change this parameter to `AUTO`, if desired, in Step 14.

See the section "Using ORADIM to Administer an Oracle Database Instance" in *Oracle Database Platform Guide for Microsoft Windows* for more information on the `ORADIM` command.

Step 6: Connect to the Instance

Start SQL*Plus and connect to your Oracle Database instance with the `SYSDBA` system privilege.

- To authenticate with a password file, enter the following commands, and then enter the `SYS` password when prompted:

  ```
  $ sqlplus /nolog
  SQL> CONNECT SYS AS SYSDBA
  ```

- To authenticate with operating system authentication, enter the following commands:

  ```
  $ sqlplus /nolog
  SQL> CONNECT / AS SYSDBA
  ```

SQL*Plus outputs the following message:

```
Connected to an idle instance.
```

Note: SQL*Plus may output a message similar to the following:

```
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
```

If so, the instance is already started. You may have connected to the wrong instance. Exit SQL*Plus with the `EXIT` command, check that `ORACLE_SID` is set properly, and repeat this step.

Step 7: Create a Server Parameter File

The server parameter file enables you to change initialization parameters with the `ALTER SYSTEM` command and persist the changes across a database shutdown and startup. You create the server parameter file from your edited text initialization file.

The following SQL*Plus command reads the text initialization parameter file (PFILE) with the default name from the default location, creates a server parameter file (SPFILE) from the text initialization parameter file, and writes the SPFILE to the default location with the default SPFILE name.

```
CREATE SPFILE FROM PFILE;
```

You can also supply the file name and path for both the PFILE and SPFILE if you are not using default names and locations.

Tip: The database must be restarted before the server parameter file takes effect.

Note: Although creating a server parameter file is optional at this point, it is recommended. If you do not create a server parameter file, the instance continues to read the text initialization parameter file whenever it starts.

Important—If you are using Oracle Managed Files and your initialization parameter file does not contain the `CONTROL_FILES` parameter, then you must create a server parameter file now so the database can save the names and location of the control files that it creates during the `CREATE DATABASE` statement. See "Specifying Oracle Managed Files at Database Creation" on page 2-19 for more information.

See Also:

- "Managing Initialization Parameters Using a Server Parameter File" on page 2-32
- *Oracle Database SQL Language Reference* for more information on the `CREATE SPFILE` command

Step 8: 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` clause. In this example, because the initialization

parameter file or server parameter file is stored in the default location, you are not required to specify the PFILE clause:

```
STARTUP NOMOUNT
```

At this point, the instance memory is allocated and its processes are started. The database itself does not yet exist.

See Also:

- Chapter 3, "Starting Up and Shutting Down", to learn how to use the STARTUP command
- "Managing Initialization Parameters Using a Server Parameter File" on page 2-32

Step 9: Issue the CREATE DATABASE Statement

To create the new database, use the CREATE DATABASE statement.

Example 1

The following statement creates database mynewdb. This database name must agree with the DB_NAME parameter in the initialization parameter file. This example assumes the following:

- The initialization parameter file specifies the number and location of control files with the CONTROL_FILES parameter.
- The directory /u01/app/oracle/oradata/mynewdb exists.
- The directories /u01/logs/my and /u02/logs/my exist.

```
CREATE DATABASE mynewdb
   USER SYS IDENTIFIED BY sys_password
   USER SYSTEM IDENTIFIED BY system_password
   LOGFILE GROUP 1 ('/u01/logs/my/redo01a.log','/u02/logs/my/redo01b.log') SIZE 100M BLOCKSIZE 512,
           GROUP 2 ('/u01/logs/my/redo02a.log','/u02/logs/my/redo02b.log') SIZE 100M BLOCKSIZE 512,
           GROUP 3 ('/u01/logs/my/redo03a.log','/u02/logs/my/redo03b.log') SIZE 100M BLOCKSIZE 512
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   EXTENT MANAGEMENT LOCAL
   DATAFILE '/u01/app/oracle/oradata/mynewdb/system01.dbf' SIZE 325M REUSE
   SYSAUX DATAFILE '/u01/app/oracle/oradata/mynewdb/sysaux01.dbf' SIZE 325M REUSE
   DEFAULT TABLESPACE users
      DATAFILE '/u01/app/oracle/oradata/mynewdb/users01.dbf'
      SIZE 500M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED
   DEFAULT TEMPORARY TABLESPACE tempts1
      TEMPFILE '/u01/app/oracle/oradata/mynewdb/temp01.dbf'
      SIZE 20M REUSE
   UNDO TABLESPACE undotbs
      DATAFILE '/u01/app/oracle/oradata/mynewdb/undotbs01.dbf'
      SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

A database is created with the following characteristics:

- The database is named mynewdb. Its global database name is mynewdb.us.example.com, where the domain portion (us.example.com) is taken

from the initialization parameter file. See "Determining the Global Database Name" on page 2-26.

- Three control files are created as specified by the CONTROL_FILES initialization parameter, which was set before database creation in the initialization parameter file. See "Sample Initialization Parameter File" on page 2-26 and "Specifying Control Files" on page 2-28.
- The passwords for user accounts SYS and SYSTEM are set to the values that you specified. Beginning with Release 11*g*, the passwords are case-sensitive. The two clauses that specify the passwords for SYS and SYSTEM are not mandatory in this release of Oracle Database. However, if you specify either clause, then 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 2-16.
- The new database has three redo log file groups, each with two members, as specified in the LOGFILE clause. MAXLOGFILES, MAXLOGMEMBERS, and MAXLOGHISTORY define limits for the redo log. See "Choosing the Number of Redo Log Files" on page 12-8. The block size for the redo logs is set to 512 bytes, the same size as physical sectors on disk. The BLOCKSIZE clause is optional if block size is to be the same as physical sector size (the default). Typical sector size and thus typical block size is 512. Permissible values for BLOCKSIZE are 512, 1024, and 4096. For newer disks with a 4K sector size, optionally specify BLOCKSIZE as 4096. See "Planning the Block Size of Redo Log Files" on page 12-7 for more information.
- 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 limited by and affected by operating system limits. For example, if you set MAXDATAFILES, Oracle Database 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 *Oracle Database SQL Language Reference* and your operating system–specific Oracle documentation.

- 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 /u01/app/oracle/oradata/mynewdb/system01.dbf is created as specified by the DATAFILE clause. If a file with that name already exists, then it is overwritten.
- The SYSTEM tablespace is created as a locally managed tablespace. See "Creating a Locally Managed SYSTEM Tablespace" on page 2-17.
- A SYSAUX tablespace is created, consisting of the operating system file /u01/app/oracle/oradata/mynewdb/sysaux01.dbf as specified in the SYSAUX DATAFILE clause. See "About the SYSAUX Tablespace" on page 2-17.

- The DEFAULT TABLESPACE clause creates and names a default permanent tablespace for this database.
- The DEFAULT TEMPORARY TABLESPACE clause creates and names a default temporary tablespace for this database. See "Creating a Default Temporary Tablespace" on page 2-18.
- The UNDO TABLESPACE clause creates and names an undo tablespace that is used to store undo data for this database if you have specified UNDO_MANAGEMENT=AUTO in the initialization parameter file. If you omit this parameter, it defaults to AUTO. See "Using Automatic Undo Management: Creating an Undo Tablespace" on page 2-18.
- Redo log files will not initially be archived, because the ARCHIVELOG clause is not specified in this CREATE DATABASE statement. This is customary during database creation. You can later use an ALTER DATABASE statement to switch to ARCHIVELOG mode. The initialization parameters in the initialization parameter file for mynewdb relating to archiving are LOG_ARCHIVE_DEST_1 and LOG_ARCHIVE_FORMAT. See Chapter 13, "Managing Archived Redo Logs".

Tips:

- Ensure that all directories used in the CREATE DATABASE statement exist. The CREATE DATABASE statement does not create directories.
- If you are not using Oracle Managed Files, then every tablespace clause must include a DATAFILE or TEMPFILE clause.
- If database creation fails, then you can look at the alert log to determine the reason for the failure and to determine corrective actions. See "Viewing the Alert Log" on page 9-20. If you receive an error message that contains a process number, then examine the trace file for that process. Look for the trace file that contains the process number in the trace file name. See "Finding Trace Files" on page 9-21 for more information.
- To resubmit the CREATE DATABASE statement after a failure, you must first shut down the instance and delete any files created by the previous CREATE DATABASE statement.

Example 2

This example illustrates creating a database with Oracle Managed Files, which enables you to use a much simpler CREATE DATABASE statement. To use Oracle Managed Files, the initialization parameter DB_CREATE_FILE_DEST must be set. This parameter defines the base directory for the various database files that the database creates and automatically names. The following statement is an example of setting this parameter in the initialization parameter file:

```
DB_CREATE_FILE_DEST='/u01/app/oracle/oradata'
```

With Oracle Managed Files and the following CREATE DATABASE statement, the database creates the SYSTEM and SYSAUX tablespaces, creates the additional tablespaces specified in the statement, and chooses default sizes and properties for all datafiles, control files, and redo log files. Note that these properties and the other default database properties set by this method may not be suitable for your production environment, so it is recommended that you examine the resulting configuration and modify it if necessary.

```
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY sys_password
```

```
USER SYSTEM IDENTIFIED BY system_password
EXTENT MANAGEMENT LOCAL
DEFAULT TEMPORARY TABLESPACE temp
UNDO TABLESPACE undotbs1
DEFAULT TABLESPACE users;
```

Tip: If your `CREATE DATABASE` statement fails, and if you did not complete Step 7, then ensure that there is not a pre-existing server parameter file (SPFILE) for this instance that is setting initialization parameters in an unexpected way. For example, an SPFILE contains a setting for the complete path to all control files, and the `CREATE DATABASE` statement fails if those control files do not exist. Ensure that you shut down and restart the instance (with `STARTUP NOMOUNT`) after removing an unwanted SPFILE. See "Managing Initialization Parameters Using a Server Parameter File" on page 2-32 for more information.

See Also:

- "Specifying CREATE DATABASE Statement Clauses" on page 2-15
- "Specifying Oracle Managed Files at Database Creation" on page 2-19
- Chapter 17, "Using Oracle Managed Files"
- *Oracle Database SQL Language Reference* for more information about specifying the clauses and parameter values for the `CREATE DATABASE` statement

Step 10: Create Additional Tablespaces

To make the database functional, you must create additional tablespaces for your application data. The following sample script creates some additional tablespaces:

```
CREATE TABLESPACE apps_tbs LOGGING
     DATAFILE '/u01/app/oracle/oradata/mynewdb/apps01.dbf'
     SIZE 500M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
-- create a tablespace for indexes, separate from user tablespace (optional)
CREATE TABLESPACE indx_tbs LOGGING
     DATAFILE '/u01/app/oracle/oradata/mynewdb/indx01.dbf'
     SIZE 100M REUSE AUTOEXTEND ON NEXT  1280K MAXSIZE UNLIMITED
     EXTENT MANAGEMENT LOCAL;
```

For information about creating tablespaces, see Chapter 14, "Managing Tablespaces".

Step 11: Run Scripts to Build Data Dictionary Views

Run the scripts necessary to build data dictionary views, synonyms, and PL/SQL packages, and to support proper functioning of SQL*Plus:

```
@?/rdbms/admin/catalog.sql
@?/rdbms/admin/catproc.sql
@?/sqlplus/admin/pupbld.sql
EXIT
```

The at-sign (@) is shorthand for the command that runs a SQL*Plus script. The question mark (?) is a SQL*Plus variable indicating the Oracle home directory. 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.
PUPBLD.SQL	Required for SQL*Plus. Enables SQL*Plus to disable commands by user.

Step 12: (Optional) Run Scripts to Install Additional Options

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 *Oracle Database Reference.*

If you plan to install other Oracle products to work with this database, then 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 data dictionary.

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

Step 13: Back Up the Database.

Take 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 *Oracle Database Backup and Recovery User's Guide.*

Step 14: (Optional) Enable Automatic Instance Startup

You might want to configure the Oracle instance to start automatically when its host computer restarts. See your operating system documentation for instructions. For example, on Windows, use the following command to configure the database service to start the instance upon computer restart:

```
ORADIM -EDIT -SID sid -STARTMODE AUTO -SRVCSTART SYSTEM [-SPFILE]
```

You must use the -SPFILE argument if you want the instance to read an SPFILE upon automatic restart.

See Also:

- Chapter 4, "Configuring Automatic Restart of an Oracle Database"
- The section "Using ORADIM to Administer an Oracle Database Instance" in *Oracle Database Platform Guide for Microsoft Windows* for more information on the ORADIM command.

Specifying CREATE DATABASE Statement Clauses

When you execute a CREATE DATABASE statement, Oracle Database performs several operations. The actual operations performed depend on the clauses that you specify in

the CREATE DATABASE statement and the initialization parameters that you have set. Oracle Database performs at least these operations:

- 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
- Creates the SYSAUX tablespace
- 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 9: Issue the CREATE DATABASE Statement" on page 2-11 and introduces additional ones. Many of the CREATE DATABASE clauses discussed here can be used to simplify the creation and management of your database.

The following topics are contained in this section:

- Protecting Your Database: Specifying Passwords for Users SYS and SYSTEM
- Creating a Locally Managed SYSTEM Tablespace
- About the SYSAUX Tablespace
- Using Automatic Undo Management: Creating an Undo Tablespace
- Creating a Default Temporary Tablespace
- Specifying Oracle Managed Files at Database Creation
- Supporting Bigfile Tablespaces During Database Creation
- 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 you omit these clauses, then these users are assigned the default passwords change_on_install and manager, respectively. A record is written to the alert log indicating that the default passwords were used. To protect your database, you must change these passwords using the ALTER USER statement immediately after database creation.

Oracle strongly recommends that you specify these clauses, even though they are optional in this release of Oracle Database. The default passwords are commonly known, and if you neglect to change them later, then you leave database vulnerable to attack by malicious users.

When choosing a password, keep in mind that beginning in Release 11*g*, passwords are case-sensitive. Also, there may be password formatting requirements for your

database. See the section entitled "How Oracle Database Checks the Complexity of Passwords" in *Oracle Database Security Guide* for more information.

See Also: "Some Security Considerations" on page 2-42

Creating a Locally Managed SYSTEM Tablespace

Specify the `EXTENT MANAGEMENT LOCAL` clause in the `CREATE DATABASE` statement to create a locally managed `SYSTEM` tablespace. The `COMPATIBLE` initialization parameter must be set to 10.0.0 or higher for this statement to be successful. If you do not specify the `EXTENT MANAGEMENT LOCAL` clause, then by default the database creates a dictionary-managed `SYSTEM` tablespace. Dictionary-managed tablespaces are deprecated.

If you create your database with a locally managed `SYSTEM` tablespace, and if you are not using Oracle Managed Files, then ensure that the following conditions are met:

- You specify the `DEFAULT TEMPORARY TABLESPACE` clause in the `CREATE DATABASE` statement.
- You include the `UNDO TABLESPACE` clause in the `CREATE DATABASE` statement.

See Also:

- *Oracle Database SQL Language 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 14-3
- "Migrating the SYSTEM Tablespace to a Locally Managed Tablespace" on page 14-29

About the SYSAUX Tablespace

The `SYSAUX` tablespace is always created at database creation. The `SYSAUX` tablespace serves as an auxiliary tablespace to the `SYSTEM` tablespace. Because it is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, it reduces the number of tablespaces required by the database. It also reduces the load on the `SYSTEM` tablespace.

You can specify only datafile attributes for the `SYSAUX` tablespace, using the `SYSAUX DATAFILE` clause in the `CREATE DATABASE` statement. Mandatory attributes of the `SYSAUX` tablespace are set by Oracle Database and include:

- `PERMANENT`
- `READ WRITE`
- `EXTENT MANAGMENT LOCAL`
- `SEGMENT SPACE MANAGMENT AUTO`

You cannot alter these attributes with an `ALTER TABLESPACE` statement, and any attempt to do so will result in an error. You cannot drop or rename the `SYSAUX` tablespace.

The size of the `SYSAUX` tablespace is determined by the size of the database components that occupy `SYSAUX`. You can view a list of these components by querying the `V$SYSAUX_OCCUPANTS` view. Based on the initial sizes of these components, the `SYSAUX` tablespace must be at least 400 MB at the time of database creation. The space

requirements of the SYSAUX tablespace will increase after the database is fully deployed, depending on the nature of its use and workload. For more information on how to manage the space consumption of the SYSAUX tablespace on an ongoing basis, see the "Managing the SYSAUX Tablespace" on page 14-25.

If you include a DATAFILE clause for the SYSTEM tablespace, then you must specify the SYSAUX DATAFILE clause as well, or the CREATE DATABASE statement will fail. This requirement does not exist if the Oracle Managed Files feature is enabled (see "Specifying Oracle Managed Files at Database Creation" on page 2-19).

The SYSAUX tablespace has the same security attributes as the SYSTEM tablespace.

Note: This documentation discusses the creation of the SYSAUX database at database creation. When upgrading from a release of Oracle Database that did not require the SYSAUX tablespace, you must create the SYSAUX tablespace as part of the upgrade process. This is discussed in *Oracle Database Upgrade Guide*.

See Also: "Managing the SYSAUX Tablespace" on page 14-25

Using Automatic Undo Management: Creating an Undo Tablespace

Automatic undo management uses an undo tablespace. To enable automatic undo management, set the UNDO_MANAGEMENT initialization parameter to AUTO in your initialization parameter file. Or, omit this parameter, and the database defaults to automatic undo management. In this mode, undo data is stored in an undo tablespace and is managed by Oracle Database. To define and name the undo tablespace yourself, you must include the UNDO TABLESPACE clause in the CREATE DATABASE statement at database creation time. If you omit this clause, and automatic undo management is enabled, then the database creates a default undo tablespace named SYS_UNDOTBS.

See Also:

- "Specifying the Method of Undo Space Management" on page 2-30
- Chapter 16, "Managing Undo", for information about the creation and use of undo tablespaces

Creating a Default Permanent Tablespace

The DEFAULT TABLESPACE clause of the CREATE DATABASE statement specifies a default permanent tablespace for the database. Oracle Database assigns to this tablespace any non-SYSTEM users for whom you do not explicitly specify a different permanent tablespace. If you do not specify this clause, then the SYSTEM tablespace is the default permanent tablespace for non-SYSTEM users. Oracle strongly recommends that you create a default permanent tablespace.

See Also: *Oracle Database SQL Language Reference* for the syntax of the DEFAULT TABLESPACE clause of CREATE DATABASE and ALTER DATABASE

Creating a Default Temporary Tablespace

The DEFAULT TEMPORARY TABLESPACE clause of the CREATE DATABASE statement creates a default temporary tablespace for the database. Oracle Database assigns this

tablespace as the temporary tablespace for users who are not explicitly assigned a temporary tablespace.

You can explicitly assign a temporary tablespace or tablespace group to a user in the `CREATE USER` statement. However, if you do not do so, and if no default temporary tablespace has been specified for the database, then by default these users are assigned the `SYSTEM` tablespace as their temporary tablespace. It is not good practice to store temporary data in the `SYSTEM` tablespace, and it is cumbersome to assign every user a temporary tablespace individually. Therefore, Oracle recommends that you use the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE`.

Note: When you specify a locally managed `SYSTEM` tablespace, the `SYSTEM` tablespace *cannot* be used as a temporary tablespace. In this case you must create a default temporary tablespace. This behavior is explained in "Creating a Locally Managed SYSTEM Tablespace" on page 2-17.

See Also:

- *Oracle Database SQL Language Reference* for the syntax of the `DEFAULT TEMPORARY TABLESPACE` clause of `CREATE DATABASE` and `ALTER DATABASE`
- "Temporary Tablespaces" on page 14-10 for information about creating and using temporary tablespaces
- "Multiple Temporary Tablespaces: Using Tablespace Groups" on page 14-13 for information about creating and using temporary tablespace groups

Specifying Oracle Managed Files at Database Creation

You can minimize the number of clauses and parameters that you specify in your `CREATE DATABASE` statement by using the Oracle Managed Files feature. You do this by specifying either a directory or Oracle Automatic Storage Management (Oracle ASM) disk group in which your files are created and managed by Oracle Database.

By including any of the initialization parameters `DB_CREATE_FILE_DEST`, `DB_CREATE_ONLINE_LOG_DEST_n`, or `DB_RECOVERY_FILE_DEST` in your initialization parameter file, you instruct Oracle Database to create and manage the underlying operating system files of your database. Oracle Database will automatically create and manage the operating system files for the following database structures, depending on which initialization parameters you specify and how you specify clauses in your `CREATE DATABASE` statement:

- Tablespaces and their datafiles
- Temporary tablespaces and their tempfiles
- Control files
- Redo log files
- Archived redo log files
- Flashback logs
- Block change tracking files
- RMAN backups

> **See Also:** "Specifying a Fast Recovery Area" on page 2-27 for information about setting initialization parameters that create a Fast Recovery Area

The following CREATE DATABASE statement shows briefly how the Oracle Managed Files feature works, assuming you have specified required initialization parameters:

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     EXTENT MANAGEMENT LOCAL
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1
     DEFAULT TABLESPACE users;
```

- The SYSTEM tablespace is created as a locally managed tablespace. Without the EXTENT MANAGEMENT LOCAL clause, the SYSTEM tablespace is created as dictionary managed, which is not recommended.
- No DATAFILE clause is specified, so the database creates an Oracle managed datafile for the SYSTEM tablespace.
- No LOGFILE clauses are included, so the database creates two Oracle managed redo log file groups.
- No SYSAUX DATAFILE is included, so the database creates an Oracle managed datafile for the SYSAUX tablespace.
- No DATAFILE subclause is specified for the UNDO TABLESPACE and DEFAULT TABLESPACE clauses, so the database creates an Oracle managed datafile for each of these tablespaces.
- No TEMPFILE subclause is specified for the DEFAULT TEMPORARY TABLESPACE clause, so the database creates an Oracle managed tempfile.
- If no CONTROL_FILES initialization parameter is specified in the initialization parameter file, then the database also creates an Oracle managed control file.
- If you are using a server parameter file (see "Managing Initialization Parameters Using a Server Parameter File" on page 2-32), then the database automatically sets the appropriate initialization parameters.

> **See Also:**
>
> - Chapter 17, "Using Oracle Managed Files", for information about the Oracle Managed Files feature and how to use it
> - *Oracle Automatic Storage Management Administrator's Guide*. for information about Automatic Storage Management

Supporting Bigfile Tablespaces During Database Creation

Oracle Database simplifies management of tablespaces and enables support for ultra-large databases by letting you create **bigfile tablespaces**. Bigfile tablespaces can contain only one file, but that file can have up to 4G blocks. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.

This section discusses the clauses of the CREATE DATABASE statement that let you include support for bigfile tablespaces.

> **See Also:** "Bigfile Tablespaces" on page 14-6 for more information about bigfile tablespaces

Specifying the Default Tablespace Type

The `SET DEFAULT...TABLESPACE` clause of the `CREATE DATABASE` statement determines the default type of tablespace for this database in subsequent `CREATE TABLESPACE` statements. Specify either `SET DEFAULT BIGFILE TABLESPACE` or `SET DEFAULT SMALLFILE TABLESPACE`. If you omit this clause, then the default is a **smallfile tablespace**, which is the traditional type of Oracle Database tablespace. A smallfile tablespace can contain up to 1022 files with up to 4M blocks each.

The use of bigfile tablespaces further enhances the Oracle Managed Files feature, because bigfile tablespaces make datafiles completely transparent for users. SQL syntax for the `ALTER TABLESPACE` statement has been extended to allow you to perform operations on tablespaces, rather than the underlying datafiles.

The `CREATE DATABASE` statement shown in "Specifying Oracle Managed Files at Database Creation" on page 2-19 can be modified as follows to specify that the default type of tablespace is a bigfile tablespace:

```
CREATE DATABASE mynewdb
     USER SYS IDENTIFIED BY sys_password
     USER SYSTEM IDENTIFIED BY system_password
     SET DEFAULT BIGFILE TABLESPACE
     UNDO TABLESPACE undotbs
     DEFAULT TEMPORARY TABLESPACE tempts1;
```

To dynamically change the default tablespace type after database creation, use the `SET DEFAULT TABLESPACE` clause of the `ALTER DATABASE` statement:

```
ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

You can determine the current default tablespace type for the database by querying the `DATABASE_PROPERTIES` data dictionary view as follows:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES
   WHERE PROPERTY_NAME = 'DEFAULT_TBS_TYPE';
```

Overriding the Default Tablespace Type

The `SYSTEM` and `SYSAUX` tablespaces are always created with the default tablespace type. However, you can explicitly override the default tablespace type for the `UNDO` and `DEFAULT TEMPORARY` tablespace during the `CREATE DATABASE` operation.

For example, you can create a bigfile `UNDO` tablespace in a database with the default tablespace type of smallfile as follows:

```
CREATE DATABASE mynewdb
...
     BIGFILE UNDO TABLESPACE undotbs
        DATAFILE '/u01/oracle/oradata/mynewdb/undotbs01.dbf'
        SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

You can create a smallfile `DEFAULT TEMPORARY` tablespace in a database with the default tablespace type of bigfile as follows:

```
CREATE DATABASE mynewdb
   SET DEFAULT BIGFILE TABLSPACE
...
     SMALLFILE DEFAULT TEMPORARY TABLESPACE tempts1
        TEMPFILE '/u01/oracle/oradata/mynewdb/temp01.dbf'
```

```
        SIZE 20M REUSE
...
```

Specifying the Database Time Zone and Time Zone File

This section contains:

- Setting the Database Time Zone
- About the Database Time Zone Files
- Specifying the Database Time Zone File

Setting the Database Time Zone

Set the database time zone when the database is created by using the `SET TIME_ZONE` clause of the `CREATE DATABASE` statement. If you do not set the database time zone, then it defaults to the time zone of the host operating system.

You can change the database time zone for a session by using the `SET TIME_ZONE` clause of the `ALTER SESSION` statement.

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

About the Database Time Zone Files

Two time zone files are included in a subdirectory of the Oracle home directory. The time zone files contain the valid time zone names. The following information is also included for each time zone:

- Offset from Coordinated Universal Time (UTC)
- Transition times for Daylight Saving Time
- Abbreviations for standard time and Daylight Saving Time

The default time zone file is *ORACLE_HOME*/oracore/zoneinfo/timezlrg_11.dat. A smaller time zone file with fewer time zones can be found in *ORACLE_HOME*/oracore/zoneinfo/timezone_11.dat.

To view the time zone names in the file being used by your database, use the following query:

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

> **See Also:** *Oracle Database Globalization Support Guide* for more information about managing and selecting time zone files

Specifying the Database Time Zone File

All databases that share information must use the same time zone datafile.

The database server always uses the large time zone file by default. If you would like to use the small time zone file on the client and know that all your data will refer only to regions in the small file, you can set the `ORA_TZFILE` environment variable on the client to the full path name of the timezone_*version*.dat file on the client, where *version* matches the time zone file version that is being used by the database server.

If you are already using the default larger time zone file on the client, then it is not practical to change to the smaller time zone file, because the database may contain data with time zones that are not part of the smaller file.

Specifying FORCE LOGGING Mode

Some data definition language statements (such as CREATE TABLE) allow the NOLOGGING clause, which causes some database operations not to generate redo records in the database redo log. The NOLOGGING setting can speed up operations that can be easily recovered outside of the database recovery mechanisms, but it can negatively affect media recovery and standby databases.

Oracle Database lets you force the writing of redo records even when NOLOGGING has been specified in DDL statements. The database never generates redo records for temporary tablespaces and temporary segments, so forced logging has no affect for objects.

> **See Also:** *Oracle Database SQL Language 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 take 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 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. However, if you re-create the control file, the database is 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 14-15 for information about using the FORCE LOGGING clause for tablespace creation.

Performance Considerations of FORCE LOGGING Mode

FORCE LOGGING mode results in some performance degradation. If the primary reason for specifying FORCE LOGGING is to ensure complete media recovery, and there is no standby database active, 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. Media recovery is not possible in

NOARCHIVELOG mode, so if you combine it with FORCE LOGGING, the result may be performance degradation with little benefit.

Specifying Initialization Parameters

This section introduces you to some of the basic initialization parameters you can add or edit before you create your new database. The following topics are covered:

- About Initialization Parameters and Initialization Parameter Files
- Determining the Global Database Name
- Specifying a Fast Recovery Area
- Specifying Control Files
- Specifying Database Block Sizes
- Specifying the Maximum Number of Processes
- Specifying the DDL Lock Timeout
- Specifying the Method of Undo Space Management
- About The COMPATIBLE Initialization Parameter
- Setting the License Parameter

See Also:

- *Oracle Database Reference* for descriptions of all initialization parameters including their default settings
- Chapter 6, "Managing Memory" for a discussion of the initialization parameters that pertain to memory management

About Initialization Parameters and Initialization Parameter Files

When an Oracle instance starts, it reads initialization parameters from an initialization parameter file. This file must at a minimum specify the DB_NAME parameter. All other parameters have default values.

The initialization parameter file can be either a read-only text file, a PFILE, or a read/write binary file.

The binary file is called a **server parameter file**. A server parameter file enables you to change initialization parameters with ALTER SYSTEM commands and to persist the changes across a shutdown and startup. It also provides a basis for self-tuning by Oracle Database. For these reasons, it is recommended that you use a server parameter file. You can create one manually from your edited text initialization file, or automatically by using Database Configuration Assistant (DBCA) to create your database.

Before you manually create a server parameter file, you can start an instance with a text initialization parameter file. Upon startup, the Oracle instance first searches for a server parameter file in a default location, and if it does not find one, searches for a text initialization parameter file. You can also override an existing server parameter file by naming a text initialization parameter file as an argument of the STARTUP command.

Default file names and locations for the text initialization parameter file are shown in the following table:

Platform	Default Name	Default Location

UNIX and Linux	`init`*`ORACLE_SID`*`.ora`
	For example, the initialization parameter file for the `mynewdb` database is named:
	`initmynewdb.ora`
		ORACLE_HOME`/dbs`

Windows	`init`*`ORACLE_SID`*`.ora`	*ORACLE_HOME*`\database`

If you are creating an Oracle database for the first time, Oracle suggests 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 text initialization parameter file, then your changes are effective only for the current instance. To make them permanent, you must update them manually in the initialization parameter file, or they will be lost over the next shutdown and startup of the database. If you are using a server parameter file, then initialization parameter file changes made by the `ALTER SYSTEM` statement can persist across shutdown and startup.

See Also:

- "Determining the Global Database Name" on page 2-26 for information about the `DB_NAME` parameter
- "Managing Initialization Parameters Using a Server Parameter File" on page 2-32
- "About Initialization Parameter Files and Startup" on page 3-3

Text Initialization Parameter File Format

The text initialization parameter file (PFILE) must contain name/value pairs in one of the following forms:

- For parameters that accept only a single value:

  ```
  parameter_name=value
  ```

- For parameters that accept one or more values (such as the `CONTROL_FILES` parameter):

  ```
  parameter_name=(value[,value] ...)
  ```

Parameter values of type string must be enclosed in single quotes ('). Case (upper or lower) in filenames is significant only if case is significant on the host operating system.

For parameters that accept multiple values, to enable you to easily copy and paste name/value pairs from the alert log, you can repeat a parameter on multiple lines, where each line contains a different value.

```
control_files='/u01/app/oracle/oradata/orcl/control01.ctl'
control_files='/u01/app/oracle/oradata/orcl/control02.ctl'
control_files='/u01/app/oracle/oradata/orcl/control03.ctl'
```

If you repeat a parameter that does not accept multiple values, then only the last value specified takes effect.

See Also:

- *Oracle Database Reference* for more information about the content and syntax of the text initialization parameter file
- "Alert Log" on page 9-5

Sample Initialization Parameter File

Oracle Database provides generally appropriate values in a sample text initialization parameter file. 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.

The sample text initialization parameter file is named `init.ora` and is found in the following location on most platforms:

ORACLE_HOME/dbs

The following is the content of the sample file:

```
###############################################################################
# Example INIT.ORA file
#
# This file is provided by Oracle Corporation to help you start by providing
# a starting point to customize your RDBMS installation for your site.
#
# NOTE: The values that are used in this file are only intended to be used
# as a starting point. You may want to adjust/tune those values to your
# specific hardware and needs. You may also consider using Database
# Configuration Assistant tool (DBCA) to create INIT file and to size your
# initial set of tablespaces based on the user input.
################################################################################

# Change '<ORACLE_BASE>' to point to the oracle base (the one you specify at
# install time)

db_name='ORCL'
memory_target=1G
processes = 150
audit_file_dest='<ORACLE_BASE>/admin/orcl/adump'
audit_trail ='db'
db_block_size=8192
db_domain=''
db_recovery_file_dest='<ORACLE_BASE>/flash_recovery_area'
db_recovery_file_dest_size=2G
diagnostic_dest='<ORACLE_BASE>'
dispatchers='(PROTOCOL=TCP) (SERVICE=ORCLXDB)'
open_cursors=300
remote_login_passwordfile='EXCLUSIVE'
undo_tablespace='UNDOTBS1'
# You may want to ensure that control files are created on separate physical
# devices
control_files = (ora_control1, ora_control2)
compatible ='11.2.0'
```

Determining the Global Database Name

The global database name consists of the user-specified local database name and the location of the database within a network structure. The `DB_NAME` initialization parameter determines the local name component of the database name, and the `DB_`

DOMAIN parameter, which is optional, 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.example.com, edit the parameters of the new parameter file as follows:

```
DB_NAME = test
DB_DOMAIN = us.example.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 files. Re-creating the control files is easily accomplished with the command ALTER DATABASE BACKUP CONTROLFILE TO TRACE. See *Oracle Database Backup and Recovery User's Guide* for more information.

> **See Also:** *Oracle 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 different, 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. If the database you are about to create will ever be part of a distributed database system, then give special attention to this initialization parameter before database creation. This parameter is optional.

> **See Also:** Part V, "Distributed Database Management" for more information about distributed databases

Specifying a Fast Recovery Area

The Fast Recovery Area is a location in which Oracle Database can store and manage files related to backup and recovery. It is distinct from the database area, which is a location for the current database files (datafiles, control files, and online redo logs).

You specify the Fast Recovery Area with the following initialization parameters:

- DB_RECOVERY_FILE_DEST: Location of the Fast Recovery Area. This can be a directory, file system, or Automatic Storage Management (Oracle ASM) disk group. It cannot be a raw file system.

  In an Oracle Real Application Clusters (Oracle RAC) environment, this location must be on a cluster file system, Oracle ASM disk group, or a shared directory configured through NFS.

- DB_RECOVERY_FILE_DEST_SIZE: Specifies the maximum total bytes to be used by the Fast Recovery Area. This initialization parameter must be specified before DB_RECOVERY_FILE_DEST is enabled.

In an Oracle RAC environment, the settings for these two parameters must be the same on all instances.

You cannot enable these parameters if you have set values for the LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters. You must disable those parameters before setting up the Fast Recovery Area. You can instead set values for the LOG_ARCHIVE_DEST_*n* parameters. The LOG_ARCHIVE_DEST_1 parameter is implicitly set to point to the Fast Recovery Area if a local archiving location has not been configured and LOG_ARCHIVE_DEST_1 value has not been set.

Oracle recommends using a Fast Recovery Area, because it can simplify backup and recovery operations for your database.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* to learn how to create and use a Fast Recovery Area

Specifying Control Files

The CONTROL_FILES initialization parameter specifies one or more control filenames for the database. When you execute the CREATE DATABASE statement, the control files listed in the CONTROL_FILES parameter are created.

If you do not include CONTROL_FILES in the initialization parameter file, then Oracle Database creates a control file in the same directory as the initialization parameter file, using a default operating system–dependent filename. If you have enabled Oracle Managed Files, the database creates Oracle managed control files.

If you want the database to create new operating system files when creating database control files, the filenames listed in the CONTROL_FILES parameter must not match any filenames that currently exist on your system. If you want the database to reuse or overwrite existing files when creating database control files, ensure that the filenames listed in the CONTROL_FILES parameter match the filenames that are to be reused, and include a CONTROLFILE REUSE clause in the CREATE DATABASE statement.

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

> **See Also:**
>
> - Chapter 11, "Managing Control Files"
> - "Specifying Oracle Managed Files at Database Creation" on page 2-19

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 Database can support up to four additional nonstandard 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 must specify. Typically, DB_BLOCK_SIZE is set to either 4K or 8K. If you do not set a value for this parameter, then the default data block size is operating system specific, which is generally adequate.

You cannot change the block size after database creation except by re-creating the database. If the database block size is different from the operating system block size,

then ensure that the database block size is a multiple of the operating system block size. For example, if your operating system block size is 2K (2048 bytes), the following setting for the DB_BLOCK_SIZE initialization parameter is valid:

```
DB_BLOCK_SIZE=4096
```

A larger data block size provides greater efficiency in disk and memory I/O (access and storage of data). Therefore, consider specifying a block size larger than your operating system block size if the following conditions exist:

- Oracle Database 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 Database 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, the database 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.

Nonstandard Block Sizes

Tablespaces of nonstandard block sizes can be created using the CREATE TABLESPACE statement and specifying the BLOCKSIZE clause. These nonstandard block sizes can have any of the following power-of-two values: 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 nonstandard block sizes, you must configure subcaches within the buffer cache area of the SGA memory for all of the nonstandard block sizes that you intend to use. The initialization parameters used for configuring these subcaches are described in "Using Automatic Shared Memory Management" on page 6-8.

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 data warehouse environment that uses a standard block size of 8K.

Note: A 32K block size is valid only on 64-bit platforms.

Caution: Oracle recommends against specifying a 2K block size when 4K sector size disks are in use, because performance degradation can occur. For an explanation, see "Planning the Block Size of Redo Log Files" on page 12-7.

See Also:

- "Creating Tablespaces" on page 14-2
- "Transporting Tablespaces Between Databases" on page 14-30

Specifying the Maximum Number of Processes

The PROCESSES initialization parameter determines the maximum number of operating system processes that can be connected to Oracle Database concurrently. The value of

this parameter must be a minimum of one for each background process plus one for each user process. The number of background processes will vary according the database features that you are using. For example, if you are using Advanced Queuing or the file mapping feature, then you will have additional background processes. If you are using Automatic Storage Management, then add three additional processes for the database instance.

If you plan on running 50 user processes, a good estimate would be to set the `PROCESSES` initialization parameter to 70.

Specifying the DDL Lock Timeout

Data definition language (DDL) statements require exclusive locks on internal structures. If these locks are unavailable when a DDL statement runs, then the DDL statement fails, though it might have succeeded if it had been executed subseconds later.

To enable DDL statements to wait for locks, specify a **DDL lock timeout**—the number of seconds a DDL command waits for its required locks before failing.

To specify a DDL lock timeout, use the `DDL_LOCK_TIMEOUT` parameter. The permissible range of values for `DDL_LOCK_TIMEOUT` is 0 to 1,000,000. The default is 0.

You can set `DDL_LOCK_TIMEOUT` at the system level, or at the session level with an `ALTER SESSION` statement.

Specifying the Method of Undo Space Management

Every Oracle Database must have a method of maintaining information that is used to undo changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Collectively these records are called **undo data**. This section provides instructions for setting up an environment for automatic undo management using an undo tablespace.

> **See Also:** Chapter 16, "Managing Undo"

UNDO_MANAGEMENT Initialization Parameter

The `UNDO_MANAGEMENT` initialization parameter determines whether an instance starts in automatic undo management mode, which stores undo in an undo tablespace. Set this parameter to `AUTO` to enable automatic undo management mode. Beginning with Release 11*g*, `AUTO` is the default if the parameter is omitted or is null.

UNDO_TABLESPACE Initialization Parameter

When an instance starts up in automatic undo management mode, it attempts to select an undo tablespace for storage of undo data. If the database was created in automatic undo management mode, then the default undo tablespace (either the system-created `SYS_UNDOTBS` tablespace or the user-specified undo tablespace) is the undo tablespace used at instance startup. You can override this default for the instance by specifying a value for the `UNDO_TABLESPACE` initialization parameter. This parameter is especially useful for assigning a particular undo tablespace to an instance in an Oracle Real Application Clusters environment.

If no undo tablespace is specified by the `UNDO_TABLESPACE` initialization parameter, then the first available undo tablespace in the database is chosen. If no undo tablespace is available, then the instance starts without an undo tablespace, and undo data is written to the `SYSTEM` tablespace. You should avoid running in this mode.

Note: When using the `CREATE DATABASE` statement to create a database, do not include an `UNDO_TABLESPACE` parameter in the initialization parameter file. Instead, include an `UNDO TABLESPACE` clause in the `CREATE DATABASE` statement.

About The COMPATIBLE Initialization Parameter

The `COMPATIBLE` initialization parameter enables or disables the use of features in the database that affect file format on disk. For example, if you create an Oracle Database 11*g* Release 2 (11.2) database, but specify `COMPATIBLE=10.0.0` in the initialization parameter file, then features that require 11.2 compatibility generate an error if you try to use them. Such a database is said to be at the 10.0.0 compatibility level.

You can advance the compatibility level of your database by changing the `COMPATIBLE` initialization parameter. If you do, there is no way to start the database using a lower compatibility level setting, except by doing a point-in-time recovery to a time before the compatibility was advanced.

The default value for the `COMPATIBLE` parameter is the release number of the most recent major release.

Note: For Oracle Database 11*g* Release 2 (11.2), the default value of the `COMPATIBLE` parameter is 11.2.0.0.0. The minimum value is 10.0.0. If you create an Oracle Database using the default value, you can immediately use all the new features in this release, and you can never downgrade the database.

See Also:

- *Oracle Database Upgrade Guide* for a detailed discussion of database compatibility and the `COMPATIBLE` initialization parameter
- *Oracle Database Backup and Recovery User's Guide* for information about point-in-time recovery of your database

Setting the License Parameter

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

If you use named user licensing, Oracle Database 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 initialization parameter file, as shown in the following example:

```
LICENSE_MAX_USERS = 200
```

Managing Initialization Parameters Using a Server Parameter File

Initialization parameters for the Oracle Database have traditionally been stored in a text initialization parameter file. For better manageability, you can choose to maintain initialization parameters in a binary server parameter file that is persistent across database startup and shutdown. 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
- Changing Initialization Parameter Values
- Clearing Initialization Parameter Values
- Exporting the Server Parameter File
- Backing Up the Server Parameter File
- Recovering a Lost or Damaged Server Parameter File
- Viewing Parameter Settings

What Is a Server Parameter File?

A server parameter file can be thought of as a repository for initialization parameters that is maintained on the system running the Oracle Database server. 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 arrangement eliminates the need to manually update initialization parameters to make persistent any changes effected by ALTER SYSTEM statements. It also provides a basis for self-tuning by the Oracle Database server.

A server parameter file is initially built from a text initialization parameter file using the CREATE SPFILE statement. (It can also be created directly by the Database Configuration Assistant.) The server parameter file is a binary file that cannot be edited using a text editor. Oracle Database provides other interfaces for viewing and modifying parameter settings in a server parameter file.

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 your instance, and if the instance is running, it could fail.

When you issue a STARTUP command with no PFILE clause, the Oracle instance searches an operating system–specific default location for a server parameter file from which to read initialization parameter settings. If no server parameter file is found, the

instance searches for a text initialization parameter file. If a server parameter file exists but you want to override it with settings in a text initialization parameter file, you must specify the `PFILE` clause when issuing the `STARTUP` command. Instructions for starting an instance using a server parameter file are contained in "Starting Up a Database" on page 3-1.

Migrating to a Server Parameter File

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

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

   Note: If you are migrating to a server parameter file in an Oracle Real Application Clusters environment, 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 instances that are part of an Oracle Real Application Clusters installation are discussed in *Oracle Real Application Clusters Administration and Deployment Guide* and in your platform-specific Oracle Real Application Clusters Installation Guide.

2. Create a server parameter file in the default location using the `CREATE SPFILE FROM PFILE` statement. See "Creating a Server Parameter File" on page 2-33 for instructions.

   This statement reads the text 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 or restart the instance.

   The instance finds the new SPFILE in the default location and starts up with it.

Creating a Server Parameter File

You use the `CREATE SPFILE` statement to create a server parameter file. You must have the `SYSDBA` or the `SYSOPER` system privilege to execute this statement.

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

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.

You can create a server parameter file (SPFILE) from an existing text initialization parameter file or from memory. Creating the SPFILE from memory means copying the current values of initialization parameters in the running instance to the SPFILE.

The following example creates a server parameter file from text initialization parameter file `/u01/oracle/dbs/init.ora`. In this example no `SPFILE` name is

specified, so the file is created with the platform-specific default name and location shown in Table 2–3 on page 2-34.

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

The next example illustrates creating a server parameter file and supplying a name and location.

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

The next example illustrates creating a server parameter file in the default location from the current values of the initialization parameters in memory.

```
CREATE SPFILE FROM MEMORY;
```

Whether you use the default SPFILE name and default location or specify an SPFILE name and location, if an SPFILE of the same name already exists in the location, it is overwritten without a warning message.

When you create an SPFILE from a text initialization parameter file, comments specified on the same lines as a parameter setting in the initialization parameter file are maintained in the SPFILE. All other comments are ignored.

Oracle recommends that you allow the database to give the SPFILE the default name and store it in the default location. This eases administration of your database. For example, the `STARTUP` command assumes this default location to read the SPFILE.

Table 2–3 shows the default name and location for both the text initialization parameter file (PFILE) and server parameter file (SPFILE) for the UNIX, Linux, and Windows platforms, both with and without the presence of Oracle Automatic Storage Management (Oracle ASM). The table assumes that the SPFILE is a file. If it is a raw device, the default name could be a logical volume name or partition device name, and the default location could differ.

Table 2–3 PFILE and SPFILE Default Names and Locations on UNIX, LInux, and Windows

Platform	PFILE Default Name	SPFILE Default Name	PFILE Default Location	SPFILE Default Location
UNIX and Linux	`init`*`ORACLE_SID`*`.ora`	`spfile`*`ORACLE_SID`*`.ora`	*OH*`/dbs` or the same location as the datafiles[1]	**Without Oracle ASM**: *OH*`/dbs` or the same location as the datafiles[1] **When Oracle ASM is present**: In the same disk group as the datafiles[2]
Windows	`init`*`ORACLE_SID`*`.ora`	`spfile`*`ORACLE_SID`*`.ora`	*OH*`\database`	**Without Oracle ASM**: *OH*`\database` **When Oracle ASM is present**: In the same disk group as the datafiles[2]

[1] *OH* represents the Oracle home directory

[2] Assumes database created with DBCA

> **Note:** Upon startup, the instance first searches for an SPFILE named `spfile`*`ORACLE_SID`*`.ora`, and if not found, searches for `spfile.ora`. Using `spfile.ora` enables all Real Application Cluster (Oracle RAC) instances to use the same server parameter file.
>
> If neither SPFILE is found, the instance searches for the text initialization parameter file `init`*`ORACLE_SID`*`.ora`.

If you create an SPFILE in a location other than the default location, you must create in the default PFILE location a "stub" PFILE that points to the server parameter file. For more information, see "Starting Up a Database" on page 3-1.

When you create the database with DBCA when Oracle ASM is present, DBCA places the SPFILE in an Oracle ASM disk group, and also causes this stub PFILE to be created.

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 database—that is, you issue a `STARTUP` command and do not specify a `PFILE` parameter—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.

Changing Initialization Parameter Values

The `ALTER SYSTEM` statement enables you to set, change, or restore to default the values of initialization parameters. If you are using a text initialization parameter file, the `ALTER SYSTEM` statement changes the value of a parameter only for the current instance, because there is no mechanism for automatically updating text initialization parameters on disk. You must update them manually to be passed to a future instance. Using a server parameter file overcomes this limitation.

There are two kinds of initialization parameters:

- **Dynamic initialization parameters** can be changed for the current Oracle Database instance. The changes take effect immediately.
- **Static initialization parameters** cannot be changed for the current instance. You must change these parameters in the text initialization file or server parameter file and then restart the database before changes take effect.

Setting or Changing Initialization Parameter Values

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

SCOPE Clause	Description
SCOPE = SPFILE	The change is applied in the server parameter file only. The effect is as follows: ■ No change is made to the current instance. ■ For both dynamic and static parameters, the change is effective at the next startup and is persistent. This is the only SCOPE specification allowed for static parameters.
SCOPE = MEMORY	The change is applied in memory only. The effect is as follows: ■ The change is made to the current instance and is effective immediately. ■ 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.
SCOPE = BOTH	The change is applied in both the server parameter file and memory. The effect is as follows: ■ The change is made to the current instance and is effective immediately. ■ For dynamic parameters, the effect is persistent because the server parameter file is updated. For static parameters, this specification is not allowed.

It is an error to specify SCOPE=SPFILE or SCOPE=BOTH if the instance did not start up with 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 text 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.

When you specify SCOPE as SPFILE or BOTH, an optional COMMENT clause lets you associate a text string with the parameter update. The comment is written to the server parameter file.

The following statement changes the maximum number of failed login attempts before the connection is dropped. It includes a comment, and explicitly states that the change is to be made only in the server parameter file.

```
ALTER SYSTEM SET SEC_MAX_FAILED_LOGIN_ATTEMPTS=3
                 COMMENT='Reduce from 10 for tighter security.'
                 SCOPE=SPFILE;
```

The next example sets a complex initialization parameter that takes a list of attributes. Specifically, the parameter value being set is the LOG_ARCHIVE_DEST_n initialization parameter. This statement could change an existing setting for this parameter or create a new archive destination.

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

When a value consists of a list of parameters, you cannot edit individual attributes 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.

Clearing Initialization Parameter Values

You can use the `ALTER SYSTEM RESET` command to clear (remove) the setting of any initialization parameter in the SPFILE that was used to start the instance. Neither `SCOPE=MEMORY` nor `SCOPE=BOTH` are allowed. The `SCOPE = SPFILE` clause is not required, but can be included.

You may want to clear a parameter in the SPFILE so that upon the next database startup a default value is used.

> **See Also:** *Oracle Database SQL Language Reference* for information about the `ALTER SYSTEM` command

Exporting the Server Parameter File

You can use the `CREATE PFILE` statement to export a server parameter file (SPFILE) to a text initialization parameter file. Doing so might be necessary for several reasons:

- 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.
- To modify the server parameter file by first exporting it, editing the resulting text file, and then re-creating it using the `CREATE SPFILE` statement

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

You must have the `SYSDBA` or the `SYSOPER` system privilege to execute the `CREATE PFILE` statement. The exported file is created on the database server system. 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 SPFILE:

```
CREATE PFILE FROM SPFILE;
```

Because no names were specified for the files, the database creates an initialization parameter file with a platform-specific name, 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, but in this example the names of the files are specified:

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

> **Note:** An alternative is to create a PFILE from the current values of the initialization parameters in memory. The following is an example of the required command:
>
> ```
> CREATE PFILE='/u01/oracle/dbs/test_init.ora' FROM MEMORY;
> ```

Backing Up the Server Parameter File

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

> **See Also:** *Oracle Database Backup and Recovery User's Guide*

Recovering a Lost or Damaged Server Parameter File

If your server parameter file (SPFILE) becomes lost or corrupted, the current instance may fail, or the next attempt at starting the database instance may fail. There are several ways to recover the SPFILE:

- If the instance is running, issue the following command to re-create the SPFILE from the current values of initialization parameters in memory:

  ```
  CREATE SPFILE FROM MEMORY;
  ```

  This command creates the SPFILE with the default name and in the default location. You can also create the SPFILE with a new name or in a specified location. See "Creating a Server Parameter File" on page 2-33 for examples.

- If you have a valid text initialization parameter file (PFILE), re-create the SPFILE from the PFILE with the following command:

  ```
  CREATE SPFILE FROM PFILE;
  ```

  This command assumes that the PFILE is in the default location and has the default name. See "Creating a Server Parameter File" on page 2-33 for the command syntax to use when the PFILE is not in the default location or has a nondefault name.

- Restore the SPFILE from backup.

  See "Backing Up the Server Parameter File" on page 2-37 for more information.

- If none of the previous methods are possible in your situation, perform these steps:

  1. Create a text initialization parameter file (PFILE) from the parameter value listings in the alert log.

     When an instance starts up, the initialization parameters used for startup are written to the alert log. You can copy and paste this section from the text version of the alert log (without XML tags) into a new PFILE.

     See "Viewing the Alert Log" on page 9-20 for more information.

  2. Create the SPFILE from the PFILE.

     See "Creating a Server Parameter File" on page 2-33 for instructions.

Read/Write Errors During a Parameter Update

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

- Shut down the instance, recover the server parameter file and described earlier in this section, and then restart the instance.
- Continue to run the database if you do not care that subsequent parameter updates will not be persistent.

Viewing Parameter Settings

You can view parameter settings in several ways, as shown in the following table.

Method	Description
SHOW PARAMETERS	This SQL*Plus command displays the values of initialization parameters in effect for the current session.
SHOW SPPARAMETERS	This SQL*Plus command displays the values of initialization parameters in the server parameter file (SPFILE).
CREATE PFILE	This SQL statement creates a text initialization parameter file (PFILE) from the SPFILE or from the current in-memory settings. You can then view the PFILE with any text editor.
V$PARAMETER	This view displays the values of initialization parameters in effect for the current session.
V$PARAMETER2	This view displays the values of initialization parameters in effect for the current session. It is easier to distinguish list parameter values in this view because each list parameter value appears in a separate row.
V$SYSTEM_PARAMETER	This view displays the values of initialization parameters in effect for the instance. A new session inherits parameter values from the instance-wide values.
V$SYSTEM_PARAMETER2	This view displays the values of initialization parameters in effect for the instance. A new session inherits parameter values from the instance-wide values. It is easier to distinguish list parameter values in this view because each list parameter value appears in a separate row.
V$SPPARAMETER	This view displays the current contents of the SPFILE. The view returns FALSE values in the ISSPECIFIED column if an SPFILE is not being used by the instance.

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

Managing Application Workloads with Database Services

This section contains:

- About Database Services
- Creating Database Services
- Database Service Data Dictionary Views

About Database Services

Database services (services) are logical abstractions for managing workloads in Oracle Database. Services divide workloads into mutually disjoint groupings. Each service represents a workload with common attributes, service-level thresholds, and priorities. The grouping is based on attributes of work that might include the application function to be used, the priority of execution for the application function, the job class to be managed, or the data range used in the application function or job class. For example, the Oracle E-Business suite defines a service for each responsibility, such as general ledger, accounts receivable, order entry, and so on. When you configure database services, you give each service a unique global name, associated performance goals, and associated importance. The services are tightly integrated with Oracle Database and are maintained in the data dictionary.

Connection requests can include a database service name. Thus, middle-tier applications and client/server applications use a service by specifying the service as

part of the connection in TNS connect data. If no service name is included and the Net Services file listener.ora designates a default service, the connection uses the default service.

Services enable you to configure a workload, administer it, enable and disable it, and measure the workload as a single entity. You can do this using standard tools such as the Database Configuration Assistant (DBCA), Net Configuration Assistant (NetCA), and Oracle Enterprise Manager. Enterprise Manager supports viewing and operating services as a whole, with drill down to the instance-level when needed.

In an Oracle Real Application Clusters (Oracle RAC) environment, a service can span one or more instances and facilitate workload balancing based on transaction performance. This provides end-to-end unattended recovery, rolling changes by workload, and full location transparency. Oracle RAC also enables you to manage several service features with Enterprise Manager, the DBCA, and the Server Control utility (`SRVCTL`).

Services also offer an extra dimension in performance tuning. Tuning by "service and SQL" can replace tuning by "session and SQL" in the majority of systems where all sessions are anonymous and shared. With services, workloads are visible and measurable. Resource consumption and waits are attributable by application. Additionally, resources assigned to services can be augmented when loads increase or decrease. This dynamic resource allocation enables a cost-effective solution for meeting demands as they occur. For example, services are measured automatically and the performance is compared to service-level thresholds. Performance violations are reported to Enterprise Manager, enabling the execution of automatic or scheduled solutions.

Several Oracle Database features support services. The Automatic Workload Repository (AWR) manages the performance of services. AWR records service performance, including execution times, wait classes, and resources consumed by service. AWR alerts warn when service response time thresholds are exceeded. The dynamic views report current service performance metrics with one hour of history. Each service has quality-of-service thresholds for response time and CPU consumption.

In addition, the Database Resource Manager can map services to consumer groups. Therefore, you can automatically manage the priority of one service relative to others. You can use consumer groups to define relative priority in terms of either ratios or resource consumption. For more information, see Chapter 27, "Managing Resources with Oracle Database Resource Manager," and specifically in "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8.

You also can specify an edition attribute for a service. Editions make it possible to have two or more versions of the same objects in the database. When you specify an edition attribute for a service, all subsequent connections that specify the service use this edition as the initial session edition. This is described in more detail in "Setting the Edition Attribute of a Database Service" on page 18-23.

Specifying an edition as a service attribute can make it easier to manage resource usage. For example, services associated with an edition can be placed on a separate instance in an Oracle RAC environment, and the Database Resource Manager can manage resources used by different editions by associating resource plans with the corresponding services.

Services describe applications, application functions, and data ranges as either functional services or data-dependent services. Functional services are the most common mapping of workloads. Sessions using a particular function are grouped together. In contrast, data-dependent routing routes sessions to services based on data

keys. The mapping of work requests to services occurs in the object relational mapping layer for application servers and TP monitors. For example, in Oracle RAC, these ranges can be completely dynamic and based on demand because the database is shared.

You can also define preconnect application services in Oracle RAC databases. Preconnect services span instances to support a service in the event of a failure. The preconnect service supports TAF preconnect mode and is managed transparently when using Oracle RAC.

In addition to services to be used by applications, Oracle Database also supports two internal services: `SYS$BACKGROUND` is used by the background processes only and `SYS$USERS` is the default service for user sessions that are not associated with services.

Using services requires no changes to your application code. Client-side work can connect to a named service. Server-side work, such as Oracle Scheduler, parallel execution, and Oracle Streams Advanced Queuing, set the service name as part of the workload definition. Work requests executing under a service inherit the performance thresholds for the service and are measured as part of the service.

For Oracle Scheduler, you optionally assign a service when you create a job class. During execution, jobs are assigned to job classes, and job classes can run within services. Using services with job classes ensures that the work executed by the job scheduler is identified for workload management and performance tuning.

For parallel query and parallel DML, the query coordinator connects to a service just like any other client. The parallel query processes inherit the service for the duration of the execution. At the end of query execution, the parallel execution processes revert to the default service.

See Also:

- Chapter 29, "Scheduling Jobs with Oracle Scheduler" for more information about the Oracle Scheduler
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about using services in an Oracle RAC environment
- *Oracle Database Net Services Administrator's Guide* for information on connecting to a service

Creating Database Services

There are a few ways to create database services, depending on your database configuration.

To create a database service:

- If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to create the database service.

  ```
  srvctl add service -d db_unique_name -s service_name
  ```

- If your single-instance database is not being managed by Oracle Restart, do one of the following:
  - Append the desired service name to the `SERVICE_NAMES` parameter.
  - Call the `DBMS_SERVICE.CREATE_SERVICE` package procedure.
- (Optional) Define service attributes with Oracle Enterprise Manager or with `DBMS_SERVICE.MODIFY_SERVICE`.

See Also:

- Chapter 4, "Configuring Automatic Restart of an Oracle Database" for information about Oracle Restart
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SERVICE` package
- *Oracle Real Application Clusters Administration and Deployment Guide* for information about creating a service in an Oracle RAC environment.

Database Service Data Dictionary Views

You can find service information in the following service-specific views:

- `DBA_SERVICES`
- `ALL_SERVICES` or `V$SERVICES`
- `V$ACTIVE_SERVICES`
- `V$SERVICE_STATS`
- `V$SERVICE_EVENT`
- `V$SERVICE_WAIT_CLASSES`
- `V$SERV_MOD_ACT_STATS`
- `V$SERVICE_METRICS`
- `V$SERVICE_METRICS_HISTORY`

The following additional views also contain some information about services:

- `V$SESSION`
- `V$ACTIVE_SESSION_HISTORY`
- `DBA_RSRC_GROUP_MAPPINGS`
- `DBA_SCHEDULER_JOB_CLASSES`
- `DBA_THRESHOLDS`

See Also: *Oracle Database Reference* for detailed information about these views

Considerations After Creating a Database

After you create a database as described in "Creating a Database with DBCA" on page 2-5 or "Creating a Database with the CREATE DATABASE Statement" on page 2-6, 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

In this release of Oracle Database, several enhancements were made to ensure the security your database. You can find security guidelines for this release in *Oracle Database Security Guide*. Oracle recommends that you read these guidelines and configure your database accordingly.

After the database is created, you can configure it to take advantage of Oracle Identity Management. For information on how to do this, see *Oracle Database Enterprise User Security Administrator's Guide*.

A newly created database has at least three user accounts that are important for administering your database: `SYS`, `SYSTEM`, and `SYSMAN`. Additional administrative accounts are provided that should be used only by authorized users. To protect these accounts from being used by unauthorized users familiar with their Oracle-supplied passwords, these accounts are initially locked with their passwords expired. As the database administrator, you are responsible for the unlocking and resetting of these accounts.

See *Oracle Database 2 Day + Security Guide* for a complete list of predefined user accounts created with each new Oracle Database installation.

Caution: To prevent unauthorized access and protect the integrity of your database, it is important that new passwords for user accounts `SYS` and `SYSTEM` be specified when the database is created. This is accomplished by specifying the following `CREATE DATABASE` clauses when manually creating you database, or by using DBCA to create the database:

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

See Also:

- "Administrative User Accounts" on page 1-14 for more information about the users `SYS` and `SYSTEM`
- *Oracle Database Security Guide* to learn how to add new users and change passwords
- *Oracle Database SQL Language Reference* for the syntax of the `ALTER USER` statement used for unlocking user accounts

Enabling Transparent Data Encryption

Transparent data encryption is a feature that enables encryption of individual database columns before storing them in the datafile, or enables encryption of entire tablespaces. If users attempt to circumvent the database access control mechanisms by looking inside datafiles directly with operating system tools, transparent data encryption prevents such users from viewing sensitive information.

Users who have the `CREATE TABLE` privilege can choose one or more columns in a table to be encrypted. The data is encrypted in the datafiles and in the audit logs (if audit is turned on). Database users with appropriate privileges can view the data in unencrypted format. For information on enabling transparent data encryption, see *Oracle Database Advanced Security Administrator's Guide*.

See Also:

- "Consider Encrypting Columns That Contain Sensitive Data" on page 20-8
- "Encrypted Tablespaces" on page 14-8

Creating a Secure External Password Store

For large-scale deployments where applications use password credentials to connect to databases, it is possible to store such credentials in a client-side Oracle wallet. An Oracle wallet is a secure software container that is used to store authentication and signing credentials.

Storing database password credentials in a client-side Oracle wallet eliminates the need to embed usernames and passwords in application code, batch jobs, or scripts. This reduces the risk of exposing passwords in the clear in scripts and application code, and simplifies maintenance because you need not change your code each time usernames and passwords change. In addition, not having to change application code also makes it easier to enforce password management policies for these user accounts.

When you configure a client to use the external password store, applications can use the following syntax to connect to databases that use password authentication:

```
CONNECT /@database_alias
```

Note that you need not specify database login credentials in this `CONNECT` statement. Instead your system looks for database login credentials in the client wallet.

> **See Also:** *Oracle Database Advanced Security Administrator's Guide* for information about configuring your client to use a secure external password store and for information about managing credentials in it

Installing the Oracle Database Sample Schemas

The Oracle Database distribution media includes various SQL files that let you experiment with the system, learn SQL, or create additional tables, views, or synonyms.

Oracle Database includes sample schemas that help you to become familiar with Oracle Database functionality. All Oracle Database documentation and training materials are being converted to the Sample Schemas environment as those materials are updated.

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

Dropping a Database

Dropping a database involves removing its datafiles, redo log files, control files, and initialization parameter files.

> **WARNING: Dropping a database deletes all data in the database.**

To drop a database:

- Submit the following statement:

  ```
  DROP DATABASE;
  ```

The `DROP DATABASE` statement first deletes all control files and all other database files listed in the control file. It then shuts down the database instance.

To use the `DROP DATABASE` statement successfully, the database must be mounted in exclusive and restricted mode.

The DROP DATABASE statement has no effect on archived log files, nor does it have any effect on copies or backups of the database. It is best to use RMAN to delete such files. If the database is on raw disks, the actual raw disk special files are not deleted.

If you used the Database Configuration Assistant to create your database, you can use that tool to delete (drop) your database and remove the files.

Database Data Dictionary Views

In addition to the views listed previously in "Viewing Parameter Settings", you can view information about your database content and structure using the following views:

View	Description
DATABASE_PROPERTIES	Displays permanent database properties
GLOBAL_NAME	Displays the global database name
V$DATABASE	Contains database information from the control file

3

Starting Up and Shutting Down

In this chapter:

- Starting Up a Database
- Altering Database Availability
- Shutting Down a Database
- Quiescing a Database
- Suspending and Resuming a Database

> **See Also:** *Oracle Real Application Clusters Administration and Deployment Guide* for additional information specific to an Oracle Real Application Clusters environment

Starting Up a Database

When you start up a database, you create an instance of that database and you determine the state of the database. Normally, you start up an instance by mounting and opening the database. Doing so makes the database available for any valid user to connect to and perform typical data access operations. Other options exist, and these are also discussed in this section.

This section contains the following topics relating to starting up an instance of a database:

- About Database Startup Options
- About Initialization Parameter Files and Startup
- About Automatic Startup of Database Services
- Starting Up an Instance
- Preparing to Start Up an Instance

About Database Startup Options

When Oracle Restart is not in use, you can start up a database instance with SQL*Plus, Recovery Manager, or Enterprise Manager. If your database is being managed by Oracle Restart, the recommended way to start the database is with SRVCTL.

See Chapter 4, "Configuring Automatic Restart of an Oracle Database" for information about Oracle Restart.

Starting Up a Database Using SQL*Plus

You can start a SQL*Plus session, connect to Oracle Database with administrator privileges, and then issue the `STARTUP` command. Using SQL*Plus in this way is the only method described in detail in this book.

Starting Up a Database Using Recovery Manager

You can also use Recovery Manager (RMAN) to execute `STARTUP` and `SHUTDOWN` commands. You may prefer to do this if your are within the RMAN environment and do not want to invoke SQL*Plus.

> **See Also:** *Oracle Database Backup and Recovery Reference* for information about the RMAN `STARTUP` command

Starting Up a Database Using Oracle Enterprise Manager

You can use Oracle Enterprise Manager to administer your database, including starting it up and shutting it down. Enterprise Manager combines a GUI console, agents, common services, and tools to provide an integrated and comprehensive systems management platform for managing Oracle products. Enterprise Manager Database Control, which is the portion of Enterprise Manager that is dedicated to administering an Oracle database, enables you to perform the functions discussed in this book using a GUI interface, rather than command line operations.

> **See Also:**
>
> - *Oracle Enterprise Manager Concepts*
> - *Oracle Database 2 Day DBA*

The remainder of this section describes using SQL*Plus to start up a database instance.

Starting Up a Database Using SRVCTL

When Oracle Restart is installed and configured for your database, Oracle recommends that you use SRVCTL to start the database. This ensures that:

- Any components on which the database depends (such as Oracle Automatic Storage Management and the Oracle Net listener) are automatically started first, and in the proper order.
- The database is started according to the settings in its Oracle Restart configuration. An example of such a setting is the server parameter file location.
- Environment variables stored in the Oracle Restart configuration for the database are set before starting the instance.

See "srvctl start database" on page 4-58 and "Starting and Stopping Components Managed by Oracle Restart" on page 4-25 for details.

Specifying Initialization Parameters at Startup

To start an instance, the database must read instance configuration parameters (the initialization parameters) from either a server parameter file (`SPFILE`) or a text initialization parameter file (PFILE).

The database looks for these files in a default location. You can specify nondefault locations for these files, and the method for doing so depends on whether you start the database with SQL*Plus (when Oracle Restart is not in use) or with SRVCTL (when the database is being managed with Oracle Restart).

The following sections provide details:

- About Initialization Parameter Files and Startup
- Starting Up with SQL*Plus with a Nondefault Server Parameter File
- Starting Up with SRVCTL with a Nondefault Server Parameter File

> **See Also:** Chapter 2, "Creating and Configuring an Oracle Database", for more information about initialization parameters, initialization parameter files, and server parameter files

About Initialization Parameter Files and Startup

When you start the database instance, it attempts to read the initialization parameters from an `SPFILE` in a platform-specific default location. If it finds no `SPFILE`, it searches for a text initialization parameter file.

Table 2–3 on page 2-34 lists PFILE and SPFILE default names and locations.

In the platform-specific default location, Oracle Database locates your initialization parameter file by examining file names in the following order:

1. `spfile`*`ORACLE_SID`*`.ora`
2. `spfile.ora`
3. `init`*`ORACLE_SID`*`.ora`

The first two files are `SPFILE`s and the third is a text initialization parameter file. If DBCA created the SPFILE in an Oracle Automatic Storage Management disk group, the database searches for the SPFILE in the disk group.

> **Note:** The `spfile.ora` file is included in this search path because in an Oracle 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 an Oracle Real Application Clusters environment, see *Oracle Real Application Clusters Administration and Deployment Guide*.

If you (or the Database Configuration Assistant) created a server parameter file, but you want to override it with a text initialization parameter file, you can do so with SQL*Plus, specifying the `PFILE` clause of the `STARTUP` command to identify the initialization parameter file:

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

Nondefault Server Parameter Files A nondefault server parameter file (`SPFILE`) is an `SPFILE` that is in a location other than the default location. It is not usually necessary to start an instance with a nondefault `SPFILE`. However, should such a need arise, both SRVCTL (with Oracle Restart) and SQL*Plus provide ways to do so. These are described later in this section.

Initialization Files and Oracle Automatic Storage Management A database that uses Oracle Automatic Storage Management (Oracle ASM) usually has a nondefault `SPFILE`. If you use the Database Configuration Assistant (DBCA) to configure a database to use Oracle ASM, DBCA creates an `SPFILE` for the database instance in an

Oracle ASM disk group, and then causes a text initialization parameter file (PFILE) to be created in the default location in the local file system to point to the SPFILE, as explained in the next section.

Starting Up with SQL*Plus with a Nondefault Server Parameter File

With SQL*Plus you can use the PFILE clause to start an instance with a nondefault server parameter file.

To start up with SQL*Plus with a nondefault server parameter file:

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
   ```

The SPFILE must reside on the database host computer. Therefore, the preceding method also provides a means for a client system to start a database that uses an SPFILE. It also eliminates the need for a client system to maintain a client-side initialization parameter file. When the client system reads the initialization parameter file containing the SPFILE parameter, it passes the value to the server where the specified SPFILE is read.

Starting Up with SRVCTL with a Nondefault Server Parameter File

If your database is being managed by Oracle Restart, you can specify the location of a nondefault SPFILE by setting or modifying the SPFILE location option in the Oracle Restart configuration for the database.

To start up with SRVCTL with a nondefault server parameter file:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl modify database -d db_unique_name -p spfile_path
   ```

   where *db_unique_name* must match the DB_UNIQUE_NAME initialization parameter setting for the database.

3. Enter the following command:

   ```
   srvctl start database -d db_unique_name [options]
   ```

See "SRVCTL Command Reference for Oracle Restart" on page 4-30 for more information.

About Automatic Startup of Database Services

When your database is managed by Oracle Restart, you can configure startup options for each individual database service (service). If you set the management policy for a service to `AUTOMATIC` (the default), the service starts automatically when you start the database with SRVCTL. If you set the management policy to `MANUAL`, the service does not automatically start, and you must manually start it with SRVCTL. A `MANUAL` setting does not prevent Oracle Restart from monitoring the service when it is running and restarting it if a failure occurs.

In an Oracle Data Guard (Data Guard) environment in which databases are managed by Oracle Restart, you can additionally control automatic startup of services by assigning Data Guard roles to the services in their Oracle Restart configurations. A service automatically starts upon manual database startup only if the management policy of the service is `AUTOMATIC` and if one of its assigned roles matches the current role of the database.

See "srvctl add service" on page 4-36 and "srvctl modify service" on page 4-51 for the syntax for setting the management policy of and Data Guard roles for a service.

Note: When using Oracle Restart, Oracle strongly recommends that you use SRVCTL to create database services.

Preparing to Start Up an Instance

Note: The following instructions are for installations where Oracle Restart is not in use. If your database is being managed by Oracle Restart, follow the instructions in "Starting and Stopping Components Managed by Oracle Restart" on page 4-25.

You must perform some preliminary steps before attempting to start an instance of your database using SQL*Plus.

1. Ensure that any Oracle components on which the database depends are started.

   For example, if the database stores data in Oracle Automatic Storage Management (Oracle ASM) disk groups, ensure that the Oracle ASM instance is running and the required disk groups are mounted. Also, it is preferable to start the Oracle Net listener before starting the database.

2. If you intend to use operating system authentication, log in to the database host computer as a member of the OSDBA group.

   See "Using Operating System Authentication" on page 1-20 for more information.

3. Ensure that environment variables are set so that you connect to the desired Oracle instance. For details, see "Submitting Commands and SQL to the Database" on page 1-6.

4. Start SQL*Plus without connecting to the database:

   ```
   SQLPLUS /NOLOG
   ```

5. Connect to Oracle Database as `SYSDBA`:

   ```
   CONNECT username AS SYSDBA
   ```

   —or—

```
CONNECT / AS SYSDBA
```

Now you are connected to the database 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.

Starting Up an Instance

When Oracle Restart is not in use, you use the SQL*Plus STARTUP command to start up an Oracle Database instance. If your database is being managed by Oracle Restart, Oracle recommends that you use the srvctl start database command.

In either case, you can start an instance in various modes:

- NOMOUNT—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.
- MOUNT—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.
- OPEN—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.
- FORCE—Force the instance to start after a startup or shutdown problem.
- OPEN RECOVER—Start the instance and have complete media recovery begin immediately.

> **Note:** You cannot start a database instance if you are connected to the database through a shared server process.

The following scenarios describe and illustrate the various states in which you can start up an instance. Some restrictions apply when combining clauses of the STARTUP command or combining startup options for the srvctl start database 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 Database 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, the database returns a warning message and does not open the database.

See Also:

- *SQL*Plus User's Guide and Reference* for details on the STARTUP command syntax
- "Starting and Stopping Components Managed by Oracle Restart" on page 4-25 for instructions for starting a database that is managed by Oracle Restart.

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 data access operations.

The following command starts an instance, reads the initialization parameters from the default location, and then mounts and opens the database.

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
STARTUP	srvctl start database -d *db_unique_name*

where *db_unique_name* matches the DB_UNIQUE_NAME initialization parameter.

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 one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
STARTUP NOMOUNT	srvctl start database -d *db_unique_name* -o 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:

- Enabling and disabling redo log archiving options. For more information, see Chapter 13, "Managing Archived Redo Logs".
- Performing full database recovery. For more information, see *Oracle Database Backup and Recovery User's Guide*

The following command starts an instance and mounts the database, but leaves the database closed:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
STARTUP MOUNT	srvctl start database -d *db_unique_name* -o mount

Restricting Access to an Instance at Startup

You can start an instance, and optionally mount and open a database, in restricted mode so that the instance is available only to administrative personnel (not general database users). Use this mode of instance startup when you must accomplish one of the following tasks:

- Perform an export or import of data

- Perform a data load (with SQL*Loader)
- Temporarily prevent typical users from using data
- Perform certain migration or 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. Further, when the instance is in restricted mode, a database administrator cannot access the instance remotely through an Oracle Net listener, but can only access the instance locally from the system that the instance is running on.

The following command starts an instance (and mounts and opens the database) in restricted mode:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP RESTRICT`	`srvctl start database -d` *`db_unique_name`* `-o restrict`

You can use the restrict mode in combination with the mount, nomount, and open modes.

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 that you must restrict access, you can use the `ALTER SYSTEM` statement to do so, as described in "Restricting Access to an Open Database" on page 3-11.

> **See Also:** *Oracle Database SQL Language 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 one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP FORCE`	`srvctl start database -d` *`db_unique_name`* `-o force`

If an instance is running, the force mode shuts it down with mode `ABORT` before restarting it. In this case, the alert log shows the message "`Shutting down instance (abort)`" followed by "`Starting ORACLE instance (normal)`."

> **See Also:** "Shutting Down with the Abort Mode" on page 3-13 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 one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`STARTUP OPEN RECOVER`	`srvctl start database -d` *`db_unique_name`* `-o "open,recover"`

If you attempt to perform recovery when no recovery is required, Oracle Database issues an error message.

Automatic Database Startup at Operating System Start

Many sites use procedures to enable automatic startup of one or more Oracle Database 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.

Beginning with Oracle Database 11*g* Release 2, the preferred (and platform-independent) method of configuring automatic startup of a database is Oracle Restart. See Chapter 4, "Configuring Automatic Restart of an Oracle Database" for details.

Starting Remote Instances

If your local Oracle Database 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 the availability of a database:

- 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 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` clause as follows:

```
ALTER DATABASE MOUNT;
```

> **See Also:** "Starting an Instance and Mounting a Database" on page 3-7 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` SQL statement with the `OPEN` clause:

```
ALTER DATABASE OPEN;
```

After executing this statement, any valid Oracle Database 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 affect 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 14-12.

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.

Limitations of a Read-only Database

- An application must not write database objects while executing against a read-only database. For example, an application writes database objects when it inserts, deletes, updates, or merges rows in a database table, including a global temporary table. An application writes database objects when it manipulates a database sequence. An application writes database objects when it locks rows, when it runs `EXPLAIN PLAN`, or when it executes DDL. Many of the functions and procedures in Oracle-supplied PL/SQL packages, such as `DBMS_SCHEDULER`, write database objects. If your application calls any of these functions and procedures, or if it performs any of the preceding operations, your application writes database objects and hence is not read-only.
- When executing on a read-only database, you must commit or roll back any in-progress transaction that involves one database link before you use another database link. This is true even if you execute a generic `SELECT` statement on the first database link and the *transaction* is currently read-only.
- You cannot compile or recompile PL/SQL stored procedures on a read-only database. To minimize PL/SQL invalidation because of remote procedure calls,

use REMOTE_DEPENDENCIES_MODE=SIGNATURE in any session that does remote procedure calls on a read-only database.

- You cannot invoke a remote procedure (even a read-only remote procedure) from a read-only database if the remote procedure has never been called on the database. This limitation applies to remote procedure calls in anonymous PL/SQL blocks and in SQL statements. You can either put the remote procedure call in a stored procedure, or you can invoke the remote procedure in the database prior to it becoming read only.

See Also: *Oracle Database SQL Language Reference* for more information about the ALTER DATABASE statement

Restricting Access to an Open Database

To place an already running instance in restricted mode, use the SQL statement ALTER SYSTEM with the ENABLE RESTRICTED SESSION clause. After this statement successfully completes, only users with the RESTRICTED SESSION privilege can initiate new connections. Users connecting as SYSDBA or connecting with the DBA role have this privilege.

Placing a running instance in restricted mode has the following affect on current sessions:

- In a single-instance environment without Oracle Restart, no user sessions are terminated or otherwise affected. Therefore, after placing an instance in restricted mode, consider killing (terminating) all current user sessions before performing administrative tasks.
- In a single-instance environment with Oracle Restart, any database services that are being managed by Oracle Restart go offline, and any sessions connected to those services are killed (terminated). The standard database service for the instance, named *DB_UNIQUE_NAME.DB_DOMAIN*, does not go offline because it is not managed by Oracle Restart.
- In an Oracle Real Application Clusters environment, any database services that are running on the instance and managed by Oracle Clusterware go offline for that instance, and any sessions connected to those services at that instance are killed. The standard database service for the instance (*DB_UNIQUE_NAME.DB_DOMAIN*) does not go offline.

To lift an instance from restricted mode, use ALTER SYSTEM with the DISABLE RESTRICTED SESSION clause.

See Also:

- "Terminating Sessions" on page 5-23 for directions for killing user sessions
- "Restricting Access to an Instance at Startup" on page 3-7 to learn some reasons for placing an instance in restricted mode

Shutting Down a Database

When Oracle Restart is not in use, you can shut down a database instance with SQL*Plus by connecting as SYSOPER or SYSDBA and issuing the SHUTDOWN command. If your database is being managed by Oracle Restart, the recommended way to shut down the database is with the srvctl stop database 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.

There are several modes for shutting down a database: normal, immediate, transactional, and abort. Some shutdown modes wait for certain events to occur (such as transactions completing or users disconnecting) before actually bringing down the database. There is a one-hour timeout period for these events.

Details are provided in the following sections:

- Shutting Down with the Normal Mode
- Shutting Down with the Immediate Mode
- Shutting Down with the Transactional Mode
- Shutting Down with the Abort Mode
- Shutdown Timeout

> **See Also:** Chapter 4, "Configuring Automatic Restart of an Oracle Database" for information about Oracle Restart.

Shutting Down with the Normal Mode

To shut down a database in normal situations, use one of these commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN [NORMAL]`	`srvctl stop database -d` *`db_unique_name`* `-o normal`

The `NORMAL` clause of the SQL*Plus `SHUTDOWN` command is optional because this is the default shutdown method. For SRVCTL, if the -o option is omitted, the shutdown operation proceeds according to the stop options stored in the Oracle Restart configuration for the database. The default stop option is `immediate`.

Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, the database 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 Mode

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 one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN IMMEDIATE`	`srvctl stop database -d` *`db_unique_name`* `-o 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 Database does not wait for users currently connected to the database to disconnect. The database 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 Mode

When you want to perform a planned shutdown of an instance while allowing active transactions to complete first, use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN TRANSACTIONAL`	`srvctl stop database -d` *`db_unique_name`* `-o 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 Mode

You can shut down a database instantaneously by aborting the database 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 must 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, use one of the following commands:

SQL*Plus	SRVCTL (When Oracle Restart Is In Use)
`SHUTDOWN ABORT`	`srvctl stop database -d` *`db_unique_name`* `-o 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 Database are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly disconnects all connected users.

The next startup of the database *will* require automatic instance recovery procedures.

Shutdown Timeout

Shutdown modes that wait for users to disconnect or for transactions to complete have a limit on the amount of time that they wait. If all events blocking the shutdown do not occur within one hour, the shutdown operation aborts with the following message: `ORA-01013: user requested cancel of current operation`. This message is also displayed if you interrupt the shutdown process, for example by pressing `CTRL-C`. Oracle recommends that you do not attempt to interrupt an instance shutdown. Instead, allow the shutdown process to complete, and then restart the instance.

After `ORA-01013` occurs, you must consider the instance to be in an unpredictable state. You must therefore continue the shutdown process by resubmitting a `SHUTDOWN` command. If subsequent `SHUTDOWN` commands continue to fail, you must submit a `SHUTDOWN ABORT` command to bring down the instance. You can then restart the instance.

Quiescing a Database

Occasionally you might want to put a database in a state that allows only DBA transactions, queries, fetches, or PL/SQL statements. Such a state is referred to as a **quiesced state**, in the sense that no ongoing non-DBA transactions, queries, fetches, or PL/SQL statements are running in the system.

> **Note:** In this discussion of quiesce database, 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.

The quiesced state lets administrators perform actions that cannot safely be done otherwise. These actions include:

- Actions that 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 when the table is first exported, then dropped, and finally imported. A concurrent user who

attempts to access the table after it was dropped, but before import, would not have an accurate view of the situation.

Without the ability to quiesce the database, you would need 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 a smaller restriction, because it eliminates the disruption to users and the downtime associated with shutting down and restarting the database.

When the database is in the quiesced state, it is through the facilities of the Database Resource Manager that non-DBA sessions are prevented from becoming active. Therefore, while this statement is in effect, any attempt to change the current resource plan will be queued until after the system is unquiesced. See Chapter 27, "Managing Resources with Oracle Database Resource Manager" for more information about the Database Resource Manager.

Placing a Database into a Quiesced State

To place a database into a quiesced state, issue the following SQL statement:

```
ALTER SYSTEM QUIESCE RESTRICTED;
```

Non-DBA active sessions will continue until they become inactive. An active session is one 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. For example, If a user 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 is processed.

Once all non-DBA sessions become inactive, the `ALTER SYSTEM QUIESCE RESTRICTED` statement completes, and the database is in a quiesced state. In an Oracle Real Application Clusters environment, this statement affects all instances, not just the one that issues the statement.

The `ALTER SYSTEM QUIESCE RESTRICTED` statement may wait a long time for active sessions to become inactive. You can determine the sessions that are blocking the quiesce operation by querying the `V$BLOCKING_QUIESCE` view. This view returns only a single column: `SID` (Session ID). You can join it with `V$SESSION` to get more information about the session, as shown in the following example:

```
select bl.sid, user, osuser, type, program
from v$blocking_quiesce bl, v$session se
where bl.sid = se.sid;
```

See *Oracle Database Reference* for details on these view.

If you interrupt the request to quiesce the database, or if your session terminates abnormally before all active sessions are quiesced, then Oracle Database automatically reverses any partial effects of the statement.

For queries that are 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.

> **Note:** You cannot perform a cold backup when the database is in the quiesced state, because Oracle Database background processes may still perform updates for internal purposes even while the database is quiesced. In addition, the file headers of online datafiles continue to appear to be accessible. They do not look the same as if a clean shutdown had been performed. However, you can still take online backups while the database is in a quiesced state.

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 quiesced the database. If the session issuing the `ALTER SYSTEM UNQUIESCE` statement terminates abnormally, then the Oracle Database server ensures that the unquiesce operation completes.

Viewing the Quiesce State of an Instance

You can query the `ACTIVE_STATE` column of the `V$INSTANCE` view to see the current state of an instance. The column values has one of these values:

- `NORMAL`: Normal unquiesced state.
- `QUIESCING`: Being quiesced, but some non-DBA sessions are still active.
- `QUIESCED`: Quiesced; no non-DBA sessions are active or allowed.

Suspending and Resuming a Database

The `ALTER SYSTEM SUSPEND` statement halts all input and output (I/O) to datafiles (file header and file data) and control files. The suspended state lets you back up a database 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 is not specific to an instance. In an Oracle Real Application Clusters environment, when you issue the suspend command on one system, internal locking mechanisms propagate the halt request across instances, thereby quiescing all active instances in a given cluster. However, if someone starts a new instance another instance is being suspended, the new instance will not be suspended.

Use the `ALTER SYSTEM RESUME` statement to resume normal database operations. The `SUSPEND` and `RESUME` commands can be issued 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` statement 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 cannot 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, because 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:** *Oracle Database Backup and Recovery User's Guide* for details about backing up a database using the database suspend/resume feature

4

Configuring Automatic Restart of an Oracle Database

Configure your Oracle database with the Oracle Restart feature to automatically restart the database, the listener, and other Oracle components after a hardware or software failure or whenever your database host computer restarts.

This chapter contains:

- About Oracle Restart
- Configuring Oracle Restart
- Starting and Stopping Components Managed by Oracle Restart
- Stopping and Restarting Oracle Restart for Maintenance Operations
- SRVCTL Command Reference for Oracle Restart
- CRSCTL Command Reference

About Oracle Restart

This section contains:

- Oracle Restart Overview
- About Startup Dependencies
- About Starting and Stopping Components with Oracle Restart
- About Starting and Stopping Oracle Restart
- Oracle Restart Configuration
- Oracle Restart Integration with Oracle Data Guard
- Fast Application Notification with Oracle Restart

Oracle Restart Overview

Oracle Restart improves the availability of your Oracle database. When you install Oracle Restart, various Oracle components can be automatically restarted after a hardware or software failure or whenever your database host computer restarts. Table 4–1 lists these components.

Table 4–1 Oracle Components Automatically Restarted by Oracle Restart

Component	Notes
Database instance	Oracle Restart can accommodate multiple databases on a single host computer.
Oracle Net listener	-
Database services	Does not include the default service created upon installation because it is automatically managed by Oracle Database, and does not include any default services created during database creation.
Oracle Automatic Storage Management (Oracle ASM) instance	-
Oracle ASM disk groups	Restarting a disk group means mounting it.
Oracle Notification Services (ONS)	In a standalone server environment, ONS can be used in Oracle Data Guard installations for automating failover of connections between primary and standby database through Fast Application Notification (FAN). ONS is a service for sending FAN events to integrated clients upon failover.

Oracle Restart runs periodic check operations to monitor the health of these components. If a check operation fails for a component, the component is shut down and restarted.

Oracle Restart is used in standalone server (non-clustered) environments only. For Oracle Real Application Clusters (Oracle RAC) environments, the functionality to automatically restart components is provided by Oracle Clusterware.

Oracle Restart runs out of the Oracle Grid Infrastructure home, which you install separately from Oracle Database homes. See the *Oracle Database Installation Guide* for your platform for information about installing the Oracle Grid Infrastructure home.

See Also:

- "Configuring Oracle Restart" on page 4-9
- *Oracle Automatic Storage Management Administrator's Guide* for information about Oracle Automatic Storage Management

About Startup Dependencies

Oracle Restart ensures that Oracle components are started in the proper order, in accordance with component dependencies. For example, if database files are stored in Oracle ASM disk groups, then before starting the database instance, Oracle Restart ensures that the Oracle ASM instance is started and the required disk groups are mounted. Likewise, if a component must be shut down, Oracle Restart ensures that dependent components are cleanly shut down first.

Oracle Restart also manages the weak dependency between database instances and the Oracle Net listener (the listener): When a database instance is started, Oracle Restart attempts to start the listener. If the listener startup fails, then the database is still started. If the listener later fails, Oracle Restart does not shut down and restart any database instances.

About Starting and Stopping Components with Oracle Restart

Oracle Restart automatically restarts various Oracle components when required, and automatically stops Oracle components in an orderly fashion when you manually shut down your system. There may be times, however, when you want to manually start or stop individual Oracle components. Oracle Restart includes the Server Control (SRVCTL) utility that you use to manually start and stop Oracle Restart–managed components. When Oracle Restart is in use, Oracle strongly recommends that you use SRVCTL to manually start and stop components.

After you stop a component with SRVCTL, Oracle Restart does not automatically restart that component if a failure occurs. If you then start the component with SRVCTL, that component is again available for automatic restart.

Oracle utilities such as SQL*Plus, the Listener Control utility (`LSNRCTL`), and `ASMCMD` are integrated with Oracle Restart. If you shut down the database with SQL*Plus, Oracle Restart does not interpret this as a database failure and does not attempt to restart the database. Similarly, if you shut down the Oracle ASM instance with SQL*Plus or `ASMCMD`, Oracle Restart does not attempt to restart it.

An important difference between starting a component with SRVCTL and starting it with SQL*Plus (or another utility) is the following:

- When you start a component with SRVCTL, any components on which this component depends are automatically started first, and in the proper order.
- When you start a component with SQL*Plus (or another utility), other components in the dependency chain are not automatically started; you must ensure that any components on which this component depends are started.

In addition, Oracle Restart enables you to start and stop all of the components managed by Oracle Restart in a specified Oracle home using a single command. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home. This capability is useful when you are installing a patch.

> **See Also:** "Starting and Stopping Components Managed by Oracle Restart" on page 4-25

About Starting and Stopping Oracle Restart

The CRSCTL utility starts and stops Oracle Restart. You can also use the CRSCTL utility to enable or disable Oracle high availability services. Oracle Restart uses Oracle high availability services to start and stop automatically the components managed by Oracle Restart. For example, Oracle high availability services daemons automatically start databases, listeners, and Oracle ASM instances. When Oracle high availability services are disabled, none of the components managed by Oracle Restart are started when a node is rebooted.

Typically, you use the CRSCTL utility when you must stop all of the running Oracle software in an Oracle installation. For example, you might need to stop Oracle Restart when you are installing a patch or performing operating system maintenance. When the maintenance is complete, you use the CRSCTL utility to start Oracle Restart.

> **See Also:** "Stopping and Restarting Oracle Restart for Maintenance Operations" on page 4-27 for information about using the CRSCTL utility

Oracle Restart Configuration

Oracle Restart maintains a list of all the Oracle components that it manages, and maintains configuration information for each component. All of this information is collectively known as the **Oracle Restart configuration**. When Oracle Restart starts a component, it starts the component according to the configuration information for that component. For example, the Oracle Restart configuration includes the location of the server parameter file (SPFILE) for databases, and the TCP port to listen on for listeners.

If you install Oracle Restart and then create your database with Database Configuration Assistant (DBCA), DBCA automatically adds the database to the Oracle Restart configuration. When DBCA then starts the database, the required dependencies between the database and other components (for example disk groups in which the database stores data) are established, and Oracle Restart begins to manage the database.

You can manually add and remove components from the Oracle Restart configuration with SRVCTL commands. For example, if you install Oracle Restart onto a host on which a database is already running, you can use SRVCTL to add that database to the Oracle Restart configuration. When you manually add a component to the Oracle Restart configuration and then start it with SRVCTL, Oracle Restart begins to manage the component, restarting it when required.

> **Note:** Adding a component to the Oracle Restart configuration is also referred to as "registering a component with Oracle Restart."

Other SRVCTL commands enable you to view the status and configuration of Oracle Restart–managed components, temporarily disable and then reenable management for components, and more.

When Oracle Restart is installed, many operations that create Oracle components automatically add the components to the Oracle Restart configuration. Table 4–2 lists some create operations and whether the created component is automatically added.

Table 4–2 Create Operations and the Oracle Restart Configuration

Create Operation	Created Component Automatically Added to Oracle Restart Configuration?
Create a database with OUI or DBCA	Yes
Create a database with the `CREATE DATABASE` SQL statement	No
Create an Oracle ASM instance with OUI, DBCA, or ASMCA	Yes
Create a disk group (any method)	Yes
Add a listener with NETCA	Yes
Create a database service with SRVCTL	Yes
Create a database service by modifying the `SERVICE_NAMES` initialization parameter[1]	No
Create a database service with `DBMS_SERVICE.CREATE_SERVICE`	No
Create a standby database	No

[1] Not recommended when Oracle Restart is in use

Table 4–3 lists some delete/drop/remove operations and whether the deleted component is also automatically removed from the Oracle Restart configuration.

Table 4–3 Delete/Drop/Remove Operations and the Oracle Restart Configuration

Operation	Deleted Component Automatically Removed from Oracle Restart Configuration?
Delete a database with DBCA	Yes
Delete a database by removing database files with operating system commands[1]	No
Delete a listener with NETCA	Yes
Drop an Oracle ASM disk group (any method)	Yes
Delete a database service with SRVCTL	Yes
Delete a database service by any other means	No

[1] Not recommended

Oracle Restart Integration with Oracle Data Guard

Oracle Restart is integrated with Oracle Data Guard (Data Guard) and the Oracle Data Guard Broker (the broker). When a database shutdown and restart is required in response to a role change request, Oracle Restart shuts down and restarts the database in an orderly fashion (taking dependencies into account), and according to the settings in the Oracle Restart configuration. Oracle Restart also ensures that, following a Data Guard role transition, all database services configured to run in the new database role are active and all services not configured to run in the new role are stopped.

In addition, the Oracle Restart configuration supports Data Guard–related configuration options for the following components:

- **Databases**—When you add a database to the Oracle Restart configuration, you can specify the current Data Guard role for the database: `PRIMARY`, `PHYSICAL_STANDBY`, `LOGICAL_STANDBY`, or `SNAPSHOT_STANDBY`. If the role is later changed using the broker, Oracle Restart automatically updates the database configuration with the new role. If you change the database role without using the broker, you must manually modify the database's role in the Oracle Restart configuration to reflect the new role.
- **Database Services**—When adding a database service to the Oracle Restart configuration, you can specify one or more Data Guard roles for the service. When this configuration option is present, upon database restart Oracle Restart starts the service only if one of the service roles matches the current database role.

See Also:

- *Oracle Data Guard Concepts and Administration* for information about Oracle Data Guard
- "Fast Application Notification with Oracle Restart" on page 4-6
- "Automating the Failover of Connections Between Primary and Standby Databases" on page 4-20

Fast Application Notification with Oracle Restart

In a standalone server environment, Oracle Restart uses Oracle Notification Services (ONS) and Oracle Advanced Queues to publish Fast Application Notification (FAN) high availability events. Integrated Oracle clients use FAN to provide fast notification to clients when the service or instance goes down. The client can automate the failover of database connections between a primary database and a standby database.

This section describes how ONS and FAN work with Oracle Restart. It contains the following topics:

- Overview of Fast Application Notification
- Application High Availability with Services and FAN
- Managing Unplanned Outages
- Managing Planned Outages
- Fast Application Notification High Availability Events
- Using Fast Application Notification Callouts
- Oracle Clients That Are Integrated with Fast Application Notification

See Also: *Oracle Database Advanced Queuing User's Guide*

Overview of Fast Application Notification

FAN is a notification mechanism that Oracle Restart can use to notify other processes about configuration changes that include service status changes, such as `UP` or `DOWN` events. FAN provides the ability to immediately terminate inflight transaction when an instance or server fails. Integrated Oracle clients receive the events and respond. Applications can respond either by propagating the error to the user or by resubmitting the transactions and masking the error from the application user. When a `DOWN` event occurs, integrated clients immediately clean up connections to the terminated database. When an `UP` event occurs, the clients create new connections to the new primary database instance.

Oracle Restart publishes FAN events whenever a managed instance or service goes up or down. After a failover, the Oracle Data Guard Broker (broker) publishes FAN events. These FAN events can be used in the following ways:

- Applications can use FAN with Oracle Restart without programmatic changes if they use one of these Oracle integrated database clients: Oracle Database JDBC, Universal Connection Pool for Java, Oracle Call Interface, and Oracle Database ODP.NET. These clients can be configured for Fast Connection Failover (FCF) to automatically connect to a new primary database after a failover.
- FAN server-side callouts can be configured on the database tier.

For `DOWN` events, such as a failed primary database, FAN provides immediate notification to the clients so that they can failover as fast as possible to the new primary database. The clients do not wait for a timeout. The clients are notified immediately, and they must be configured to failover when they are notified.

For `UP` events, when services and instances are started, new connections can be created so that the application can immediately take advantage of the extra resources.

Through server-side callouts, you can also use FAN to:

- Log status information
- Page DBAs or open support tickets when resources fail to start

- Automatically start dependent external applications that must be co-located with a service

FAN events are published using ONS and Oracle Streams Advanced Queuing queues. The queues are configured automatically when you configure a service. You must configure ONS manually using SRVCTL commands.

The Connection Manager (CMAN) and Oracle Net Services listeners are integrated with FAN events, enabling the CMAN and the listener to immediately de-register services provided by the failed instance and to avoid erroneously sending connection requests to a failed database.

See Also:

- *Oracle Data Guard Broker* for information about FAN events in an Oracle Data Guard environment
- The Maximum Availability Architecture (MAA) white paper about client failover:

  `http://www.oracle.com/technetwork/database/features/availability/maa-090890.html`

Application High Availability with Services and FAN

Oracle Database focuses on maintaining service availability. With Oracle Restart, Oracle services are designed to be continuously available. Oracle Restart monitors the database and its services and, when configured, sends event notifications using FAN.

Managing Unplanned Outages If Oracle Restart detects an outage, then it isolates the failed component and recovers the dependent components. If the failed component is the database instance, then after Oracle Data Guard fails over to the standby database, Oracle Restart on the new primary database starts any services defined with the current role.

FAN events are published by Oracle Restart and the Oracle Data Guard Broker through ONS and Advanced Queuing. You can also perform notifications using FAN callouts.

Note: Oracle Restart does not run callouts with guaranteed ordering. Callouts are run asynchronously, and they are subject to scheduling variability.

With Oracle Restart, restart and recovery are automatic, including the restarting of the subsystems, such as the listener and the Oracle Automatic Storage Management (Oracle ASM) processes, not just the database. You can use FAN callouts to report faults to your fault management system and to initiate repair jobs.

Managing Planned Outages For repairs, upgrades, and changes that require you to shut down the primary database, Oracle Restart provides interfaces that disable and enable services to minimize service disruption to application users. Using Oracle Data Guard Broker with Oracle Restart allows a coordinated failover of the database service from the primary to the standby for the duration of the planned outage. Once you complete the operation, you can return the service to normal operation.

The management policy for a service controls whether the service starts automatically when the database is restarted. If the management policy for a service is set to

AUTOMATIC, then it restarts automatically. If the management policy for a service is set to MANUAL, then it must be started manually.

> **See Also:** "Modifying the Oracle Restart Configuration for a Component" on page 4-16

Fast Application Notification High Availability Events Table 4–4 describes the FAN event record parameters and the event types, followed by name-value pairs for the event properties. The event type is always the first entry and the timestamp is always the last entry. In the following example, the name in the name-value pair is shown in Fan event type (service_member), and the value in the name-value pair is shown in Properties:

```
FAN event type: service_member
Properties: version=1.0 service=ERP database=FINPROD instance=FINPROD host=node1
status=up
```

Table 4–4 Event Record Parameters and Descriptions

Parameter	Description
VERSION	Version of the event record. Used to identify release changes.
EVENT TYPE	SERVICE, SERVICE_MEMBER, DATABASE, INSTANCE, NODE, ASM, SRV_PRECONNECT. Note that database and Instance types provide the database service, such as DB_UNIQUE_NAME.DB_DOMAIN.
DATABASE UNIQUE NAME	The unique database supporting the service; matches the initialization parameter value for DB_UNIQUE_NAME, which defaults to the value of the initialization parameter DB_NAME.
INSTANCE	The name of the instance that supports the service; matches the ORACLE_SID value.
NODE NAME	The name of the node that supports the service or the node that has stopped; matches the node name known to Cluster Synchronization Services (CSS).
SERVICE	The service name; matches the service in DBA_SERVICES.
STATUS	Values are UP, DOWN, NOT_RESTARTING, PRECONN_UP, PRECONN_DOWN, and UNKNOWN.
REASON	Data_Guard_Failover, Failure, Dependency, User, Autostart, Restart.
CARDINALITY	The number of service members that are currently active; included in all UP events.
TIMESTAMP	The local time zone to use when ordering notification events.

A FAN record matches the database signature of each session as shown in Table 4–5.

Table 4–5 FAN Parameters and Matching Database Signatures

FAN Parameter	Matching Oracle Database Signature
SERVICE	sys_context('userenv', 'service_name')
DATABASE UNIQUE NAME	sys_context('userenv', 'db_unique_name')
INSTANCE	sys_context('userenv', 'instance_name')
NODE NAME	sys_context('userenv', 'server_host')

Using Fast Application Notification Callouts FAN callouts are server-side executables that Oracle Restart executes immediately when high availability events occur. You can use FAN callouts to automate the following activities when events occur, such as:

- Opening fault tracking tickets
- Sending messages to pagers
- Sending e-mail
- Starting and stopping server-side applications
- Maintaining an uptime log by logging each event as it occurs

To use FAN callouts, place an executable in the directory grid_home/racg/usrco on both the primary and the standby database servers. If you are using scripts, then set the shell as the first line of the executable. The following is an example file for the grid_home/racg/usrco/callout.sh callout:

```
#! /bin/ksh
FAN_LOGFILE= [your path name]/admin/log/`hostname`_uptime.log
echo $* "reported="`date` >> $FAN_LOGFILE &
```

The following output is from the previous example:

```
NODE VERSION=1.0 host=sun880-2 status=nodedown reason=
timestamp=08-Oct-2004 04:02:14 reported=Fri Oct 8 04:02:14 PDT 2004
```

A FAN record matches the database signature of each session, as shown in Table 4–5. Use this information to take actions on sessions that match the FAN event data.

> **See Also:** Table 4–4 on page 4-8 for information about the callout and event details

Oracle Clients That Are Integrated with Fast Application Notification Oracle has integrated FAN with many of the common Oracle client drivers that are used to connect to Oracle Restart databases. Therefore, the easiest way to use FAN is to use an integrated Oracle Client.

You can use the CMAN session pools, Oracle Call Interface, Universal Connection Pool for Java, JDBC simplefan API, and ODP.NET connection pools. The overall goal is to enable applications to consistently obtain connections to the available primary database at anytime.

> **See Also:** "Automating the Failover of Connections Between Primary and Standby Databases" on page 4-20

Configuring Oracle Restart

If you install Oracle Restart by installing the Oracle Grid Infrastructure for a standalone server and then create your database, the database is automatically added to the Oracle Restart configuration, and is then automatically restarted when required. However, if you install Oracle Restart on a host computer on which a database already exists, you must manually add the database, the listener, the Oracle Automatic Storage Management (Oracle ASM) instance, and possibly other components to the Oracle Restart configuration.

After configuring Oracle Restart to manage your database, you may want to:

- Add additional components to the Oracle Restart configuration.
- Remove components from the Oracle Restart configuration.

- Temporarily suspend Oracle Restart management for one or more components.
- Modify the Oracle Restart configuration options for an individual component.

This section describes the SRVCTL commands that you use to accomplish these and other tasks. It contains the following topics:

- Preparing to Run SRVCTL
- Obtaining Help for SRVCTL
- Adding Components to the Oracle Restart Configuration
- Removing Components from the Oracle Restart Configuration
- Disabling and Enabling Oracle Restart Management for a Component
- Viewing Component Status
- Viewing the Oracle Restart Configuration for a Component
- Modifying the Oracle Restart Configuration for a Component
- Managing Environment Variables in the Oracle Restart Configuration
- Creating and Deleting Database Services with SRVCTL
- Enabling FAN Events in an Oracle Restart Environment
- Automating the Failover of Connections Between Primary and Standby Databases
- Enabling Clients for Fast Connection Failover

See Also: "About Oracle Restart" on page 4-1

Preparing to Run SRVCTL

The tasks in the following sections require that you run the SRVCTL utility. You must ensure that you run SRVCTL from the correct Oracle home, and that you log in to the host computer with the correct user account. Table 4–6 lists the components that you can configure with SRVCTL, and for each component, lists the Oracle home from which you must run SRVCTL.

Table 4–6 Determining the Oracle Home from which to Start SRVCTL

Component Being Configured	Oracle Home from which to Start SRVCTL
Database, database service	Database home
Oracle ASM instance, disk group, listener[1], ONS	Oracle Grid Infrastructure home

[1] Assumes the listener was started from the Oracle Grid Infrastructure home. If you installed Oracle Restart for an existing database, the listener may have been started from the database home, in which case you start SRVCTL from the database home.

To prepare to run SRVCTL:

1. Use Table 4–6 to determine the Oracle home from which you must run SRVCTL.
2. If you intend to run a SRVCTL command that modifies the Oracle Restart configuration (`add`, `remove`, `enable`, `disable`, and so on), then do one of the following:
   - On UNIX and Linux, log in to the database host computer as the user who installed the Oracle home that you determined in Step 1.
   - On Windows, log in as an Administrator.

Otherwise, log in to the host computer as any user.

3. Open the command window that you will use to enter the SRVCTL commands.

   To enter commands, you might need to ensure that the SRVCTL program is in your `PATH` environment variable. Otherwise, you can enter the absolute path to the program.

Obtaining Help for SRVCTL

Online help is available for the SRVCTL utility.

To obtain help for SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl
   ```

For more detailed help, enter the following command:

```
srvctl -h
```

For detailed help on a particular command, enter:

srvctl *command* -h

For example, to obtain help for the `add` command and the different options for each component type, enter:

```
srvctl add -h
```

For detailed help on a particular command for a particular component type, enter:

srvctl *command object* -h

For example, to obtain help about adding a database service, enter the following command:

```
srvctl add service -h
```

See Table 4–7 on page 4-30 for a list of SRVCTL commands and Table 4–8 on page 4-31 for a list of components.

Adding Components to the Oracle Restart Configuration

In most cases, creating an Oracle component on a host that is running Oracle Restart automatically adds the component to the Oracle Restart configuration. (See Table 4–2 on page 4-4.) The component is then automatically restarted when required.

The following are occasions when you must manually add components to the Oracle Restart configuration with SRVCTL:

- You install Oracle Restart after creating the database.
- You create an additional Oracle database on the same host computer using the `CREATE DATABASE` SQL statement.
- You create a database service with `DBMS_SERVICE.CREATE_SERVICE` package procedure. (The recommended way is to use SRVCTL.)

> **Note:** Adding a component to the Oracle Restart configuration is also referred to as "registering a component with Oracle Restart."

Adding a component to the Oracle Restart configuration does not start that component. You must use a `srvctl start` command to start it.

You can also use Oracle Enterprise Manager Database Control (Database Control) to add a database or listener to the Oracle Restart configuration. Both the SRVCTL and Database Control methods are described in the following sections:

- Adding Components with SRVCTL
- Adding Components with Oracle Enterprise Manager Database Control

> **Important:** When you manually add a database to the Oracle Restart configuration, you must also add the Oracle grid infrastructure software owner as a member of the OSDBA group of that database. This is because the grid infrastructure components must be able to connect to the database as `SYSDBA` to start and stop the database.
>
> For example, if the host user who installed the grid infrastructure home is named `grid` and the OSDBA group of the new database is named `dba`, then user `grid` must be a member of the `dba` group.

See Also:

- "Starting and Stopping Components Managed by Oracle Restart" on page 4-25
- "OSDBA and OSOPER" on page 1-20

Adding Components with SRVCTL

When you add a component to the Oracle Restart configuration with SRVCTL, you can specify optional configuration settings for the component.

To add a component to the Oracle Restart configuration with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl add object options
   ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL add command on page 4-32 for available options for each component.

Example 4–1 Adding a Database

This example adds a database with a `DB_UNIQUE_NAME` of `dbcrm`. The mandatory `-o` option specifies the Oracle home location.

```
srvctl add database -d dbcrm -o /u01/app/oracle/product/11.2.0/dbhome_1
```

Example 4–2 Adding a Database Service

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example both creates a new database service named `crmbatch` and adds it to the Oracle Restart configuration.

```
srvctl add service -d dbcrm -s crmbatch
```

See "Creating and Deleting Database Services with SRVCTL" on page 4-19 for more examples.

Example 4–3 Adding the Default Listener

This example adds the default listener to the Oracle Restart configuration.

```
srvctl add listener
```

> **See Also:** "SRVCTL Command Reference for Oracle Restart" on page 4-30

Adding Components with Oracle Enterprise Manager Database Control

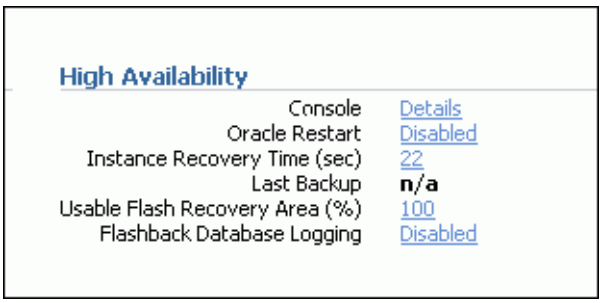
With Oracle Enterprise Manager Database Control (Database Control), you can add only database instances and listeners to the Oracle Restart configuration.

To add a database instance with Database Control:

1. Access the Database Home page for the desired database instance.

   See "Accessing the Database Home Page" in *Oracle Database 2 Day DBA* for instructions.

2. In the High Availability section, next to the Oracle Restart label, click the **Disabled** link.



> **Note:** If the Oracle Restart label shows "Enabled," then the database is already being managed by Oracle Restart, and there is no need to continue.

3. If prompted for host credentials, enter credentials for the user who installed the database Oracle home, and then click **Login**.

4. On the confirmation page, click **Continue**.

To add a listener with Database Control:

1. Access the Database Home page for the desired database instance.

   See "Accessing the Database Home Page" in *Oracle Database 2 Day DBA* for instructions.

2. In the General section, click the link next to the Listener label.

3. in the High Availability section, next to the Oracle Restart label, click the **Disabled** link.

> **Note:** If the Oracle Restart label shows "Enabled," then the listener is already being managed by Oracle Restart and there is no need to continue.

4. On the confirmation page, click **Continue**.

Removing Components from the Oracle Restart Configuration

When you use an Oracle-recommended method to delete an Oracle component, the component is also automatically removed from the Oracle Restart configuration. For example, if you use Database Configuration Assistant (DBCA) to delete a database, DBCA removes the database from the Oracle Restart configuration. Likewise, if you use Oracle Net Configuration Assistant (NETCA) to delete a listener, NETCA removes the listener from the Oracle Restart configuration. See Table 4–3 on page 4-5 for more examples. If you use a non-recommended or manual method to delete an Oracle component, you must first use SRVCTL to remove the component from the Oracle Restart configuration. Failing to do so could result in an error.

To remove a component from the Oracle Restart configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl remove object [options]
   ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL remove command on page 4-53 for available options for each component.

Example 4–4 Removing a Database

This example removes a database with a `DB_UNIQUE_NAME` of `dbcrm`.

```
srvctl remove database -d dbcrm
```

> **See Also:** "SRVCTL Command Reference for Oracle Restart" on page 4-30

Disabling and Enabling Oracle Restart Management for a Component

You can temporarily disable Oracle Restart management for a component. One reason to do this is when you are performing maintenance on the component. For example, if a component must be repaired, then you might not want it to be automatically restarted if it fails or if the host computer is restarted.

When maintenance is complete, you can reenable management for the component.

When you disable a component:

- It is no longer automatically restarted.
- It is no longer automatically started through a dependency.
- It cannot be started with SRVCTL.
- Any component dependent on this resource is no longer automatically started or restarted.

To disable or enable automatic restart for a component:

1. Prepare to run SRVCTL, as described in "Preparing to Run SRVCTL" on page 4-10.

2. Do one of the following:
   - To disable a component, enter the following command:

     ```
     srvctl disable object [options]
     ```

   - To enable a component, enter the following command:

     ```
     srvctl enable object [options]
     ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL disable command on page 4-41 and the enable command on page 4-44 for available options for each component.

Example 4–5 Disabling Automatic Restart for a Database

This example disables automatic restart for a database with a DB_UNIQUE_NAME of dbcrm.

```
srvctl disable database -d dbcrm
```

Example 4–6 Disabling Automatic Restart for an Oracle ASM Disk Group

This example disables automatic restart for the Oracle ASM disk group named recovery.

```
srvctl disable diskgroup -g recovery
```

Example 4–7 Enabling Automatic Restart for an Oracle ASM Disk Group

This example reenables automatic restart for the disk group recovery.

```
srvctl enable diskgroup -g recovery
```

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Viewing Component Status

You can use SRVCTL to view the running status (running or not running) for any component managed by Oracle Restart. For some components, additional information is also displayed.

To view component status:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl status object [options]
   ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL status command on page 4-62 for available options for each component.

Example 4–8 Viewing Status of a Database

This example displays the status of the database with a DB_UNIQUE_NAME of dbcrm.

```
srvctl status database -d dbcrm

Database is running.
```

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Viewing the Oracle Restart Configuration for a Component

You can use SRVCTL to view the Oracle Restart configuration for any component. Oracle Restart maintains different configuration information for each component type. In one form of the SRVCTL command, you can obtain a list of components managed by Oracle Restart.

To view component configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl config object options
   ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL config command on page 4-38 for available options for each component.

Example 4–9 Viewing a List of All Databases Managed by Oracle Restart

```
srvctl config database

dbcrm
orcl
```

Example 4–10 Viewing the Configuration of a Particular Database

This example displays the configuration of the database with a `DB_UNIQUE_NAME` of `orcl`.

```
srvctl config database -d orcl

Database unique name: orcl
Database name: orcl
Oracle home: /u01/app/oracle/product/11.2.0/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl.ora
Domain: us.example.com
Start options: open
Stop options: immediate
Database role:
Management policy: automatic
Disk Groups: DATA
Services: mfg,sales
```

> **See Also:** "SRVCTL Command Reference for Oracle Restart" on page 4-30

Modifying the Oracle Restart Configuration for a Component

You can use SRVCTL to modify the Oracle Restart configuration of a component. For example, you can modify the port number that a listener listens on when Oracle Restart starts it, or the server parameter file (SPFILE) that Oracle Restart points to when it starts a database.

To modify the Oracle Restart configuration for a component:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl modify object options
   ```

where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL modify command on page 4-49 for available options for each component.

Example 4–11 Modifying the Oracle Restart Configuration for a Database

For the database with a DB_UNIQUE_NAME of dbcrm, the following command changes the management policy to MANUAL and the start option to NOMOUNT.

```
srvctl modify database -d dbcrm -y MANUAL -s NOMOUNT
```

With a MANUAL management policy, the database is never automatically started when the database host computer is restarted. However, Oracle Restart continues to monitor the database and restarts it if a failure occurs.

See Also:

- "Viewing the Oracle Restart Configuration for a Component" on page 4-16
- "SRVCTL Command Reference for Oracle Restart" on page 4-30

Managing Environment Variables in the Oracle Restart Configuration

The Oracle Restart configuration can store name/value pairs for environment variables. If you typically set environment variables (other than ORACLE_HOME and ORACLE_SID) prior to starting your Oracle database, you can set these environment variable values in the Oracle Restart configuration. You can store any number environment variables in the individual configurations of the following components:

- Database instance
- Listener
- Oracle ASM instance

When Oracle Restart starts one of these components, it first sets environment variables for that component to the values stored in the component configuration. Although you can set environment variables that are used by Oracle components in this manner, this capability is primarily intended for operating system environment variables.

The following sections provide instructions for setting, unsetting, and viewing environment variables:

- Setting and Unsetting Environment Variables
- Viewing Environment Variables

Note: Do not use this facility to set standard environment variables like ORACLE_HOME and ORACLE_SID; these are set automatically by Oracle Restart.

Setting and Unsetting Environment Variables

You use SRVCTL to set and unset environment variable values in the Oracle Restart configuration for a component.

To set or unset environment variables in the configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Do one of the following:

- To set an environment variable in the configuration, enter the following command:

  ```
  srvctl setenv {asm|database|listener} options
  ```

- To remove an environment variable from the configuration, enter the following command:

  ```
  srvctl unsetenv {asm|database|listener} options
  ```

See the SRVCTL setenv command on page 4-56 and the unsetenv command on page 4-69 for available options for each component.

Example 4–12 Setting Database Environment Variables

This examples sets the `NLS_LANG` and the AIX `AIXTHREAD_SCOPE` environment variables in the Oracle Restart configuration for the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl setenv database -d dbcrm -t "NLS_LANG=AMERICAN_AMERICA.AL32UTF8,
   AIXTHREAD_SCOPE=S"
```

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Viewing Environment Variables

You use SRVCTL to view the values of environment variables in the Oracle Restart configuration for a component.

To view environment variable values in the configuration:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl getenv {database|listener|asm} options
   ```

   See the SRVCTL getenv command on page 4-47 for available options for each component.

Example 4–13 Viewing All Environment Variables for a Database

This example gets and displays the environment variables in the Oracle Restart configuration for the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl getenv database -d dbcrm

dbcrm:
NLS_LANG=AMERICAN_AMERICA
AIXTHREAD_SCOPE=S
GCONF_LOCAL_LOCKS=1
```

Example 4–14 Viewing Specific Environment Variables for a Database

This example gets and displays the `NLS_LANG` and `AIXTHREAD_SCOPE` environment variables from the Oracle Restart configuration for the same database:

```
srvctl getenv database -d dbcrm -t "NLS_LANG,AIXTHREAD_SCOPE"

dbcrm:
NLS_LANG=AMERICAN_AMERICA
AIXTHREAD_SCOPE=S
```

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Creating and Deleting Database Services with SRVCTL

When managing a database with Oracle Restart, Oracle recommends that you use SRVCTL to create and delete database services. When you use SRVCTL to add a database service, the service is automatically added to the Oracle Restart configuration and a dependency between the service and the database is established. Thus, if you start the service, Oracle Restart first starts the database if it is not started.

When you use SRVCTL to delete a database service, the service is also removed from the Oracle Restart configuration.

To create a database service with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl add service -d db_unique_name -s service_name [options]
   ```

   The database service is created and added to the Oracle Restart configuration. See the srvctl add service command on page 4-36 for available options.

Example 4–15 Creating a Database Service

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example creates a new database service named `crmbatch`.

```
srvctl add service -d dbcrm -s crmbatch
```

Example 4–16 Creating a Role-Based Database Service

This example creates the `crmbatch` database service and assigns it the Data Guard role of `PHYSICAL_STANDBY`. The service is automatically started only if the current role of the `dbcrm` database is physical standby.

```
srvctl add service -d dbcrm -s crmbatch -l PHYSICAL_STANDBY
```

To delete a database service with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Enter the following command:

   ```
   srvctl remove service -d db_unique_name -s service_name [-f]
   ```

   The database service is removed from the Oracle Restart configuration. If the `-f` (force) flag is present, the service is removed even if it is still running. Without this flag, an error occurs if the service is running.

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Enabling FAN Events in an Oracle Restart Environment

To enable Oracle Restart to publish Fast Application Notification (FAN) events, you must create an Oracle Notification Services (ONS) network that includes the Oracle Restart servers and the integrated clients. These clients can include Oracle Connection Manager (CMAN), Java Database Connectivity (JDBC), and Universal Connection Pool (UCP) clients. If you are using Oracle Call Interface or ODP.NET clients, then you

must enable Oracle Advanced Queuing (AQ) HA notifications for your services. In addition, ONS must be running on the server.

To enable FAN events in an Oracle Restart environment:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Add the database to the Oracle Restart Configuration if it is not already managed by Oracle Restart. See "Adding Components to the Oracle Restart Configuration" on page 4-11.
3. Add ONS to the configuration:

   ```
   srvctl add ons
   ```

   ONS is disabled when it is added.
4. Enable ONS:

   ```
   srvctl enable ons
   ```
5. Start ONS:

   ```
   srvctl start ons
   ```
6. Add the service to the Oracle Restart Configuration.

   For Oracle Call Interface and ODP.NET clients, ensure that the `-q` option is set to `TRUE` to enable the database queue.

   See "Creating and Deleting Database Services with SRVCTL" on page 4-19.
7. Enable each client for fast connection failover. See "Enabling Clients for Fast Connection Failover" on page 4-21.

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Automating the Failover of Connections Between Primary and Standby Databases

In a configuration that uses Oracle Restart and Oracle Data Guard primary and standby databases, the database services fail over automatically from the primary to the standby during either a switchover or failover. You can use Oracle Notification Services (ONS) to immediately notify clients of the failover of services between the primary and standby databases. The Oracle Data Guard Broker uses Fast Application Notification (FAN) to send notifications to clients when a failover occurs. Integrated Oracle clients automatically failover connections and applications can mask the failure from end-users.

To automate connection failover, you must create an ONS network that includes the Oracle Restart servers and the integrated clients (CMAN, listener, JDBC, and UCP). If you are using Oracle Call Interface or ODP.NET clients, you must enable the Oracle Advanced Queuing queue. The database and the services must be managed by Oracle Restart and the Oracle Data Guard Broker to automate the failover of services.

To automate the failover of services between primary and standby databases:

1. Configure the primary and standby database with the Oracle Data Guard Broker. See *Oracle Data Guard Broker*.
2. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.

3. Add the primary database to the Oracle Restart configuration on the primary server if it has not been added. Ensure that you specify `PRIMARY` for the database role. See "Adding Components to the Oracle Restart Configuration" on page 4-11.
4. Add the standby database to the Oracle Restart configuration on the standby server if it has not been added. Ensure that you specify the appropriate standby database role.
5. Enable FAN events on both the primary database server and the standby database server. "Enabling FAN Events in an Oracle Restart Environment" on page 4-19.
6. Add the services that clients will use to connect to the databases to the Oracle Restart configuration on the primary database and the standby database. When you add a service, ensure that:
   - The `-l` option is set to the proper role for each service
   - The `-q` option is set to `TRUE` if you are using ODP.NET or Oracle Call Interface

   See "Creating and Deleting Database Services with SRVCTL" on page 4-19.
7. Enable each client for fast connection failover. See "Enabling Clients for Fast Connection Failover" on page 4-21.

See Also: "SRVCTL Command Reference for Oracle Restart" on page 4-30

Enabling Clients for Fast Connection Failover

In a configuration with a standby database, after you have added Oracle Notification Services (ONS) to your Oracle Restart configurations and enabled Oracle Advanced Queuing (AQ) HA notifications for your services, you can enable clients for fast connection failover. The clients receive Fast Application Notification (FAN) events and can relocate connections to the current primary database after an Oracle Data Guard failover. See "Automating the Failover of Connections Between Primary and Standby Databases" on page 4-20 for information about adding ONS.

For databases with no standby database configured, you can still configure the client FAN events. When there is a failure, you can configure the client to retry the connection to the database. Since Oracle Restart will restart the failed database, the client can reconnect when the database restarts. Ensure that you program the appropriate delay and retries on the connection string, as illustrated in the examples in this section.

You can enable fast connection failover for the following types of clients in an Oracle Restart configuration:

- Enabling Fast Connection Failover for JDBC Clients
- Enabling Fast Connection Failover for Oracle Call Interface Clients
- Enabling Fast Connection Failover for ODP.NET Clients

Enabling Fast Connection Failover for JDBC Clients

Enabling FAN for the Oracle Universal Connection Pool enables Fast Connection Failover (FCF) for the client. Your application can use either thick or thin JDBC clients to use FCF.

To configure the JDBC client, set the `FastConnectionFailoverEnabled` property before making the first `getConnection()` request to a data source. When you enable Fast Connection Failover, the failover applies to every connection in the connection cache.

If your application explicitly creates a connection cache using the Connection Cache Manager, then you must first set `FastConnectionFailoverEnabled`.

This section describes how to enable FCF for JDBC with the Universal Connection Pool. For thick JDBC clients, if you enable Fast Connection Failover, do not enable Transparent Application Failover (TAF), either on the client or for the service. Enabling FCF with thin or thick JDBC clients enables the connection pool to receive and react to all FAN events.

To enable Fast Connection Failover for JDBC clients:

1. On a cache enabled DataSource, set the DataSource property `FastConnectionFailoverEnabled` to `true` as in the following example to enable FAN for the Oracle JDBC Implicit Connection Cache:

   ```
   PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
   pds.setONSConfiguration("nodes=primaryhost:6200,standbyhost:6200");
   pds.setFastConnectionFailoverEnabled(true);
   pds.setURL("jdbc:oracle:thin:@(DESCRIPTION=
     (LOAD_BALANCE=on)
     (ADDRESS=(PROTOCOL=TCP)(HOST=primaryhost)(PORT=1521))
     (ADDRESS=(PROTOCOL=TCP)(HOST=standbyhost)(PORT=1521))
     (CONNECT_DATA=(service_name=service_name)))");

   ......
   ```

   In this example, `primaryhost` is the server for the primary database, and `standbyhost` is the server for the standby database.

   Applications must have both ucp.jar and ons.jar in their `CLASSPATH`.

   > **Note:** Use the following system property to enable FAN without making data source changes: `-D oracle.jdbc.FastConnectionFailover=true`.

2. When you start the application, ensure that the ons.jar file is located on the application `CLASSPATH`. The ons.jar file is part of the Oracle client installation.

   See Also: *Oracle Database JDBC Developer's Guide*

Enabling Fast Connection Failover for Oracle Call Interface Clients

Oracle Call Interface clients can enable Fast Connection Failover (FCF) by registering to receive notifications about Oracle Restart high availability FAN events and respond when events occur. This improves the session failover response time in Oracle Call Interface and removes terminated connections from connection and session pools. This feature works on Oracle Call Interface applications, including those that use Transparent Application Failover (TAF), connection pools, or session pools.

First, you must enable a service for high availability events to automatically populate the Advanced Queuing `ALERT_QUEUE`. If your application is using TAF, then enable the TAF settings for the service. Configure client applications to connect to an Oracle Restart database. Clients can register callbacks that are used whenever an event occurs. This reduces the time that it takes to detect a connection failure.

During `DOWN` event processing, Oracle Call Interface:

- Terminates affected connections at the client and returns an error

- Removes connections from the Oracle Call Interface connection pool and the Oracle Call Interface session pool

  The session pool maps each session to a physical connection in the connection pool, and there can be multiple sessions for each connection.
- Fails over the connection if you have configured TAF

If TAF is not configured, then the client only receives an error.

> **Note:** Oracle Call Interface does not manage `UP` events.

To Enable Fast Connection Failover for an Oracle Call Interface client:

1. Ensure that the service that you are using has Advanced Queuing notifications enabled by setting the services' values using the SRVCTL `modify` command. For example:

   ```
   srvctl modify service -d proddb -s gl.us.example.com -q true -l primary -e
   select -m basic -z 5 -w 180 -j long
   ```

2. Enable `OCI_EVENTS` at environment creation time on the client as follows:

   ```
   ( OCIEnvCreate(...) )
   ```

3. Link client applications with the client thread or operating system library.
4. Optionally, register a client `EVENT` callback.
5. Ensure that the client uses an Oracle Net connect descriptor that includes all primary and standby hosts in the `ADDRESS_LIST`. For example:

   ```
   gl =
   (DESCRIPTION =
    (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = BOSTON1)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = CHICAGO1)(PORT = 1521))
        (LOAD_BALANCE = yes)
   )
   (CONNECT_DATA=
     (SERVICE_NAME=gl.us.example.com)))
   ```

To see the alert information, query the views `DBA_OUTSTANDING_ALERTS` and `DBA_ALERT_HISTORY`.

> **See Also:**
>
> - *Oracle Call Interface Programmer's Guide*
> - *Oracle Database Net Services Administrator's Guide* for information about configuring TAF

Enabling Fast Connection Failover for ODP.NET Clients

Oracle Data Provider for .NET (ODP.NET) connection pools can subscribe to notifications that indicate when services are down. After a `DOWN` event, Oracle Database cleans up sessions in the connection pool that go to the instance that stops, and ODP.NET proactively disposes connections that are no longer valid.

To enable Fast Connection Failover for ODP.NET clients:

1. Enable Advanced Queuing notifications by using SRVCTL `modify service` command, as in the following example:

   ```
   srvctl modify service -d dbname -s gl -q true, -j long
   ```

2. Execute the following for the users that will be connecting by way of the .Net Application, where *user_name* is the user name:

   ```
   execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE','SYS.SYS$SERVICE_METRICS',
   user_name);
   ```

3. Enable Fast Connection Failover for ODP.NET connection pools by subscribing to FAN high availability events. Set the `HA` events connection string attribute to true at connection time. The pooling attribute must be set to `true`, which is the default. The following example illustrates these settings, where *user_name* is the name of the user and *password* is the user password:

   ```
   // C#
   using System;
   using Oracle.DataAccess.Client;

   class HAEventEnablingSample
   {
     static void Main()
     {
       OracleConnection con = new OracleConnection();

       // Open a connection using ConnectionString attributes
       // Also, enable "load balancing"
       con.ConnectionString =
         "User Id=user_name;Password=password;Data Source=oracle;" +
         "Min Pool Size=10;Connection Lifetime=120;Connection Timeout=60;" +
         "HA Events=true;Incr Pool Size=5;Decr Pool Size=2";

       con.Open();

       // Create more connections and perform work against the database here.

       // Dispose OracleConnection object
       con.Dispose();
     }
   }
   ```

4. Ensure that the client uses an Oracle Net connect descriptor that includes all primary and standby hosts in the `ADDRESS_LIST`. For example:

   ```
   gl =
   (DESCRIPTION =
    (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = BOSTON1)(PORT = 1521))
        (ADDRESS = (PROTOCOL = TCP)(HOST = CHICAGO1)(PORT = 1521))
        (LOAD_BALANCE = yes)
   )
   (CONNECT_DATA=
     (SERVICE_NAME=gl.us.example.com)))
   ```

See Also:

- *Oracle Data Provider for .NET Developer's Guide* for information about ODP.NET
- "SRVCTL Command Reference for Oracle Restart" on page 4-30

Starting and Stopping Components Managed by Oracle Restart

When Oracle Restart is in use, Oracle strongly recommends that you use the SRVCTL utility to start and stop components, for the following reasons:

- When starting a component with SRVCTL, Oracle Restart can first start any components on which this component depends. When stopping a component with SRVCTL, Oracle Restart can stop any dependent components first.
- SRVCTL always starts a component according to its Oracle Restart configuration. Starting a component by other means may not.

  For example, if you specified a server parameter file (SPFILE) location when you added a database to the Oracle Restart configuration, and that location is not the default location for SPFILEs, if you start the database with SQL*Plus, the SPFILE specified in the configuration may not be used.

  See the srvctl add database command on page 4-33 for a table of configuration options for a database instance.
- When you start a component with SRVCTL, environment variables stored in the Oracle Restart configuration for the component are set.

  See "Managing Environment Variables in the Oracle Restart Configuration" on page 4-17 for more information.

You can also use Oracle Enterprise Manager Database Control (Database Control) to start a database managed by Oracle Restart. Both the SRVCTL and Database Control methods are described in the following sections:

- Starting and Stopping Components Managed by Oracle Restart with SRVCTL
- Starting a Database Managed by Oracle Restart with Oracle Enterprise Manager

Starting and Stopping Components Managed by Oracle Restart with SRVCTL

You can start and stop any component managed by Oracle Restart with SRVCTL.

To start or stop a component managed by Oracle Restart with SRVCTL:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Do one of the following:
   - To start a component, enter the following command:

     ```
     srvctl start object [options]
     ```

   - To stop a component, enter the following command:

     ```
     srvctl stop object [options]
     ```

   where *object* is one of the components listed in Table 4–8 on page 4-31. See the SRVCTL start command on page 4-58 and the stop command on page 4-65 for available options for each component.

Example 4–17 Starting a Database

This example starts the database with a DB_UNIQUE_NAME of dbcrm:

```
srvctl start database -d dbcrm
```

Example 4–18 Starting a Database NOMOUNT

This example starts the database instance without mounting the database:

```
srvctl start database -d dbcrm -o nomount
```

Example 4–19 Starting the Default Listener

This example starts the default listener:

```
srvctl start listener
```

Example 4–20 Starting a Specified Listener

This example starts the listener named crmlistener:

```
srvctl start listener -l crmlistener
```

Example 4–21 Starting Database Services

This example starts the database services bizdev and support for the database with a DB_UNIQUE_NAME of dbcrm. If the database is not started, Oracle Restart first starts the database.

```
srvctl start service -d dbcrm -s "bizdev,support"
```

Example 4–22 Starting (Mounting) Oracle ASM Disk Groups

This example starts (mounts) the Oracle ASM disk groups data and recovery. The user running this command must be a member of the OSASM group.

```
srvctl start diskgroup -g "data,recovery"
```

Example 4–23 Shutting Down a Database

This example stops (shuts down) the database with a DB_UNIQUE_NAME of dbcrm. Because a stop option (-o) is not provided, the database shuts down according to the stop option in its Oracle Restart configuration. The default stop option is IMMEDIATE.

```
srvctl stop database -d dbcrm
```

Example 4–24 Shutting Down a Database with the ABORT option

This example does a SHUTDOWN ABORT of the database with a DB_UNIQUE_NAME of dbcrm.

```
srvctl stop database -d dbcrm -o abort
```

See Also: The SRVCTL start command on page 4-58

Starting a Database Managed by Oracle Restart with Oracle Enterprise Manager

With Oracle Enterprise Manager Database Control (Database Control), you can use Oracle Restart to start a database.

To start a database managed by Oracle Restart with Oracle Enterprise Manager:

1. Access the Database Home page for the desired database instance.

See "Accessing the Database Home Page" in *Oracle Database 2 Day DBA* for instructions.

2. Click **Startup**.

   The Startup/Shutdown Credentials page appears.

3. Enter credentials as follows:

   a. Enter the host computer credentials for the user who installed the database Oracle home.

   b. Enter the database credentials consisting of the user name `SYS` and the password that you assigned to `SYS` during the installation.

   c. In the Connect As list, choose the value `SYSOPER`.

4. (Optional) Select the **Save as Preferred Credential** option if you want these credentials to be automatically filled in for you the next time that this page appears.

5. Click **OK**.

   The Select Startup Type page appears.

6. To start the database with Oracle Restart, select **Start database along with dependent resources**.

   This ensures that resources on which the database depends, such as the Oracle Automatic Storage Management instance, are successfully started before the database is started.

7. Click **OK**.

   A confirmation page appears.

8. Click **Yes**.

   The Startup/Shutdown: Activity Information page appears, indicating that the database is being started up. When startup is complete, the Login page appears.

9. Log in to the database (and to Database Control).

   The Database Home page appears indicating that the database instance status is Up.

Stopping and Restarting Oracle Restart for Maintenance Operations

When several components in an Oracle home are managed by Oracle Restart, you can stop Oracle Restart and the components managed by Oracle Restart in the Oracle home. You can also disable Oracle Restart so that it is not restarted if the node reboots. You might need to do this when you are performing maintenance that includes the Oracle home, such as installing a patch. When the maintenance operation is complete, you can enable and restart Oracle Restart, and you can restart the components managed by Oracle Restart in the Oracle home.

Use both the SRVCTL utility and the CRSCTL utility for the stop and start operations:

- The `stop home` SRVCTL command stops all of the components that are managed by Oracle Restart in the specified Oracle home. The `start home` SRVCTL command starts these components. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

When you use the `home` object, a state file, specified in the `-s` option, tracks the state of each component. The `stop` and `status` commands create the state file. The `start` command uses the state file to identify the components to restart.

In addition, you can check the status of the components managed by Oracle Restart using the `status home` command.

- The `stop` CRSCTL command stops Oracle Restart, and the `disable` CRSCTL command ensures that the components managed by Oracle Restart do not restart automatically. The `enable` CRSCTL command enables automatic restart and the `start` CRSCTL command restarts Oracle Restart.

To stop and start the components in an Oracle home while installing a patch:

1. Prepare to run SRVCTL as described in "Preparing to Run SRVCTL" on page 4-10.
2. Use the SRVCTL utility to stop the components managed by Oracle Restart in an Oracle home:

   ```
   srvctl stop home -o oracle_home -s state_file [-t stop_options] [-f]
   ```

   where *`oracle_home`* is the complete path of the Oracle home and `state_file` is the complete path to the state file. State information for the Oracle home is recorded in the specified state file. Make a note of the state file location because it must be specified in Step 7.

   Before stopping the components in an Oracle Grid Infrastructure home, ensure that you first stop the components in a dependent Oracle Database home.
3. If you are patching an Oracle Grid Infrastructure home, then disable and stop Oracle Restart. Otherwise, go to Step 4.

   To disable and stop Oracle Restart, use the CRSCTL utility to run the following commands:

   ```
   crsctl disable has
   ```

   ```
   crsctl stop has
   ```
4. Perform the maintenance operation.
5. Use the CRSCTL utility to enable automatic restart of the components managed by Oracle Restart:

   ```
   crsctl enable has
   ```
6. Use the CRSCTL utility to start Oracle Restart:

   ```
   crsctl start has
   ```
7. Use the SRVCTL utility to start the components that were stopped in Step 2:

   ```
   srvctl start home -o oracle_home -s state_file
   ```

   The state file must match the state file specified in Step 2.
8. Optionally, use the SRVCTL utility to check the status of the components managed by Oracle Restart in the Oracle home:

   ```
   srvctl status home -o oracle_home -s state_file
   ```

Example 4–25 Stopping Components Managed by Oracle Restart in an Oracle Home

```
srvctl stop home -o /u01/app/oracle/product/11.2.0/dbhome_1 -s /usr1/or_state
```

Example 4–26 Starting Components Managed by Oracle Restart in an Oracle Home

```
srvctl start home -o /u01/app/oracle/product/11.2.0/dbhome_1 -s /usr1/or_state
```

Example 4–27 Displaying the Status of Components Managed by Oracle Restart in an Oracle Home

```
srvctl status home -o /u01/app/oracle/product/11.2.0/dbhome_1 -s /usr1/or_state
```

See Also:

- The srvctl stop home command on page 4-66
- The srvctl status home command on page 4-63
- The srvctl start home command on page 4-59
- "CRSCTL Command Reference" on page 4-71

SRVCTL Command Reference for Oracle Restart

This section provides details about the syntax and options for SRVCTL commands specific to Oracle Restart. See *Oracle Real Application Clusters Administration and Deployment Guide* for the full list of SRVCTL commands.

SRVCTL Command Syntax and Options Overview

SRVCTL expects the following command syntax:

```
srvctl command object options
```

where:

- *command* is a verb such as `start`, `stop`, or `remove`. See Table 4–7 on page 4-30 for a complete list.
- *object* is the component on which SRVCTL performs the command, such as database, listener, and so on. You can also use component abbreviations. See Table 4–8 on page 4-31 for a complete list of components and their abbreviations.
- *options* extend the use of a preceding command combination to include additional parameters for the command. For example, the `-d` option indicates that a database unique name follows, and the `-s` option indicates that a comma-delimited list of database service names follows.

> **Note:** On the Windows platform, when specifying a comma-delimited list, you must enclose the list within double-quotes ("...,..."). You must also use double-quotes on the UNIX and Linux platforms if any list member contains shell metacharacters.

Case Sensitivity SRVCTL commands and components are case insensitive. Options are case sensitive. Database and database service names are case insensitive and case preserving.

Table 4–7 Summary of SRVCTL Commands

Command	Description
add on page 4-32	Adds a component to the Oracle Restart configuration.
config on page 4-38	Displays the Oracle Restart configuration for a component.
disable on page 4-41	Disables management by Oracle Restart for a component.
enable on page 4-44	Reenables management by Oracle Restart for a component.
getenv on page 4-47	Displays environment variables in the Oracle Restart configuration for a database, Oracle ASM instance, or listener.
modify on page 4-49	Modifies the Oracle Restart configuration for a component.
remove on page 4-53	Removes a component from the Oracle Restart configuration.
setenv on page 4-56	Sets environment variables in the Oracle Restart configuration for a database, Oracle ASM instance, or listener.
start on page 4-58	Starts the specified component.
status on page 4-62	Displays the running status of the specified component.
stop on page 4-65	Stops the specified component.

Table 4–7 (Cont.) Summary of SRVCTL Commands

Command	Description
unsetenv on page 4-69	Unsets environment variables in the Oracle Restart configuration for a database, Oracle ASM instance, or listener.

SRVCTL Components Summary

Table 4–8 lists the keywords that can be used for the *object* portion of SRVCTL commands. You can use either the full name or the abbreviation for each component keyword.

Table 4–8 Component Keywords and Abbreviations

Component	Abbreviation	Description
asm	asm	Oracle ASM instance
database	db	Database instance
diskgroup	dg	Oracle ASM disk group
filesystem	filesystem	Oracle ASM file system
home	home	Oracle home or Oracle Clusterware home
listener	lsnr	Oracle Net listener
service	serv	Database service
ons	ons	Oracle Notification Services (ONS)

See Also: Table 4–1, " Oracle Components Automatically Restarted by Oracle Restart" on page 4-2

add

The `srvctl add` command adds the specified component to the Oracle Restart configuration, and optionally sets Oracle Restart configuration parameters for the component. After a component is added, Oracle Restart begins to manage it, restarting it when required.

To perform `srvctl add` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–9 srvctl add Summary

Command	Description
srvctl add asm on page 4-32	Adds an Oracle ASM instance.
srvctl add database on page 4-33	Adds a database.
srvctl add listener on page 4-34	Adds a listener.
srvctl add ons on page 4-35	Adds an ONS (used by Oracle Data Guard configurations with Oracle Data Guard Broker).
srvctl add service on page 4-36	Adds a database service managed by Oracle Restart.

Note: There is no `srvctl add` command for Oracle ASM disk groups. Disk groups are automatically added to the Oracle Restart configuration when they are first mounted. If you remove a disk group from the Oracle Restart configuration and later want to add it back, connect to the Oracle ASM instance with SQL*Plus and use an `ALTER DISKGROUP ... MOUNT` command.

srvctl add asm

Adds an Oracle ASM instance to the Oracle Restart configuration.

Syntax and Options

Use the `srvctl add asm` command with the following syntax:

```
srvctl add asm [-l listener_name [-p spfile] [-d asm_diskstring]
]
```

Table 4–10 srvctl add asm Options

Option	Description
`-l listener_name`	Name of the listener with which Oracle ASM should register. A weak dependency is established with this listener. (Before starting the Oracle ASM instance, Oracle Restart attempts to start the listener. If the listener does not start, the Oracle ASM instance is still started. If the listener later fails, Oracle Restart does not restart Oracle ASM.) If omitted, defaults to the listener named `listener`.
`-p spfile`	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.

Table 4–10 (Cont.) srvctl add asm Options

Option	Description
`-d asm_diskstring`	Oracle ASM disk group discovery string. An Oracle ASM discovery string is a comma-delimited list of strings that limits the set of disks that an Oracle ASM instance discovers. The discovery strings can include wildcard characters. Only disks that match one of the strings are discovered.

Example

An example of this command is:

```
srvctl add asm -l crmlistener
```

> **See Also:** *Oracle Automatic Storage Management Administrator's Guide* for more information about Oracle ASM disk group discovery strings

srvctl add database

Adds a database to the Oracle Restart configuration.

After adding a database to the Oracle Restart configuration, if the database then accesses data in an Oracle ASM disk group, a dependency between the database that disk group is created. Oracle Restart then ensures that the disk group is mounted before attempting to start the database.

However, if the database and Oracle ASM instance are not running when you add the database to the Oracle Restart configuration, you must manually establish the dependency between the database and its disk groups by specifying the `-a` option in the SRVCTL command. See the example later in this section.

> **Important:** When you manually add a database to the Oracle Restart configuration, you must also add the Oracle grid infrastructure software owner as a member of the OSDBA group of that database. This is because the grid infrastructure components must be able to connect to the database as `SYSDBA` to start and stop the database.
>
> For example, if the host user who installed the grid infrastructure home is named `grid` and the OSDBA group of the new database is named `dba`, then user `grid` must be a member of the `dba` group.

Syntax and Options

Use the `srvctl add database` command with the following syntax:

```
srvctl add database -d db_unique_name -o oracle_home [-m domain_name]
  [-n db_name] [-p spfile] [-s start_options] [-t stop_options]
  [-r {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY | SNAPSHOT_STANDBY}]
  [-y {automatic | manual}] [-a disk_group_list]
```

Table 4–11 srvctl add database Options

Syntax	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-o Oracle_home`	The full path of Oracle home for the database.

Table 4–11 (Cont.) srvctl add database Options

Syntax	Description
`-m domain_name`	The domain for the database. Must match the `DB_DOMAIN` initialization parameter
`-n db_name`	If provided, must match the `DB_NAME` initialization parameter setting. You must include this option if `DB_NAME` is different from the unique name given by the `-d` option
`-p spfile`	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.
`-s start_options`	Startup options for the database (`OPEN`, `MOUNT`, or `NOMOUNT`). If omitted, defaults to `OPEN`.
`-t stop_options`	Shutdown options for the database (`NORMAL`, `IMMEDIATE`, `TRANSACTIONAL`, or `ABORT`). If omitted, defaults to `IMMEDIATE`.
`-r {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY | SNAPSHOT_STANDBY}`	The current role of the database (`PRIMARY`, `PHYSICAL_STANDBY`, `SNAPSHOT_STANDBY`, or `LOGICAL_STANDBY`). Applicable in Oracle Data Guard environments only.
`-y {AUTOMATIC | MANUAL}`	Management policy for the database. If `AUTOMATIC` (the default), the database is automatically restored to its previous running condition (started or stopped) upon restart of the database host computer. If `MANUAL`, the database is never automatically restarted upon restart of the database host computer. A `MANUAL` setting does not prevent Oracle Restart from monitoring the database while it is running and restarting it if a failure occurs.
`-a disk_group_list`	List of disk groups upon which the database is dependent. When starting the database, Oracle Restart first ensures that these disk groups are mounted. This option is required only if the database instance and the Oracle ASM instance are not started when adding the database. Otherwise, the dependency is recorded automatically between the database and its disk groups.

Examples

This example adds the database with the `DB_UNIQUE_NAME` `dbcrm`:

```
srvctl add database -d dbcrm -o /u01/app/oracle/product/11.2.0/dbhome_1
```

This example adds the same database and also establishes a dependency between the database and the disk groups `DATA` and `RECOVERY`.

```
srvctl add database -d dbcrm -o /u01/app/oracle/product/11.2.0/dbhome_1
  -a "DATA,RECOVERY"
```

See Also:

- "Oracle Restart Integration with Oracle Data Guard" on page 4-5
- *Oracle Data Guard Concepts and Administration*

srvctl add listener

Adds a listener to the Oracle Restart configuration.

Syntax and Options

Use the `srvctl add listener` command with the following syntax:

```
srvctl add listener [-l listener_name] [-p endpoints] [-s] [-o Oracle_home]
```

Table 4–12 srvctl add listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, defaults to `LISTENER`
`-p endpoints`	Comma separated TCP ports or listener endpoints. If omitted, defaults to TCP:1521. *endpoints* syntax is: `"[TCP:]port[, ...] [/IPC:key] [/NMP:pipe_name] [/TCPS:s_port] [/SDP:port]"`
`-s`	Skip checking for port conflicts with the supplied endpoints
`-o Oracle_home`	Oracle home for the listener. If omitted, the Oracle Grid Infrastructure home is assumed.

Example

The following command adds a listener (named `LISTENER`) running out of the database Oracle home and listening on TCP port 1522:

```
srvctl add listener -p TCP:1522 -o /u01/app/oracle/product/11.2.0/dbhome_1
```

srvctl add ons

Adds an ONS to an Oracle Restart configuration.

ONS must be added to an Oracle Restart configuration to enable the sending of Fast Application Notification (FAN) events after an Oracle Data Guard failover.

When ONS is added to an Oracle Restart configuration, it is initially disabled. You can enable it with the `srvctl enable ons` command.

> **See Also:** "srvctl enable ons" on page 4-45

Syntax and Options

Use the `srvctl add ons` command with the following syntax:

```
srvctl add ons [-e em_port] [-l ons_local_port] [-r ons_remote_port] [-t
host[:port],[host[:port]...]] [-v]
```

Table 4–13 srvctl add ons Options

Option	Description
`-e em_port`	ONS listening port for Oracle Enterprise Manager. The default is 2016.
`-l ons_local_port`	ONS listening port for local client connections. The default is 6100.
`-r ons_remote_port`	ONS listening port for connections from remote hosts. The default is 6200.
`-t host[:port],[host[:port],...]`	A list of `host:port` pairs of remote hosts that are part of the ONS network **Note:** If `port` is not specified for a remote host, then `ons_remote_port` is used.
`-v`	Verbose output

srvctl add service

Adds a database service to the Oracle Restart configuration. Creates the database service if it does not exist. This method of creating a service is preferred over using the DBMS_SERVICE PL/SQL package.

Syntax and Options

Use the srvctl add service command with the following syntax:

```
srvctl add service -d db_unique_name -s service_name
  [-l [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]]
  [-y {AUTOMATIC | MANUAL}] [-e {NONE | SESSION | SELECT}] [-m {NONE | BASIC}]
  [-w integer] [-z integer] [-j {SHORT | LONG}]
  [-B {SERVICE_TIME | THROUGHPUT | NONE}] [-q {TRUE | FALSE}] -t edition_name
```

Table 4–14 srvctl add service Options

Option	Description
-d *db_unique_name*	Unique name for the database. Must match the DB_UNIQUE_NAME initialization parameter setting. If DB_UNIQUE_NAME is unspecified, then this option must match the DB_NAME initialization parameter setting. The default setting for DB_UNIQUE_NAME uses the setting for DB_NAME.
-s *service_name*	The database service name
-l [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]	A list of service roles. Applicable in Oracle Data Guard environments only. When this option is present, upon database startup, the service is started only when one of its service roles matches the current database role.
-y {AUTOMATIC | MANUAL}	Management policy for the service. If AUTOMATIC (the default), the service is automatically started upon restart of the database, either by a planned restart (with SRVCTL) or after a failure. Automatic restart is also subject to the service role, however (the -l option). If MANUAL, the service is never automatically restarted upon planned restart of the database (with SRVCTL). A MANUAL setting does not prevent Oracle Restart from monitoring the service when it is running and restarting it if a failure occurs.
-e {NONE |SESSION | SELECT}	Failover type. For standalone servers, applicable in Oracle Data Guard environments only.
-m {NONE | BASIC}	Failover method. For standalone servers, applicable in Oracle Data Guard environments only.
-w *integer*	Failover delay. For standalone servers, applicable in Oracle Data Guard environments only.
-z *integer*	Failover retries. For standalone servers, applicable in Oracle Data Guard environments only.
-j {SHORT | LONG}	Connection load balancing goal
-B {SERVICE_TIME | THROUGHPUT | NONE}	Run-time load balancing goal
-q {TRUE | FALSE}	Send Oracle Advanced Queuing (AQ) HA notifications. For standalone servers, applicable in Oracle Data Guard environments only.

Table 4–14 (Cont.) srvctl add service Options

Option	Description
`-t edition_name`	The initial session edition of the service When an edition is specified for a service, all subsequent connections that specify the service use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the initial session edition. SRVCTL does not validate the specified edition name. During connection, the connect user must have `USE` privilege on the specified edition. If the edition does not exist or if the connect user does not have `USE` privilege on the specified edition, then an error is raised.

Example

This example adds the `sales` service for the database with `DB_UNIQUE_NAME` `dbcrm`. The service is started only when `dbcrm` is in `PRIMARY` mode.

```
srvctl add service -d dbcrm -s sales -l PRIMARY
```

See Also:

- The section in *Oracle Database PL/SQL Packages and Types Reference* on the `DBMS_SERVICE` package for more information about the options for this command
- "Oracle Restart Integration with Oracle Data Guard" on page 4-5
- *Oracle Data Guard Concepts and Administration*

config

The `srvctl config` command displays the Oracle Restart configuration of the specified component or set of components.

Table 4–15 srvctl config Summary

Command	Description
srvctl config asm on page 4-38	Displays the Oracle Restart configuration information for the Oracle ASM instance
srvctl config database on page 4-38	Displays the Oracle Restart configuration information for the specified database, or lists all databases managed by Oracle Restart
srvctl config listener on page 4-39	Displays the Oracle Restart configuration information for all listeners or for the specified listener
srvctl config ons on page 4-40	Displays the current configuration information for ONS.
srvctl config service on page 4-40	For the specified database, displays the Oracle Restart configuration information for the specified database service or for all database services

srvctl config asm

Displays the Oracle Restart configuration information for the Oracle ASM instance.

Syntax and Options

Use the `srvctl config asm` command with the following syntax:

```
srvctl config asm [-a]
```

Table 4–16 srvctl config asm Options

Option	Description
`-a`	Display enabled/disabled status also

Example

An example of this command is:

```
srvctl config asm -a

asm home: /u01/app/oracle/product/11.2.0/grid
ASM is enabled.
```

srvctl config database

Displays the Oracle Restart configuration information for the specified database, or lists all databases managed by Oracle Restart.

Syntax and Options

Use the `srvctl config database` command with the following syntax:

```
srvctl config database [-d db_unique_name [-a]]
```

Table 4–17 srvctl config database Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-a`	Display enabled/disabled status also

Examples

An example of this command to list all Oracle Restart–managed databases is:

```
srvctl config database

dbcrm
orcl
```

An example of this command to display configuration and enabled/disabled status for the database with the `DB_UNIQUE_ID` `orcl` is:

```
srvctl config database -d orcl -a

Database unique name: orcl
Database name: orcl
Oracle home: /u01/app/oracle/product/11.2.0/dbhome_1
Oracle user: oracle
Spfile: +DATA/orcl/spfileorcl.ora
Domain: us.example.com
Start options: open
Stop options: immediate
Database role:
Management policy: automatic
Disk Groups: DATA
Services: mfg,sales
Database is enabled
```

srvctl config listener

Displays the Oracle Restart configuration information for all Oracle Restart–managed listeners or for the specified listener.

Syntax and Options

Use the `srvctl config listener` command with the following syntax:

```
srvctl config listener [-l listener_name]
```

Table 4–18 srvctl config listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, configuration information for all Oracle Restart–managed listeners is displayed.

Example

This example displays the configuration information and enabled/disabled status for the default listener:

```
srvctl config listener
```

```
Name: LISTENER
Home: /u01/app/oracle/product/11.2.0/dbhome_1
End points: TCP:1521
Listener is enabled.
```

srvctl config ons

Displays the current configuration information for ONS.

Syntax and Options

Use the `srvctl config ons` command with the following syntax:

```
srvctl config ons
```

srvctl config service

For the specified database, displays the Oracle Restart configuration information for the specified database service or for all Oracle Restart–managed database services.

Syntax and Options

Use the `srvctl config service` command with the following syntax:

```
srvctl config service -d db_unique_name [-s service_name] [-a]
```

Table 4–19 srvctl config service Options

Option	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s` *`service_name`*	Database service name. If omitted, SRVCTL displays configuration information for all Oracle Restart–managed services for the database
`-a`	Display detailed configuration information

Example

An example of this command is:

```
srvctl config service -d dbcrm -s sales

Service name: sales
Service is enabled
Cardinality: SINGLETON
Disconnect: true
Service role: PRIMARY
Management policy: automatic
DTP transaction: false
AQ HA notifications: false
Failover type: NONE
Failover method: NONE
TAF failover retries: 0
TAF failover delay: 0
Connection Load Balancing Goal: NONE
Runtime Load Balancing Goal: NONE
TAF policy specification: NONE
Edition: e2
```

disable

Disables a component, which suspends management of that component by Oracle Restart. The `srvctl disable` command is intended to be used when a component must be repaired or shut down for maintenance, and should not be restarted automatically. When you disable a component:

- It is no longer automatically restarted.
- It is no longer automatically started through a dependency.
- It cannot be started with SRVCTL.

To perform `srvctl disable` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

See Also: The enable command on page 4-44

Table 4–20 srvctl disable Summary

Command	Description
srvctl disable asm on page 4-41	Disables the Oracle ASM instance
srvctl disable database on page 4-41	Disables a database
srvctl disable diskgroup on page 4-42	Disables an Oracle ASM disk group
srvctl disable listener on page 4-42	Disables the specified listener or all listeners
srvctl disable ons on page 4-42	Disables ONS
srvctl disable service on page 4-42	Disables one or more database services for the specified database

srvctl disable asm

Disables the Oracle ASM instance.

Syntax and Options

Use the `srvctl disable asm` command with the following syntax:

```
srvctl disable asm
```

srvctl disable database

Disables the specified database.

Syntax and Options

Use the `srvctl disable database` command with the following syntax:

```
srvctl disable database -d db_unique_name
```

Table 4–21 srvctl disable database Options

Option	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

Example

An example of this command is:

```
srvctl disable database -d dbcrm
```

srvctl disable diskgroup

Disables an Oracle ASM disk group.

Syntax and Options

Use the `srvctl disable diskgroup` command with the following syntax:

```
srvctl disable diskgroup -g diskgroup_name
```

Table 4–22 srvctl disable diskgroup Options

Option	Description
`-g` *`diskgroup_name`*	Disk group name

Example

An example of this command is:

```
srvctl disable diskgroup -g DATA
```

srvctl disable listener

Disables the specified listener or all listeners.

Syntax and Options

Use the `srvctl disable listener` command with the following syntax:

```
srvctl disable listener [-l listener_name]
```

Table 4–23 srvctl disable listener Options

Option	Description
`-l` *`listener_name`*	Listener name. If omitted, all listeners are disabled.

Example

An example of this command is:

```
srvctl disable listener -l crmlistener
```

srvctl disable ons

Disables ONS.

Syntax and Options

Use the `srvctl disable ons` command with the following syntax:

```
srvctl disable ons -v
```

Table 4–24 srvctl disable ons Options

Option	Description
`-v`	Verbose output

srvctl disable service

Disables one or more database services.

Syntax and Options

Use the `srvctl disable service` command with the following syntax:

```
srvctl disable service -d db_unique_name -s service_name_list
```

Table 4–25 srvctl disable service Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s service_name_list`	Comma-delimited list of database service names

Example

The following example disables the database service `sales` and `mfg`:

```
srvctl disable service -d dbcrm -s sales,mfg
```

enable

The `srvctl enable` command reenables the specified disabled component so that:

- Oracle Restart can automatically restart it.
- It can be automatically started through a dependency.
- You can start it manually with SRVCTL.

If the component is already enabled, then the command is ignored.

When you add a component to the Oracle Restart configuration, it is enabled by default.

To perform `srvctl enable` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–26 srvctl enable Summary

Command	Description
srvctl enable asm on page 4-44	Enables an Oracle ASM instance.
srvctl enable database on page 4-44	Enables a database.
srvctl enable diskgroup on page 4-45	Enables an Oracle ASM disk group.
srvctl enable listener on page 4-45	Enables the specified listener or all listeners.
srvctl enable ons on page 4-45	Enables ONS.
srvctl enable service on page 4-46	Enables one or more database services for the specified database.

See Also: The disable command on page 4-41

srvctl enable asm

Enables an Oracle ASM instance.

Syntax and Options

Use the `srvctl enable asm` command with the following syntax:

```
srvctl enable asm
```

srvctl enable database

Enables the specified database.

Syntax and Options

Use the `srvctl enable database` command with the following syntax:

```
srvctl enable database -d db_unique_name
```

Table 4–27 srvctl enable database Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.

Example

An example of this command is:

```
srvctl enable database -d dbcrm
```

srvctl enable diskgroup

Enables an Oracle ASM disk group.

Syntax and Options

Use the `srvctl enable diskgroup` command with the following syntax:

```
srvctl enable diskgroup -g diskgroup_name
```

Table 4–28 srvctl enable diskgroup Options

Option	Description
`-g diskgroup_name`	Disk group name

Example

An example of this command is:

```
srvctl enable diskgroup -g DATA
```

srvctl enable listener

Enables the specified listener or all listeners.

Syntax and Options

Use the `srvctl enable listener` command with the following syntax:

```
srvctl enable listener [-l listener_name]
```

Table 4–29 srvctl enable listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, all listeners are enabled.

Example

An example of this command is:

```
srvctl enable listener -l crmlistener
```

srvctl enable ons

Enables ONS.

Syntax and Options

Use the `srvctl enable ons` command with the following syntax:

```
srvctl enable ons -v
```

Table 4–30 srvctl enable ons Options

Option	Description
-v	Verbose output

srvctl enable service

Enables one or more database services for the specified database.

Syntax and Options

Use the srvctl enable service command with the following syntax:

```
srvctl enable service -d db_unique_name -s service_name_list
```

Table 4–31 srvctl enable service Options

Option	Description
-d *db_unique_name*	Unique name for the database. Must match the DB_UNIQUE_NAME initialization parameter setting. If DB_UNIQUE_NAME is unspecified, then this option must match the DB_NAME initialization parameter setting. The default setting for DB_UNIQUE_NAME uses the setting for DB_NAME.
-s *service_name_list*	Comma-delimited list of database service names

Examples

The following example enables the database services sales and mfg in the database with DB_UNIQUE_NAME dbcrm:

```
srvctl enable service -d dbcrm -s "sales,mfg"
```

getenv

Gets and displays environment variables and their values from the Oracle Restart configuration for a database, listener, or Oracle ASM instance.

Table 4–32 srvctl getenv Summary

Command	Description
srvctl getenv asm on page 4-47	Displays the configured environment variables for the Oracle ASM instance
srvctl getenv database on page 4-47	Displays the configured environment variables for the specified database instance
srvctl getenv listener on page 4-48	Displays the configured environment variables for the specified listener

See Also:

- setenv command on page 4-56
- unsetenv command on page 4-69
- "Managing Environment Variables in the Oracle Restart Configuration" on page 4-17

srvctl getenv asm

Displays the configured environment variables for the Oracle ASM instance.

Syntax and Options

Use the `srvctl getenv asm` command with the following syntax:

```
srvctl getenv asm [-t name_list]
```

Table 4–33 srvctl getenv asm Options

Options	Description
`-t` *`name_list`*	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables for Oracle ASM.

Example

The following example displays all configured environment variables for the Oracle ASM instance:

```
srvctl getenv asm
```

srvctl getenv database

Displays the configured environment variables for the specified database.

Syntax and Options

Use the `srvctl getenv database` command with the following syntax:

```
srvctl getenv database -d db_unique_name [-t name_list]
```

Table 4–34 srvctl getenv database Options

Options	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-t name_list`	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables.

Example

The following example displays all configured environment variables for the database with `DB_UNIQUE_NAME` `dbcrm`:

```
srvctl getenv database -d dbcrm
```

srvctl getenv listener

Displays the configured environment variables for the specified listener.

Syntax and Options

Use the `srvctl getenv listener` command with the following syntax:

```
srvctl getenv listener [-l listener_name] [-t name_list]
```

Table 4–35 srvctl getenv listener Options

Options	Description
`-l listener_name`	Listener name. If omitted, SRVCTL lists environment variables for all listeners.
`-t name_list`	Comma-delimited list of names of environment variables to display. If omitted, SRVCTL displays all configured environment variables.

Example

The following example displays all configured environment variables for the listener named `crmlistener`:

```
srvctl getenv listener -l crmlistener
```

modify

Modifies the Oracle Restart configuration of a component. The change takes effect when the component is next restarted.

To perform `srvctl modify` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–36 srvctl modify Summary

Command	Description
srvctl modify asm on page 4-49	Modifies the configuration for Oracle ASM
srvctl modify database on page 4-49	Modifies the configuration for a database
srvctl modify listener on page 4-50	Modifies the configuration for the specified listener or all listeners
srvctl modify ons on page 4-51	Modifies ONS
srvctl modify service on page 4-51	Modifies the configuration for a database service

srvctl modify asm

Modifies the Oracle Restart configuration for the Oracle ASM instance.

Syntax and Options

Use the `srvctl modify asm` command with the following syntax:

```
srvctl modify asm [-l listener_name] [-p spfile] [-d asm_diskstring]
```

Table 4–37 srvctl modify asm Options

Option	Description
`-l listener_name`	Name of the listener with which Oracle ASM must register. A weak dependency is established with this listener. (Before Oracle ASM is started, Oracle Restart ensures that this listener is started.)
`-p spfile`	The full path of the server parameter file for the database. If omitted, the default SPFILE is used.
`-d asm_diskstring`	Oracle ASM disk group discovery string. An Oracle ASM discovery string is a comma-delimited list of strings that limits the set of disks that an Oracle ASM instance discovers. The discovery strings can include wildcard characters. Only disks that match one of the strings are discovered.

Example

An example of this command is:

```
srvctl modify asm -l crmlistener
```

See Also: *Oracle Automatic Storage Management Administrator's Guide* for more information about Oracle ASM disk group discovery strings

srvctl modify database

Modifies the Oracle Restart configuration for a database.

Syntax and Options

Use the `srvctl modify database` command with the following syntax:

```
srvctl modify database -d db_unique_name [-o oracle_home] [-u oracle_user]
  [-m domain_name] [-n db_name] [-p spfile] [-s start_options]
  [-t stop_options] [-r {PRIMARY | PHYSICAL_STANDBY | LOGICAL_STANDBY |
  SNAPSHOT_STANDBY}] [-y {automatic | manual}] [-a disk_group_list] [-z]
```

Table 4–38 srvctl modify database Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-u oracle_user`	Name of the Oracle user who owns the Oracle home directory
`-z`	Remove the database's dependency on Oracle ASM disk groups
(Other options)	See Table 4–11 on page 4-33

Example

The following example changes the role of the database with `DB_UNIQUE_NAME` `dbcrm` to `LOGICAL_STANDBY`:

```
srvctl modify database -d dbcrm -r logical_standby
```

See Also:

- "Oracle Restart Integration with Oracle Data Guard" on page 4-5
- *Oracle Data Guard Concepts and Administration*

srvctl modify listener

Modifies the Oracle Restart configuration for the specified listener or all listeners.

Syntax and Options

Use the `srvctl modify listener` command with the following syntax:

```
srvctl modify listener [-l listener_name] [-p endpoints] [-o Oracle_home]
```

Table 4–39 srvctl modify listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, all listener configurations are modified.
`-p endpoints`	Comma separated TCP ports or listener endpoints. *endpoints* syntax is: `"[TCP:]port[, ...] [/IPC:key] [/NMP:pipe_name] [/TCPS:s_port] [/SDP:port]"`
`-o Oracle_home`	New Oracle home for the listener

Example

This example modifies the TCP port on which the listener named `crmlistener` listens:

```
srvctl modify listener -l crmlistener -p TCP:1522
```

srvctl modify ons

Modifies ONS.

Syntax and Options

Use the `srvctl modify ons` command with the following syntax:

```
srvctl modify ons [-e em_port] [-l ons_local_port] [-r ons_remote_port] [-t
host[:port],[host[:port]...]] [-v]
```

Table 4–40 srvctl modify ons Options

Option	Description
`-e em_port`	ONS listening port for Oracle Enterprise Manager. The default is 2016.
`-l ons_local_port`	ONS listening port for local client connections
`-r ons_remote_port`	ONS listening port for connections from remote hosts
`-t host[:port],[host[:port]],...`	A list of `host:port` pairs of remote hosts that are part of the ONS network **Note:** If `port` is not specified for a remote host, then `ons_remote_port` is used.
`-v`	Verbose output

srvctl modify service

Modifies the Oracle Restart configuration of a database service.

> **Important:** Oracle recommends that you limit configuration changes to the minimum requirement and that you not perform other service operations while the online service modification is in progress.

Syntax and Options

Use the `srvctl modify service` command with the following syntax:

```
srvctl modify service -d db_unique_name -s service_name]
  [-l [PRIMARY][,PHYSICAL_STANDBY][,LOGICAL_STANDBY][,SNAPSHOT_STANDBY]]
  [-y {AUTOMATIC | MANUAL}] [-e {NONE | SESSION | SELECT}] [-m {NONE | BASIC}]
  [-w integer] [-z integer] [-j {SHORT | LONG}]
  [-B {SERVICE_TIME | THROUGHPUT | NONE}] [-q {TRUE | FALSE}] -t edition_name
```

Table 4–41 srvctl modify service Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s service_name`	Service name

Table 4–41 (Cont.) srvctl modify service Options

Option	Description
`-t` *`edition_name`*	The initial session edition of the service If this option is not specified, then the edition is not modified for the service. If this option is specified but *`edition_name`* is empty, then the edition is set to `NULL`. A `NULL` edition has no effect. When an edition is specified for a service, all subsequent connections that specify the service use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the initial session edition. SRVCTL does not validate the specified edition name. During connection, the connect user must have `USE` privilege on the specified edition. If the edition does not exist or if the connect user does not have `USE` privilege on the specified edition, then an error is raised.
(other options)	See Table 4–14 on page 4-36

Example

For the database with a `DB_UNIQUE_NAME` of `dbcrm`, the following command changes the Oracle Data Guard role of the database service named `support` to `standby`:

```
srvctl modify service -d dbcrm -s support -l standby
```

remove

Removes the specified component from the Oracle Restart configuration. Oracle Restart no longer manages the component. Any environment variable settings for the component are also removed.

Before you remove a component from the Oracle Restart configuration, you must use SRVCTL to stop it. Oracle recommends that you disable the component before removing it, but this is not required.

To perform `srvctl remove` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–42 srvctl remove Summary

Command	Description
srvctl remove asm on page 4-53	Removes the Oracle ASM instance
srvctl remove database on page 4-53	Removes a database
srvctl remove diskgroup on page 4-54	Removes an Oracle ASM disk group
srvctl remove listener on page 4-54	Removes a listener
srvctl remove ons on page 4-55	Removes an ONS
srvctl remove service on page 4-55	Removes one or more database services

See Also:

- stop command on page 4-65
- disable command on page 4-41

srvctl remove asm

Removes an Oracle ASM instance.

Syntax and Options

Use the `srvctl remove asm` command with the following syntax:

```
srvctl remove asm  [-f]
```

Table 4–43 srvctl remove asm Options

Options	Description
`-f`	Force remove, even when disk groups and databases that use Oracle ASM exist or when the Oracle ASM instance is running.

Example

An example of this command is:

```
srvctl remove asm
```

srvctl remove database

Removes a database. Prompts for confirmation first.

Syntax and Options

Use the `srvctl remove database` command with the following syntax:

```
srvctl remove database -d db_unique_name [-f] [-y] [-v]
```

Table 4–44 srvctl remove database Options

Options	Description
-d *db_unique_name*	Unique name for the database. Must match the DB_UNIQUE_NAME initialization parameter setting. If DB_UNIQUE_NAME is unspecified, then this option must match the DB_NAME initialization parameter setting. The default setting for DB_UNIQUE_NAME uses the setting for DB_NAME.
-f	Force. Removes the database even if it is running.
-y	Suppresses the confirmation prompt and removes immediately
-v	Verbose output. A success or failure message is displayed.

Example

An example of this command is:

```
srvctl remove database -d dbcrm
```

srvctl remove diskgroup

Removes an Oracle ASM disk group.

Syntax and Options

Use the srvctl remove diskgroup command with the following syntax:

```
srvctl remove diskgroup -g diskgroup_name [-f]
```

Table 4–45 srvctl remove diskgroup Options

Option	Description
-g *diskgroup_name*	Disk group name
-f	Force. Removes the disk group even if files are open on it.

Examples

This example removes the disk group named DATA. An error is returned if files are open on this disk group.

```
srvctl remove diskgroup -g DATA
```

srvctl remove listener

Removes the specified listener or all listeners.

Syntax and Options

Use the srvctl remove listener command with the following syntax:

```
srvctl remove listener [-l listener_name] [-a] [-f]
```

Table 4–46 srvctl remove listener Options

Options	Description
-l *listener_name*	Name of the listener that you want to remove. If omitted, then the default is LISTENER.
-a	Remove all listeners
-f	Force. Removes the listener even if databases are using it.

Example

The following command removes the listener `lsnr01`:

```
srvctl remove listener -l lsnr01
```

srvctl remove ons

Removes ONS.

Syntax and Options

Use the `srvctl remove ons` command as follows:

```
srvctl remove ons -f -v
```

Table 4–47 srvctl remove ons Options

Options	Description
`-f`	Force. Removes ONS even if it is enabled.
`-v`	Verbose output

srvctl remove service

Removes the specified database service.

Syntax and Options

Use the `srvctl remove service` command as follows:

```
srvctl remove service -d db_unique_name -s service_name] [-f]
```

Table 4–48 srvctl remove service Options

Options	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s` *`service_name`*	Service name
`-f`	Force. Removes the service even if the service is running. Transactions in any active sessions that are connected to the service are rolled back and sessions are disconnected.

Example

An example of this command is:

```
srvctl remove service -d dbcrm -s sales
```

setenv

The `setenv` command sets values of environment variables in the Oracle Restart configuration for a database, a listener, or the Oracle ASM instance.

To perform `srvctl setenv` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–49 srvctl setenv and unsetenv Summary

Command	Description
srvctl setenv asm on page 4-56	Sets environment variables in the Oracle Restart configuration for an Oracle ASM instance
srvctl setenv database on page 4-56	Sets environment variables in the Oracle Restart configuration for a database instance
srvctl setenv listener on page 4-57	Sets environment variables in the Oracle Restart configuration for the specified listener or all listeners

See Also:

- getenv command on page 4-47
- unsetenv command on page 4-69
- "Managing Environment Variables in the Oracle Restart Configuration" on page 4-17

srvctl setenv asm

Sets the values of environment variables in the Oracle Restart configuration for the Oracle ASM instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.

Syntax and Options

Use the `srvctl setenv asm` command with the following syntax:

```
srvctl setenv asm {-t name=val[,name=val,...] | -T name=val}
```

Table 4–50 srvctl setenv database Options

Options	Description
`-t name=val[,name=val,...]`	Comma-delimited list of name/value pairs of environment variables
`-T name=val`	Enables single environment variable to be set to a value that contains commas or other special characters

Example

The following example sets the AIX operating system environment variable `AIXTHREAD_SCOPE` in the Oracle ASM instance configuration:

```
srvctl setenv asm -t AIXTHREAD_SCOPE=S
```

srvctl setenv database

Sets the values of environment variables in the Oracle Restart configuration for a database instance. Before starting the instance, Oracle Restart sets environment variables to the values stored in the configuration.

Syntax and Options

Use the `srvctl setenv database` command with the following syntax:

```
srvctl setenv database -d db_unique_name {-t name=val[,name=val,...] |
  -T name=val}
```

Table 4–51 srvctl setenv database Options

Options	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-t name=val[,name=val,...]`	Comma-delimited list of name/value pairs of environment variables
`-T name=val`	Enables single environment variable to be set to a value that contains commas or other special characters

Example

The following example sets the `LANG` environment variable in the configuration of the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl setenv database -d dbcrm -t LANG=en
```

srvctl setenv listener

Sets the values of environment variables in the Oracle Restart configuration for a listener. Before starting the listener, Oracle Restart sets environment variables to the values stored in the configuration.

Syntax and Options

Use the `srvctl setenv listener` command with the following syntax:

```
srvctl setenv listener [-l listener_name] {-t name=val[,name=val,...] |
  -T name=val}
```

Table 4–52 srvctl setenv listener Options

Options	Description
`-l listener_name`	Listener name. If omitted, sets the specified environment variables in all listener configurations.
`-t name=val[,name=val,...]`	Comma-delimited list of name/value pairs of environment variables
`-T name=val`	Enables single environment variable to be set to a value that contains commas or other special characters

Example

The following example sets the AIX operating system environment variable `AIXTHREAD_SCOPE` in the configuration of the listener named `crmlistener`:

```
srvctl setenv listener -l crmlistener -t AIXTHREAD_SCOPE=S
```

start

Starts the specified component or components.

Table 4–53 srvctl start Summary

Command	Description
srvctl start asm on page 4-58	Starts the Oracle ASM instance
srvctl start database on page 4-58	Starts the specified database
srvctl start diskgroup on page 4-59	Starts (mounts) the specified Oracle ASM disk group
srvctl start home on page 4-59	Starts all of the components managed by Oracle Restart in the specified Oracle home
srvctl start listener on page 4-60	Starts the specified listener or all Oracle Restart–managed listeners
srvctl start ons on page 4-60	Starts ONS
srvctl start service on page 4-60	Starts the specified database service or services

See Also: "Starting and Stopping Components Managed by Oracle Restart" on page 4-25

srvctl start asm

Starts the Oracle ASM instance.

For this command, SRVCTL connects "/ as sysasm" to perform the operation. To run such operations, the owner of the executables in the Oracle Grid Infrastructure home must be a member of the OSASM group, and users running the commands must also be in the OSASM group.

Syntax and Options

Use the `srvctl start asm` command with the following syntax:

```
srvctl start asm [-o start_options]
```

Table 4–54 srvctl start asm Option

Option	Description
`-o start_options`	Comma-delimited list of options for the startup command (`OPEN`, `MOUNT`, `NOMOUNT`, or `FORCE`). If omitted, defaults to normal startup (`OPEN`).

Examples

This example starts the Oracle ASM instance, which then mounts any disk groups named in the `ASM_DISKGROUPS` initialization parameter:

```
srvctl start asm
```

This example starts the Oracle ASM instance without mounting any disk groups:

```
srvctl start asm -o nomount
```

srvctl start database

Starts the specified database instance.

For this command, SRVCTL connects "/ as sysdba" to perform the operation. To run such operations, the owner of the Oracle executables in the database Oracle home must be a member of the OSDBA group (for example, the `dba` group on UNIX and Linux), and users running the commands must also be in the OSDBA group.

Syntax and Options

Use the `srvctl start database` command with the following syntax:

```
srvctl start database -d db_unique_name [-o start_options]
```

Table 4–55 srvctl start database Options

Option	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-o` *`start_options`*	Comma-delimited list of options for the startup command (for example: `OPEN`, `MOUNT`, `NOMOUNT`, `RESTRICT`, `PFILE=`*path*, and so on)

Example

An example of this command is:

```
srvctl start database -d dbcrm -o pfile=testparm,nomount
```

srvctl start diskgroup

Starts (mounts) an Oracle ASM disk group.

Syntax and Options

Use the `srvctl start diskgroup` command with the following syntax:

```
srvctl start diskgroup -g diskgroup_name
```

Table 4–56 srvctl start diskgroup Options

Option	Description
`-g` *`diskgroup_name`*	Disk group name

Example

An example of this command is:

```
srvctl start diskgroup -g DATA
```

srvctl start home

Starts all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command starts the components that were stopped by a `srvctl stop home`. This command uses the information in the specified state file to identify the components to start.

> **Note:** Use this command to restart components after you install a patch in an Oracle home.

Syntax and Options

Use the `srvctl start home` command with the following syntax:

```
srvctl start home -o oracle_home -s state_file
```

Table 4–57 srvctl start home Options

Option	Description
-o	Complete path of the Oracle home
-s	Complete path of the state file. The state file contains the current state information for the components in the Oracle home and is created when the `srvctl stop home` command or the `srvctl status home` command is run.

srvctl start listener

Starts the specified listener or all listeners.

Syntax and Options

Use the `srvctl start listener` command with the following syntax:

```
srvctl start listener [-l listener_name]
```

Table 4–58 srvctl start listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, all Oracle Restart–managed listeners are started.

Example

An example of this command is:

```
srvctl start listener -l listener
```

srvctl start ons

Starts ONS.

Syntax and Options

Use the `srvctl start ons` command with the following syntax:

```
srvctl start ons -v
```

Table 4–59 srvctl start ons Options

Option	Description
-v	Verbose output

srvctl start service

Starts the specified database service or services.

Syntax and Options

Use the `srvctl start service` command with the following syntax:

```
srvctl start service -d db_unique_name [-s service_name_list] [-o start_options]
```

Table 4–60 srvctl start service Options

Option	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s` *`service_name_list`*	Comma-delimited list of service names. The service name list is optional and if not provided, SRVCTL starts all of the database's services
`-o` *`start_options`*	Options for database startup (for example: `OPEN`, `MOUNT`, `NOMOUNT` and so on) if the database must be started first

Example

For the database with a `DB_UNIQUE_NAME` of `dbcrm`, the following example starts the `sales` database service:

```
srvctl start service -d dbcrm -s sales
```

status

Displays the running status of the specified component or set of components.

Table 4–61 srvctl status Summary

Command	Description
srvctl status asm on page 4-62	Displays the running status of the Oracle ASM instance
srvctl status database on page 4-62	Displays the running status of a database
srvctl status diskgroup on page 4-63	Displays the running status of an Oracle ASM disk group
srvctl status home on page 4-63	Displays the running status of all of the components that are managed by Oracle Restart in the specified Oracle home
srvctl status listener on page 4-64	Displays the running status of the specified listener or all Oracle Restart–managed listeners
srvctl status ons on page 4-64	Displays the running status of ONS
srvctl status service on page 4-64	Displays the running status of one or more services

srvctl status asm

Displays the running status of the Oracle ASM instance.

Syntax and Options

Use the `srvctl status asm` command with the following syntax:

```
srvctl status asm [-a]
```

Table 4–62 srvctl status asm Options

Option	Description
`-a`	Display enabled/disabled status also

Example

An example of this command is:

```
srvctl status asm

ASM is running on dbhost
```

srvctl status database

Displays the running status of the specified database.

Syntax and Options

Use the `srvctl status database` command with the following syntax:

srvctl status database -d *db_unique_name* [-f] [-v]

Table 4–63 srvctl status database Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-f`	Display a message if the database is disabled
`-v`	Verbose output. Lists the database services that are running.

Example

An example of this command is:

```
srvctl status database -d dbcrm -v

Database dbcrm is running with online services mfg,sales
```

srvctl status diskgroup

Displays the running status of an Oracle ASM disk group.

Syntax and Options

Use the `srvctl status diskgroup` command with the following syntax:

```
srvctl status diskgroup -g diskgroup_name [-a]
```

Table 4–64 srvctl status diskgroup Options

Option	Description
`-g diskgroup_name`	Disk group name
`-a`	Display enabled/disabled status also

Example

An example of this command is:

```
srvctl status diskgroup -g DATA

Disk Group DATA is running on dbhost
```

srvctl status home

Displays the running status of all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command writes the current status of the components to the specified state file.

Syntax and Options

Use the `srvctl status home` command with the following syntax:

```
srvctl status home -o oracle_home -s state_file
```

Table 4–65 srvctl status home Options

Option	Description
-o	Complete path of the Oracle home
-s	Complete path of the state file

srvctl status listener

Displays the running status of the specified listener or of all Oracle Restart–managed listeners.

Syntax and Options

Use the `srvctl status listener` command with the following syntax:

```
srvctl status listener -l listener_name
```

Table 4–66 srvctl status listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, the status of all listeners is displayed.

Example

An example of this command is:

```
srvctl status listener -l crmlistener

Listener CRMLISTENER is running on dbhost
```

srvctl status ons

Displays the running status of ONS.

Syntax and Options

Use the `srvctl status ons` command with the following syntax:

```
srvctl status ons
```

srvctl status service

Displays the running status of one or more database services.

Syntax and Options

Use the `srvctl status service` command with the following syntax:

```
srvctl status service -d db_unique_name [-s service_name_list] [-f] [-v]
```

Table 4–67 srvctl status service Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s service_name_list`	Comma-delimited list of service names. If omitted, status is listed for all database services for the designated database.
`-f`	Display a message if a service is disabled
`-v`	Verbose output

Example

For the database with the `DB_UNIQUE_NAME` of `dbcrm`, the following example displays the running status of the service `sales`:

```
srvctl status service -d dbcrm -s sales

Service sales is running on dbhost
```

stop

Stops the specified component or components.

If you want a component to remain stopped after you issue a `srvctl stop` command, disable the component. See the disable command on page 4-41.

> **Note:** If a component is stopped and is not disabled, it could restart as a result of another planned operation. That is, although a stopped component will not restart as a result of a failure, it might be started if a dependent component is started with a `srvctl start` command.

Table 4–68 srvctl stop Summary

Command	Description
srvctl stop asm on page 4-65	Stops the Oracle ASM instance
srvctl stop database on page 4-66	Stops the specified database instance
srvctl stop diskgroup on page 4-66	Stops (dismounts) the specified Oracle ASM disk group
srvctl stop home on page 4-66	Stops all of the components managed by Oracle Restart in the specified Oracle home
srvctl stop listener on page 4-67	Stops the specified listener or all listeners
srvctl stop ons on page 4-67	Stops ONS
srvctl stop service on page 4-68	Stops the specified database service or services

> **See Also:** "Starting and Stopping Components Managed by Oracle Restart" on page 4-25

srvctl stop asm

Stops the Oracle ASM instance.

Syntax and Options

Use the `srvctl stop asm` command with the following syntax:

```
srvctl stop asm [-o stop_options] [-f]
```

Table 4–69 srvctl stop asm Option

Option	Description
`-o` *`stop_options`*	Options for the shutdown operation, for example, `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT`
`-f`	Force. Must be present if disk groups are currently started (mounted). This option enables SRVCTL to stop the disk groups before stopping Oracle ASM. Each dependent database instance is also stopped according to its stop options, or with the `ABORT` option if the configured stop options fail.

Example

An example of this command is:

```
srvctl stop asm -o abort -f
```

srvctl stop database

Stops a database.

Syntax and Options

Use the `srvctl stop database` command with the following syntax:

```
srvctl stop database -d db_unique_name [-o stop_options] [-f]
```

Table 4–70 srvctl stop database Options

Option	Description
`-d db_unique_name`	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-o stop_options`	`SHUTDOWN` command options (for example: `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT`). Default is `IMMEDIATE`.
`-f`	Force. Performs a `SHUTDOWN ABORT` of the database.

Example

An example of this command is:

```
srvctl stop database -d dbcrm
```

srvctl stop diskgroup

Stops (dismounts) an Oracle ASM disk group.

Syntax and Options

Use the `srvctl stop diskgroup` command with the following syntax:

```
srvctl stop diskgroup -g diskgroup_name [-f]
```

Table 4–71 srvctl stop diskgroup Options

Option	Description
`-g diskgroup_name`	Disk group name
`-f`	Force. Dismount the disk group even if some files in the disk group are open.

Examples

This example stops the disk group named `DATA`. An error is returned if files are open on this disk group.

```
srvctl stop diskgroup -g DATA
```

srvctl stop home

Stops all of the components that are managed by Oracle Restart in the specified Oracle home. The Oracle home can be an Oracle Database home or an Oracle Grid Infrastructure home.

This command identifies the components that it stopped in the specified state file.

Note:

- Before stopping the components in an Oracle Grid Infrastructure home, stop the components in a dependent Oracle Database home.
- Use this command to stop components before you install a patch in an Oracle home.

Syntax and Options

Use the `srvctl stop home` command with the following syntax:

```
srvctl stop home -o oracle_home -s state_file [-t stop_options] [-f]
```

Table 4–72 srvctl stop home Options

Option	Description
-o	Complete path of the Oracle home
-s	Complete path of the state file
`-t stop_options`	`SHUTDOWN` command options for the database (for example: `NORMAL`, `TRANSACTIONAL`, `IMMEDIATE`, or `ABORT`). Default is `IMMEDIATE`.
`-f`	Force stop each component

srvctl stop listener

Stops the designated listener or all Oracle Restart–managed listeners. Stopping a listener does not cause databases that are registered with the listener to be stopped.

Syntax and Options

Use the `srvctl stop listener` command with the following syntax:

```
srvctl stop listener [-l listener_name] [-f]
```

Table 4–73 srvctl stop listener Options

Option	Description
`-l listener_name`	Listener name. If omitted, all Oracle Restart–managed listeners are stopped.
`-f`	Force. Passes the stop command with the `-f` option to Oracle Clusterware. See *Oracle Clusterware Administration and Deployment Guide* for more information about the Oracle Clusterware `-f` option.

Example

An example of this command is:

```
srvctl stop listener -l crmlistener
```

srvctl stop ons

Stops ONS.

Syntax and Options

Use the `srvctl stop ons` command with the following syntax:

```
srvctl stop ons -v
```

Table 4–74 srvctl stop ons Options

Option	Description
`-v`	Verbose output

srvctl stop service

Stops one or more database services.

Syntax and Options

Use the `srvctl stop service` command with the following syntax:

```
srvctl stop service -d db_unique_name [-s service_name_list] [-f]
```

Table 4–75 srvctl stop service Options

Option	Description
`-d` *`db_unique_name`*	Unique name for the database. Must match the `DB_UNIQUE_NAME` initialization parameter setting. If `DB_UNIQUE_NAME` is unspecified, then this option must match the `DB_NAME` initialization parameter setting. The default setting for `DB_UNIQUE_NAME` uses the setting for `DB_NAME`.
`-s` *`service_name_list`*	Comma-delimited list of database service names. If you do not provide a service name list, then SRVCTL stops all services on the database
`-f`	Force. This option disconnects all of the stopped services' sessions immediately. Uncommitted transactions are rolled back. If this option is omitted, active sessions remain connected to the services, but no further connections to the services can be made.

Examples

The following example stops the `sales` database service on the database with a `DB_UNIQUE_NAME` of `dbcrm`:

```
srvctl stop service -d dbcrm -s sales
```

unsetenv

The `unsetenv` command deletes one or more environment variables from the Oracle Restart configuration for a database, a listener, or an Oracle ASM instance.

To perform `srvctl unsetenv` operations, you must be logged in to the database host computer with the proper user account. See "Preparing to Run SRVCTL" on page 4-10 for more information.

Table 4–76 srvctl unsetenv Command Summary

Command	Description
srvctl unsetenv asm on page 4-69	Removes the specified environment variables from the Oracle Restart configuration for the Oracle ASM instance
srvctl unsetenv database on page 4-69	Removes the specified environment variables from the Oracle Restart configuration for a database
srvctl unsetenv listener on page 4-70	Removes the specified environment variables from the Oracle Restart configuration for a listener or all listeners

See Also:

- setenv command on page 4-56
- getenv command on page 4-47
- "Managing Environment Variables in the Oracle Restart Configuration" on page 4-17

srvctl unsetenv asm

Removes the specified environment variables from the Oracle Restart configuration for the Oracle ASM instance.

Syntax and Options

Use the `srvctl unsetenv asm` command with the following syntax:

```
srvctl unsetenv asm -t name_list
```

Table 4–77 srvctl unsetenv asm Options

Options	Description
`-t` *`name_list`*	Comma-delimited list of environment variables to remove

Example

The following example removes the AIX operating system environment variable `AIXTHREAD_SCOPE` from the Oracle ASM instance configuration:

```
srvctl unsetenv asm -t AIXTHREAD_SCOPE
```

srvctl unsetenv database

Removes the specified environment variables from the Oracle Restart configuration for the specified database.

Syntax and Options

Use the `srvctl unsetenv database` command as follows:

```
srvctl unsetenv database -d db_unique_name -t name_list
```

Table 4–78 srvctl unsetenv database Options

Options	Description
-d *db_unique_name*	Unique name for the database. Must match the DB_UNIQUE_NAME initialization parameter setting. If DB_UNIQUE_NAME is unspecified, then this option must match the DB_NAME initialization parameter setting. The default setting for DB_UNIQUE_NAME uses the setting for DB_NAME.
-t *name_list*	Comma-delimited list of environment variables to remove

Example

The following example deletes the AIXTHREAD_SCOPE environment variable from the Oracle Restart configuration for the database with a DB_UNIQUE_NAME of dbcrm:

```
srvctl unsetenv database -d dbcrm -t AIXTHREAD_SCOPE
```

srvctl unsetenv listener

Removes the specified environment variables from the Oracle Restart configuration for the specified listener or all listeners.

Syntax and Options

Use the srvctl unsetenv listener command with the following syntax:

```
srvctl unsetenv listener [-l listener_name] -t name_list
```

Table 4–79 srvctl unsetenv listener Options

Options	Description
-l *listener_name*	Listener name. If omitted, the specified environment variables are removed from the configurations of all listeners.
-t *name_list*	Comma-delimited list of environment variables to remove

Example

The following example removes the AIX operating system environment variable AIXTHREAD_SCOPE from the listener configuration for the listener named crmlistener:

```
srvctl unsetenv listener -l crmlistener -t AIXTHREAD_SCOPE
```

CRSCTL Command Reference

This section provides details about the syntax for the CRSCTL commands that are relevant for Oracle Restart.

> **Note:** You must be the root user or Oracle grid infrastructure software owner to run these CRSCTL commands.

CRSCTL Command Syntax Overview

CRSCTL expects the following command syntax:

```
crsctl command has
```

where *command* is a verb such as `start`, `stop`, or `enable`. The `has` object indicates Oracle high availability services. See Table 4–80 on page 4-71 for a complete list of commands.

Case Sensitivity

CRSCTL commands and components are case insensitive.

Table 4–80 Summary of CRSCTL Commands

Command	Description
check on page 4-72	Displays the Oracle Restart status.
config on page 4-73	Displays the Oracle Restart configuration.
disable on page 4-74	Disables automatic restart of Oracle Restart.
enable on page 4-75	Enables automatic restart of Oracle Restart.
start on page 4-76	Starts Oracle Restart.
stop on page 4-77	Stops Oracle Restart.

check

Displays the Oracle Restart status.

Syntax and Options

```
crsctl check has
```

config

Displays the Oracle Restart configuration.

Syntax and Options

```
crsctl config has
```

disable

Disables automatic restart of Oracle Restart.

Syntax and Options

```
crsctl disable has
```

enable

Enables automatic restart of Oracle Restart.

Syntax and Options

```
crsctl enable has
```

start

Starts Oracle Restart.

Syntax and Options

```
crsctl start has
```

stop

Stops Oracle Restart.

Syntax and Options

```
crsctl stop has [-f]
```

Table 4–81 crsctl stop has Options

Options	Description
`-f`	Force. If any resources that are managed by Oracle Restart are still running, then try to stop these resources gracefully. If a resource cannot be stopped gracefully, then try to force the resource to stop. For example, if an Oracle ASM instance is running, then `SHUTDOWN IMMEDIATE` attempts to stop the Oracle ASM instance gracefully, while `SHUTDOWN ABORT` attempts to force the Oracle ASM instance to stop. When the `-f` option *is not* specified, this command tries to stop resources managed by Oracle Restart gracefully but does not try to force them to stop.

5

Managing Processes

In this chapter:

- About Dedicated and Shared Server Processes
- About Database Resident Connection Pooling
- Configuring Oracle Database for Shared Server
- Configuring Database Resident Connection Pooling
- About Oracle Database Background Processes
- Managing Processes for Parallel SQL Execution
- Managing Processes for External Procedures
- Terminating Sessions
- Process and Session Data Dictionary Views

About Dedicated and Shared Server Processes

Oracle Database creates server processes to handle the requests of user processes connected to an instance. A server process can be either of the following:

- A **dedicated server process**, which services only one user process
- A **shared server process**, which can service multiple user processes

Your database is always enabled to allow dedicated server processes, but you must specifically configure and enable shared server by setting one or more initialization parameters.

Dedicated Server Processes

Figure 5–1, "Oracle Database Dedicated Server Processes" illustrates how dedicated server processes work. In this diagram two user processes are connected to the database 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 Database 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:

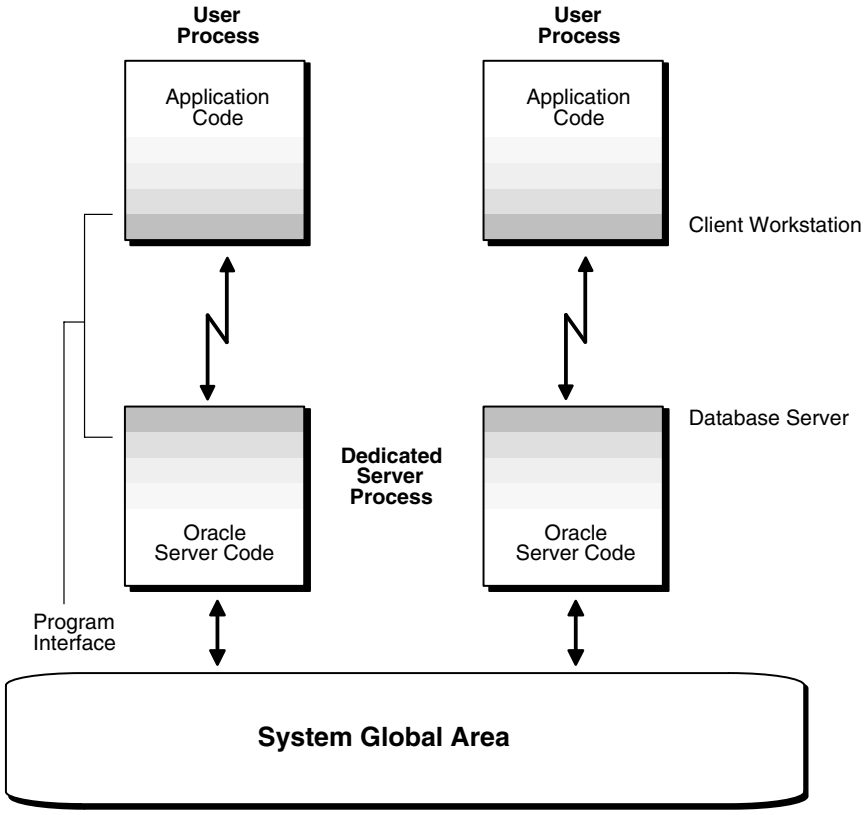
- To submit a batch job (for example, when a job can allow little or no idle time for the server process)

- To use Recovery Manager (RMAN) to back up, restore, or recover a database

To request a dedicated server connection when Oracle Database 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:** *Oracle Database Net Services Administrator's Guide* for more information about requesting a dedicated server connection

Figure 5–1 Oracle Database Dedicated Server Processes

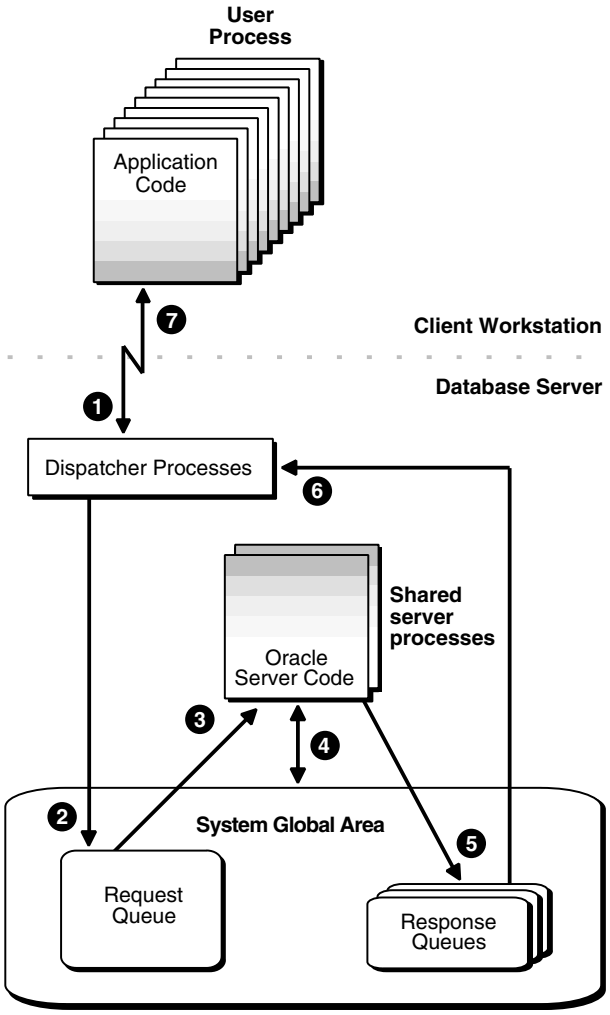


Shared Server Processes

Consider an order entry system with dedicated server processes. A customer phones the order desk and places an order, and the clerk taking the call enters the order into the database. For most of the transaction, the clerk is on the telephone talking to the customer. A server process is not needed during this time, so the server process dedicated to the clerk's user process remains idle. The system is slower for other clerks entering orders, because the idle server process is holding system resources.

Shared server architecture eliminates the need for a dedicated server process for each connection (see Figure 5–2).

Figure 5–2 Oracle Database Shared Server Processes



In a shared server configuration, client user processes connect to a dispatcher. The dispatcher can support multiple client connections concurrently. Each client connection is bound to a **virtual circuit**, which 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 process 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.

For even better resource management, shared server can be configured for **connection pooling**. Connection pooling lets a dispatcher support more users by enabling the database server to time out protocol connections and to use those connections to service an active session. Further, shared server can be configured for **session multiplexing**, which combines multiple sessions for transmission over a single network connection in order to conserve the operating system's resources.

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 system as the Oracle Database instance.

> **See Also:** *Oracle Database Net Services Administrator's Guide* for more detailed information about shared server, including features such as connection pooling and session multiplexing

About Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides a connection pool in the database server for typical Web application usage scenarios where the application acquires a database connection, works on it for a relatively short duration, and then releases it. DRCP pools "dedicated" servers. A **pooled server** is the equivalent of a server foreground process and a database session combined.

DRCP complements middle-tier connection pools that share connections between threads in a middle-tier process. In addition, DRCP enables sharing of database connections across middle-tier processes on the same middle-tier host and even across middle-tier hosts. This results in significant reduction in key database resources needed to support a large number of client connections, thereby reducing the database tier memory footprint and boosting the scalability of both middle-tier and database tiers. Having a pool of readily available servers also has the additional benefit of reducing the cost of creating and tearing down client connections.

DRCP is especially relevant for architectures with multi-process single threaded application servers (such as PHP/Apache) that cannot perform middle-tier connection pooling. The database can still scale to tens of thousands of simultaneous connections with DRCP.

> **See Also:**
>
> - *Oracle Database Concepts* for more details on DRCP
> - *Oracle Call Interface Programmer's Guide* for information about options that are available when obtaining a DRCP session

When To Use Database Resident Connection Pooling

Database resident connection pooling is useful when multiple clients access the database and when any of the following apply:

- A large number of client connections need to be supported with minimum memory usage.
- The client applications are similar and can share or reuse sessions.

  Applications are similar if they connect with the same database credentials and use the same schema.
- The client applications acquire a database connection, work on it for a relatively short duration, and then release it.
- Session affinity is not required across client requests.
- There are multiple processes and multiple hosts on the client side.

Advantages of Database Resident Connection Pooling

Using database resident connection pooling provides the following advantages:

- Enables resource sharing among multiple middle-tier client applications.
- Improves scalability of databases and applications by reducing resource usage.

Database Resident Connection Pooling and LOGON/LOGOFF Triggers

`LOGON` triggers fire for every authentication and every time a new session is created in DRCP.

`LOGOFF` triggers fire on every log off and every time a session is released in DRCP. Therefore, a `LOGOFF` trigger fires when a session is terminated due to an idle time limit.

See Also:

- *Oracle Database PL/SQL Language Reference*
- *Oracle Database Security Guide*

Comparing DRCP to Dedicated Server and Shared Server

Table 5–1 lists the differences between dedicated server, shared server, and database resident connection pooling.

Table 5–1 Dedicated Servers, Shared Servers, and Database Resident Connection Pooling

Dedicated Server	Shared Server	Database Resident Connection Pooling
When a client request is received, a new server process and a session are created for the client.	When the first request is received from a client, the Dispatcher process places this request on a common queue. The request is picked up by an available shared server process. The Dispatcher process then manages the communication between the client and the shared server process.	When the first request is received from a client, the Connection Broker picks an available pooled server and hands off the client connection to the pooled server. If no pooled servers are available, the Connection Broker creates one. If the pool has reached its maximum size, the client request is placed on the wait queue until a pooled server is available.
Releasing database resources involves terminating the session and server process.	Releasing database resources involves terminating the session.	Releasing database resources involves releasing the pooled server to the pool.
Memory requirement is proportional to the number of server processes and sessions. There is one server and one session for each client.	Memory requirement is proportional to the sum of the shared servers and sessions. There is one session for each client.	Memory requirement is proportional to the number of pooled servers and their sessions. There is one session for each pooled server.
Session memory is allocated from the PGA.	Session memory is allocated from the SGA.	Session memory is allocated from the PGA.

Example of Memory Usage for Dedicated Server, Shared Server, and Database Resident Connection Pooling

Consider an application in which the memory required for each session is 400 KB and the memory required for each server process is 4 MB. The pool size is 100 and the number of shared servers used is 100.

If there are 5000 client connections, the memory used by each configuration is as follows:

- Dedicated Server

  Memory used = 5000 X (400 KB + 4 MB) = 22 GB

- Shared Server

  Memory used = 5000 X 400 KB + 100 X 4 MB = 2.5 GB

  Out of the 2.5 GB, 2 GB is allocated from the SGA.

- Database Resident Connection Pooling

  Memory used = 100 X (400 KB + 4 MB) + (5000 X 35KB)= 615 MB

Restrictions on Using Database Resident Connection Pooling

You cannot perform the following activities with pooled servers:

- Shut down the database
- Stop the database resident connection pool
- Change the password for the connected user
- Use shared database links to connect to a database resident connection pool
- Use Advanced Security Option (ASO) options such as encryption, certificates, and so on
- Use migratable sessions on the server side directly by using the OCI_MIGRATE option or indirectly through OCIConnectionPool

DDL statements that pertain to database users in the pool need to be performed carefully, as the pre-DDL sessions in the pool can still be given to clients post-DDL. For example, while dropping users, ensure that there are no sessions of that user in the pool and no connections to the Broker that were authenticated as that user.

Sessions with explicit roles enabled, that are released to the pool, can be later handed out to connections (of the same user) that need the default logon role. Avoid releasing sessions with explicit roles, and instead terminate them.

Configuring Oracle Database for Shared Server

This section discusses how to enable shared server and how to set or alter shared server initialization parameters. It contains the following topics:

- Initialization Parameters for Shared Server
- Memory Management for Shared Server
- Enabling Shared Server
- Configuring Dispatchers
- Disabling Shared Server
- Shared Server Data Dictionary Views

See Also:

- "About Dedicated and Shared Server Processes" on page 5-1
- *Oracle Database SQL Language Reference* for further information about the `ALTER SYSTEM` statement

Initialization Parameters for Shared Server

The following initialization parameters control shared server operation:

- SHARED_SERVERS: Specifies the initial number of shared servers to start and the minimum number of shared servers to keep. This is the only required parameter for using shared servers.
- MAX_SHARED_SERVERS: Specifies the maximum number of shared servers that can run simultaneously.
- SHARED_SERVER_SESSIONS: Specifies the total number of shared server user sessions that can run simultaneously. Setting this parameter enables you to reserve user sessions for dedicated servers.
- DISPATCHERS: Configures dispatcher processes in the shared server architecture.
- MAX_DISPATCHERS: Specifies the maximum number of dispatcher processes that can run simultaneously. This parameter can be ignored for now. It will only be useful in a future release when the number of dispatchers is auto-tuned according to the number of concurrent connections.
- CIRCUITS: Specifies the total number of virtual circuits that are available for inbound and outbound network sessions.

> **See Also:** *Oracle Database Reference* for more information about these initialization parameters

Memory Management for Shared Server

Shared server requires some user global area (UGA) in either the shared pool or large pool. For installations with a small number of simultaneous sessions, the default sizes for these system global area (SGA) components are generally sufficient. However, if you expect a large number of sessions for your installation, you may have to tune memory to support shared server.

See the "Configuring and Using Memory" section of *Oracle Database Performance Tuning Guide* for guidelines.

Enabling Shared Server

Shared server is enabled by setting the SHARED_SERVERS initialization parameter to a value greater than 0. The other shared server initialization parameters need not be set. Because shared server requires at least one dispatcher in order to work, a dispatcher is brought up even if no dispatcher has been configured. Dispatchers are discussed in "Configuring Dispatchers" on page 5-10.

Shared server can be started dynamically by setting the SHARED_SERVERS parameter to a nonzero value with the ALTER SYSTEM statement, or SHARED_SERVERS can be included at database startup in the initialization parameter file. If SHARED_SERVERS is not included in the initialization parameter file, or is included but is set to 0, then shared server is not enabled at database startup.

> **Note:** If SHARED_SERVERS is not included in the initialization parameter file at database startup, but DISPATCHERS is included and it specifies at least one dispatcher, shared server is enabled. In this case, the default for SHARED_SERVERS is 1.
>
> If neither SHARED_SERVERS nor DISPATCHERS is included in the initialization file, you cannot start shared server after the instance is brought up by just altering the DISPATCHERS parameter. You must specifically alter SHARED_SERVERS to a nonzero value to start shared server.

Note: If you create your Oracle database with Database Configuration Assistant (DBCA), DBCA configures a dispatcher for Oracle XML DB (XDB). This is because XDB protocols like HTTP and FTP require shared server. This results in a SHARED_SERVER value of 1. Although shared server is enabled, this configuration permits only sessions that connect to the XDB service to use shared server. To enable shared server for regular database sessions (for submitting SQL statements), you must add an additional dispatcher configuration, or replace the existing configuration with one that is not specific to XDB. See "Configuring Dispatchers" on page 5-10 for instructions.

Determining a Value for SHARED_SERVERS

The SHARED_SERVERS initialization parameter specifies the minimum number of shared servers that you want created when the instance is started. After instance startup, Oracle Database can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.

In typical systems, the number of shared servers stabilizes at a ratio of one shared server for every ten connections. For OLTP applications, when the rate of requests is low, or when the ratio of server usage to request is low, the connections-to-servers ratio could be higher. In contrast, 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.

The PMON (process monitor) background process cannot terminate shared servers below the value specified by SHARED_SERVERS. Therefore, you can use this parameter to stabilize the load and minimize strain on the system by preventing PMON from terminating and then restarting shared servers because of coincidental fluctuations in load.

If you know the average load on your system, you can set SHARED_SERVERS to an optimal value. The following example shows how you can use this parameter:

Assume a database is being used by a telemarketing center staffed by 1000 agents. On average, each agent spends 90% of the time talking to customers and only 10% of the time looking up and updating records. To keep the shared servers from being terminated as agents talk to customers and then spawned again as agents access the database, a DBA specifies that the optimal number of shared servers is 100.

However, not all work shifts are staffed at the same level. On the night shift, only 200 agents are needed. Since SHARED_SERVERS is a dynamic parameter, a DBA reduces the number of shared servers to 20 at night, thus allowing resources to be freed up for other tasks such as batch jobs.

Decreasing the Number of Shared Server Processes

You can decrease the minimum number of shared servers that must be kept active by dynamically setting the SHARED_SERVERS parameter to a lower value. Thereafter, until the number of shared servers is decreased to the value of the SHARED_SERVERS parameter, any shared servers that become inactive are marked by PMON for termination.

The following statement reduces the number of shared servers:

```
ALTER SYSTEM SET SHARED_SERVERS = 5;
```

Setting SHARED_SERVERS to 0 disables shared server. For more information, see "Disabling Shared Server" on page 5-14.

Limiting the Number of Shared Server Processes

The MAX_SHARED_SERVERS parameter specifies the maximum number of shared servers that can be automatically created by PMON. It has no default value. If no value is specified, then PMON starts as many shared servers as is required by the load, subject to these limitations:

- The process limit (set by the PROCESSES initialization parameter)
- A minimum number of free process slots (at least one-eighth of the total process slots, or two slots if PROCESSES is set to less than 24)
- System resources

> **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.

The value of SHARED_SERVERS overrides the value of MAX_SHARED_SERVERS. Therefore, you can force PMON to start more shared servers than the MAX_SHARED_SERVERS value by setting SHARED_SERVERS to a value higher than MAX_SHARED_SERVERS. You can subsequently place a new upper limit on the number of shared servers by dynamically altering the MAX_SHARED_SERVERS to a value higher than SHARED_SERVERS.

The primary reason to limit the number of shared servers is to reserve resources, such as memory and CPU time, for other processes. For example, consider the case of the telemarketing center discussed previously:

The DBA wants to reserve two thirds of the resources for batch jobs at night. He sets MAX_SHARED_SERVERS to less than one third of the maximum number of processes (PROCESSES). By doing so, the DBA ensures that even if all agents happen to access the database at the same time, batch jobs can connect to dedicated servers without having to wait for the shared servers to be brought down after processing agents' requests.

Another reason to limit the number of shared servers is to prevent the concurrent run of too many server processes from slowing down the system due to heavy swapping, although PROCESSES can serve as the upper bound for this rather than MAX_SHARED_SERVERS.

Still other reasons to limit the number of shared servers are testing, debugging, performance analysis, and tuning. For example, to see how many shared servers are needed to efficiently support a certain user community, you can vary MAX_SHARED_SERVERS from a very small number upward until no delay in response time is noticed by the users.

Limiting the Number of Shared Server Sessions

The SHARED_SERVER_SESSIONS initialization parameter specifies the maximum number of concurrent shared server user sessions. Setting this parameter, which is a dynamic parameter, lets you reserve database sessions for dedicated servers. This in turn ensures that administrative tasks that require dedicated servers, such as backing up or recovering the database, are not preempted by shared server sessions.

This parameter has no default value. If it is not specified, the system can create shared server sessions as needed, limited by the SESSIONS initialization parameter.

Protecting Shared Memory

The CIRCUITS parameter sets a maximum limit on the number of virtual circuits that can be created in shared memory. This parameter has no default. If it is not specified, then the system can create circuits as needed, limited by the DISPATCHERS initialization parameter and system resources.

Configuring Dispatchers

The DISPATCHERS initialization parameter configures dispatcher processes in the shared server architecture. At least one dispatcher process is required for shared server to work.If you do not specify a dispatcher, but you enable shared server by setting SHARED_SERVER to a nonzero value, then by default Oracle Database creates one dispatcher for the TCP protocol. The equivalent DISPATCHERS explicit setting of the initialization parameter for this configuration is:

```
dispatchers="(PROTOCOL=tcp)"
```

You can configure more dispatchers, using the DISPATCHERS initialization parameter, if either of the following conditions apply:

- You must configure a protocol other than TCP/IP. You configure a protocol address with one of the following attributes of the DISPATCHERS parameter:
  - ADDRESS
  - DESCRIPTION
  - PROTOCOL
- You want to configure one or more of the optional dispatcher attributes:
  - DISPATCHERS
  - CONNECTIONS
  - SESSIONS
  - TICKS
  - LISTENER
  - MULTIPLEX
  - POOL
  - SERVICE

Note: Database Configuration Assistant helps you configure this parameter.

DISPATCHERS Initialization Parameter Attributes

This section provides brief descriptions of the attributes that can be specified with the DISPATCHERS initialization parameter.

A protocol address is required and is specified using one or more of the following attributes:

Attribute	Description
ADDRESS	Specify the network protocol address of the endpoint on which the dispatchers listen.

Attribute	Description
DESCRIPTION	Specify the network description of the endpoint on which the dispatchers listen, including the network protocol address. The syntax is as follows: (DESCRIPTION=(ADDRESS=...))
PROTOCOL	Specify the network protocol for which the dispatcher generates a listening endpoint. For example: (PROTOCOL=tcp) See the *Oracle Database Net Services Reference* for further information about protocol address syntax.

The following attribute specifies how many dispatchers this configuration should have. It is optional and defaults to 1.

Attribute	Description
DISPATCHERS	Specify the initial number of dispatchers to start.

The following attributes tell the instance about the network attributes of each dispatcher of this configuration. They are all optional.

Attribute	Description
CONNECTIONS	Specify the maximum number of network connections to allow for each dispatcher.
SESSIONS	Specify the maximum number of network sessions to allow for each dispatcher.
TICKS	Specify the duration of a TICK in seconds. A TICK is a unit of time in terms of which the connection pool timeout can be specified. Used for connection pooling.
LISTENER	Specify an alias name for the listeners with which the PMON process registers dispatcher information. Set the alias to a name that is resolved through a naming method.
MULTIPLEX	Used to enable the Oracle Connection Manager session multiplexing feature.
POOL	Used to enable connection pooling.
SERVICE	Specify the service names the dispatchers register with the listeners.

You can specify either an entire attribute name a substring consisting of at least the first three characters. For example, you can specify SESSIONS=3, SES=3, SESS=3, or SESSI=3, and so forth.

> **See Also:** *Oracle Database Reference* for more detailed descriptions of the attributes of the DISPATCHERS initialization parameter

Determining the Number of Dispatchers

Once you know the number of possible connections for each process for the operating system, calculate the initial number of dispatchers to create during instance startup, for each network protocol, using the following formula:

```
Number of dispatchers =
```

```
CEIL ( max. concurrent sessions / connections for each dispatcher )
```

CEIL returns the result roundest up to the next whole integer.

For example, assume a system that can support 970 connections for each process, and that has:

- A maximum of 4000 sessions concurrently connected through TCP/IP and
- A maximum of 2,500 sessions concurrently connected through TCP/IP with SSL

The DISPATCHERS attribute for TCP/IP should be set to a minimum of five dispatchers (4000 / 970), and for TCP/IP with SSL three dispatchers (2500 / 970:

```
DISPATCHERS='(PROT=tcp)(DISP=5)', '(PROT-tcps)(DISP=3)'
```

Depending on performance, you may need to adjust the number of dispatchers.

Setting the Initial Number of Dispatchers

You can specify multiple dispatcher configurations by setting DISPATCHERS to a comma separated list of strings, or by specifying multiple DISPATCHERS parameters in the initialization file. If you specify DISPATCHERS multiple times, the lines must be adjacent to each other in the initialization parameter file. Internally, Oracle Database assigns an INDEX value (beginning with zero) to each DISPATCHERS parameter. You can later refer to that DISPATCHERS parameter in an ALTER SYSTEM statement by its index number.

Some examples of setting the DISPATCHERS initialization parameter follow.

Example: Typical This is a typical example of setting the DISPATCHERS initialization parameter.

```
DISPATCHERS="(PROTOCOL=TCP)(DISPATCHERS=2)"
```

Example: Forcing the IP Address Used for Dispatchers The following hypothetical example will create two dispatchers that will listen on the specified IP address. The address must be a valid IP address for the host that the instance is on. (The host may be configured with multiple IP addresses.)

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201))(DISPATCHERS=2)"
```

Example: Forcing the Port Used by Dispatchers To force the dispatchers to use a specific port as the listening endpoint, add the PORT attribute as follows:

```
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5000))"
DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(PORT=5001))"
```

Altering the Number of Dispatchers

You can control the number of dispatcher processes in the instance. Unlike the number of shared servers, the number of dispatchers does not change automatically. You change the number of dispatchers explicitly with the ALTER SYSTEM statement. In this release of Oracle Database, you can increase the number of dispatchers to more than the limit specified by the MAX_DISPATCHERS parameter. It is planned that MAX_DISPATCHERS will be taken into consideration in a future release.

Monitor the following views to determine the load on the dispatcher processes:

- V$QUEUE
- V$DISPATCHER

- V$DISPATCHER_RATE

> **See Also:** *Oracle Database Performance Tuning Guide* for information about monitoring these views to determine dispatcher load and performance

If these views indicate that the load on the dispatcher processes is consistently high, then performance may be improved by starting additional dispatcher processes to route user requests. In contrast, if the load on dispatchers is consistently low, reducing the number of dispatchers may improve performance.

To dynamically alter the number of dispatchers when the instance is running, use the ALTER SYSTEM statement to modify the DISPATCHERS attribute setting for an existing dispatcher configuration. You can also add new dispatcher configurations to start dispatchers with different network attributes.

When you reduce the number of dispatchers for a particular dispatcher configuration, the dispatchers are not immediately removed. Rather, as users disconnect, Oracle Database terminates dispatchers down to the limit you specify in DISPATCHERS,

For example, suppose the instance was started with this DISPATCHERS setting in the initialization parameter file:

```
DISPATCHERS='(PROT=tcp)(DISP=2)', '(PROT=tcps)(DISP=2)'
```

To increase the number of dispatchers for the TCP/IP protocol from 2 to 3, and decrease the number of dispatchers for the TCP/IP with SSL protocol from 2 to 1, you can issue the following statement:

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=0)(DISP=3)', '(INDEX=1)(DISP=1)';
```

or

```
ALTER SYSTEM SET DISPATCHERS = '(PROT=tcp)(DISP=3)', '(PROT-tcps)(DISP=1)';
```

> **Note:** You need not specify (DISP=1). It is optional because 1 is the default value for the DISPATCHERS parameter.

If fewer than three dispatcher processes currently exist for TCP/IP, the database creates new ones. If multiple dispatcher processes currently exist for TCP/IP with SSL, then the database terminates the extra ones as the connected users disconnect.

Suppose that instead of changing the number of dispatcher processes for the TCP/IP protocol, you want to add another TCP/IP dispatcher that supports connection pooling. You can do so by entering the following statement:

```
ALTER SYSTEM SET DISPATCHERS = '(INDEX=2)(PROT=tcp)(POOL=on)';
```

The INDEX attribute is needed to add the new dispatcher configuration. If you omit (INDEX=2) in the preceding statement, then the TCP/IP dispatcher configuration at INDEX 0 will be changed to support connection pooling, and the number of dispatchers for that configuration will be reduced to 1, which is the default when the number of dispatchers (attribute DISPATCHERS) is not specified.

Notes on Altering Dispatchers

- The INDEX keyword can be used to identify which dispatcher configuration to modify. If you do not specify INDEX, then the first dispatcher configuration matching the DESCRIPTION, ADDRESS, or PROTOCOL specified will be modified. If no

match is found among the existing dispatcher configurations, then a new dispatcher will be added.

- The INDEX value can range from 0 to *n*-1, where *n* is the current number of dispatcher configurations. If your ALTER SYSTEM statement specifies an INDEX value equal to *n*, where *n* is the current number of dispatcher configurations, a new dispatcher configuration will be added.
- To see the values of the current dispatcher configurations--that is, the number of dispatchers, whether connection pooling is on, and so forth--query the V$DISPATCHER_CONFIG dynamic performance view. To see which dispatcher configuration a dispatcher is associated with, query the CONF_INDX column of the V$DISPATCHER view.
- When you change the DESCRIPTION, ADDRESS, PROTOCOL, CONNECTIONS, TICKS, MULTIPLEX, and POOL attributes of a dispatcher configuration, the change does not take effect for existing dispatchers but only for new dispatchers. Therefore, in order for the change to be effective for all dispatchers associated with a configuration, you must forcibly terminate existing dispatchers after altering the DISPATCHERS parameter, and let the database start new ones in their place with the newly specified properties.

  The attributes LISTENER and SERVICES are not subject to the same constraint. They apply to existing dispatchers associated with the modified configuration. Attribute SESSIONS applies to existing dispatchers only if its value is reduced. However, if its value is increased, it is applied only to newly started dispatchers.

Shutting Down Specific Dispatcher Processes

With the ALTER SYSTEM statement, you leave it up to the database to determine which dispatchers to shut down to reduce the number of dispatchers. Alternatively, 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;
```

Each dispatcher is uniquely identified by a name of the form D*nnn*.

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 the database 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.

Disabling Shared Server

You disable shared server by setting SHARED_SERVERS to 0. You can do this dynamically with the ALTER SYSTEM statement. When you disable shared server, no new clients can connect in shared mode. However, Oracle Database retains some shared servers until all shared server connections are closed. The number of shared servers retained is either the number specified by the preceding setting of SHARED_SERVERS or the value of the MAX_SHARED_SERVERS parameter, whichever is smaller. If both SHARED_SERVERS and MAX_SHARED_SERVERS are set to 0, then all shared servers will terminate and requests from remaining shared server clients will be queued until the value of SHARED_SERVERS or MAX_SHARED_SERVERS is raised again.

To terminate dispatchers once all shared server clients disconnect, enter this statement:

```
ALTER SYSTEM SET DISPATCHERS = '';
```

Shared Server Data Dictionary Views

The following views are useful 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_CONFIG	Provides configuration information about the dispatchers.
V$DISPATCHER_RATE	Provides rate statistics for the dispatcher processes.
V$QUEUE	Contains information on the shared server message queues.
V$SHARED_SERVER	Contains information on the shared servers.
V$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$SGASTAT	Contains detailed statistical information about the SGA, useful for tuning.
V$SHARED_POOL_RESERVED	Lists statistics to help tune the reserved pool and space within the shared pool.

See Also:

- *Oracle Database Reference* for detailed descriptions of these views
- *Oracle Database Performance Tuning Guide* for specific information about monitoring and tuning shared server

Configuring Database Resident Connection Pooling

The database server is preconfigured to allow database resident connection pooling. However, you must explicitly enable this feature by starting the connection pool.

This section contains the following topics:

- Enabling Database Resident Connection Pooling
- Configuring the Connection Pool for Database Resident Connection Pooling
- Data Dictionary Views for Database Resident Connection Pooling

See Also: "About Database Resident Connection Pooling" on page 5-4

Enabling Database Resident Connection Pooling

Oracle Database includes a default connection pool called SYS_DEFAULT_CONNECTION_POOL. By default, this pool is created, but not started. To enable database resident connection pooling, you must explicitly start the connection pool.

To enable database resident connection pooling:

1. Start the database resident connection pool, as described in "Starting the Database Resident Connection Pool" on page 5-16.
2. Route the client connection requests to the connection pool, as described in "Routing Client Connection Requests to the Connection Pool" on page 5-16.

Starting the Database Resident Connection Pool

To start the connection pool, use the following steps:

1. Start SQL*Plus and connect to the database as the SYS user.
2. Issue the following command:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

Once started, the connection pool remains in this state until it is explicitly stopped. The connection pool is automatically restarted when the database instance is restarted if the pool was active at the time of instance shutdown.

In an Oracle Real Application Clusters (Oracle RAC) environment, you can use any instance to manage the connection pool. Any changes you make to the pool configuration are applicable on all Oracle RAC instances.

Routing Client Connection Requests to the Connection Pool

In the client application, the connect string must specify the connect type as POOLED.

The following example shows an easy connect string that enables clients to connect to a database resident connection pool:

```
examplehost.company.com:1521/books.company.com:POOLED
```

The following example shows a TNS connect descriptor that enables clients to connect to a database resident connection pool:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=myhost)
       (PORT=1521))(CONNECT_DATA=(SERVICE_NAME=sales)
       (SERVER=POOLED)))
```

Disabling Database Resident Connection Pooling

To disable database resident connection pooling, you must explicitly stop the connection pool. Use the following steps:

1. Start SQL*Plus and connect to the database as the SYS user.
2. Issue the following command:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information on the DBMS_CONNECTION_POOL package.

Note: The operation of disabling the database resident connection pool can be completed only when all client requests that have been handed off to a server are completed.

Configuring the Connection Pool for Database Resident Connection Pooling

The connection pool is configured using default parameter values. You can use the procedures in the DBMS_CONNECTION_POOL package to configure the connection pool according to your usage. In an Oracle Real Application Clusters (Oracle RAC) environment, the configuration parameters are applicable to each Oracle RAC instance.

Table 5–2 lists the parameters that you can configure for the connection pool.

Table 5–2 Configuration Parameters for Database Resident Connection Pooling

Parameter Name	Description
MINSIZE	The minimum number of pooled servers in the pool. The default value is 4.
MAXSIZE	The maximum number of pooled servers in the pool. The default value is 40. The connection pool reserves 5% of the pooled servers for authentication, and at least one pooled server is always reserved for authentication. When setting this parameter, ensure that there are enough pooled servers for both authentication and connections.
INCRSIZE	The number of pooled servers by which the pool is incremented if servers are unavailable when a client application request is received. The default value is 2.
SESSION_CACHED_CURSORS	The number of session cursors to cache in each pooled server session. The default value is 20.
INACTIVITY_TIMEOUT	The maximum time, in seconds, the pooled server can stay idle in the pool. After this time, the server is terminated. The default value is 300. This parameter does not apply if the pool is at MINSIZE.
MAX_THINK_TIME	The maximum time of inactivity, in seconds, for a client after it obtains a pooled server from the pool. After obtaining a pooled server from the pool, if the client application does not issue a database call for the time specified by MAX_THINK_TIME, the pooled server is freed and the client connection is terminated. The default value is 120.
MAX_USE_SESSION	The number of times a pooled server can be taken and released to the pool. The default value is 500000.
MAX_LIFETIME_SESSION	The time, in seconds, to live for a pooled server in the pool. The default value is 86400.
NUM_CBROK	The number of Connection Brokers that are created to handle client requests. The default value is 1. Creating multiple Connection Broker processes helps distribute the load of client connection requests if there are a large number of client applications.

Table 5–2 (Cont.) Configuration Parameters for Database Resident Connection Pooling

Parameter Name	Description
MAXCONN_CBROK	The maximum number of connections that each Connection Broker can handle. The default value is 40000. But if the maximum connections allowed by the platform on which the database is installed is lesser than the default value, this value overrides the value set using MAXCONN_CBROK. Set the per-process file descriptor limit of the operating system sufficiently high so that it supports the number of connections specified by MAXCONN_CBROK.

Using the CONFIGURE_POOL Procedure

The CONFIGURE_POOL procedure of the DBMS_CONNECTION_POOL package enables you to configure the connection pool with advanced options. This procedure is usually used when you must modify all the parameters of the connection pool.

Using the ALTER_PARAM Procedure

The ALTER_PARAM procedure of the DBMS_CONNECTION_POOL package enables you to alter a specific configuration parameter without affecting other parameters.

For example, the following command changes the minimum number of pooled servers used:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MINSIZE','10');
```

The following example, changes the maximum number of connections that each connection broker can handle to 50000.

```
SQL> EXECUTE DBMS_CONNECTION_POOL.ALTER_PARAM ('','MAXCONN_CBROK','50000');
```

Before you execute this command, ensure that the maximum number of connections allowed by the platform on which your database is installed is not less than the value you set for MAXCONN_CBROK.

For example, in Linux, the following entry in the /etc/security/limits.conf file indicates that the maximum number of connections allowed for the user test_user is 30000.

```
test_user HARD NOFILE 30000
```

To set the maximum number of connections that each connection broker can allow to 50000, first change the value in the limits.conf file to a value not less than 50000.

Restoring the Connection Pool Default Settings

If you have made changes to the connection pool parameters, but you want to revert to the default pool settings, use the RESTORE_DEFAULT procedure of the DBMS_CONNECTION_POOL package. The command to restore the connection pool to its default settings is:

```
SQL> EXECUTE DBMS_CONNECTION_POOL.RESTORE_DEFAULTS();
```

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information on the DBMS_CONNECTION_POOL package.

Data Dictionary Views for Database Resident Connection Pooling

Table 5–3 lists the data dictionary views that provide information about database resident connection pooling. Use these views to obtain information about your connection pool and to monitor the performance of database resident connection pooling.

Table 5–3 Data Dictionary Views for Database Resident Connection Pooling

View	Description
`DBA_CPOOL_INFO`	Contains information about the connection pool such as the pool status, the maximum and minimum number of connections, and timeout for idle sessions.
`V$CPOOL_CONN_INFO`	Contains information about each connection to the connection broker.
`V$CPOOL_STATS`	Contains pool statistics such as the number of session requests, number of times a session that matches the request was found in the pool, and total wait time for a session request.
`V$CPOOL_CC_STATS`	Contains connection class level statistics for the pool.

See Also: *Oracle Database Reference* for more information about these views.

About Oracle Database Background Processes

To maximize performance and accommodate many users, a multiprocess Oracle Database system uses **background processes**. Background processes consolidate functions that would otherwise be handled by multiple database programs running for each user process. Background processes asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.

Table 5–4 describes the fundamental background processes, many of which are discussed in more detail elsewhere in this book. The use of additional database features or options can cause more background processes to be present. For example:

- When you use Oracle Streams Advanced Queuing, the queue monitor (QMN*n*) background process is present.
- When you specify the `FILE_MAPPING` initialization parameter for mapping datafiles to physical devices on a storage subsystem, then the FMON process is present.
- If you use Oracle Automatic Storage Management (Oracle ASM), then additional Oracle ASM–specific background processes are present.

Table 5–4 Oracle Database Background Processes

Process Name	Description
Database writer (DBW*n*)	The database writer writes modified blocks from the database buffer cache to the datafiles. Oracle Database allows a maximum of 36 database writer processes (DBW0-DBW9 and DBWa-DBWj). The `DB_WRITER_PROCESSES` initialization parameter specifies the number of DBW*n* processes. The database selects an appropriate default setting for this initialization parameter or adjusts a user-specified setting based on the number of CPUs and the number of processor groups. For more information about setting the `DB_WRITER_PROCESSES` initialization parameter, see the *Oracle Database Performance Tuning Guide*.
Log writer (LGWR)	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). LGWR writes the redo log entries sequentially into a redo log file. If the database has a multiplexed redo log, then LGWR writes the redo log entries to a group of redo log files. See Chapter 12, "Managing the Redo Log" for information about the log writer process.
Checkpoint (CKPT)	At specific times, all modified database buffers in the system global area are written to the datafiles by DBW*n*. This event is called a checkpoint. The checkpoint process is responsible for signalling DBW*n* at checkpoints and updating all the datafiles and control files of the database to indicate the most recent checkpoint.
System monitor (SMON)	The system monitor performs recovery when a failed instance starts up again. In an Oracle Real Application Clusters database, 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 terminated transactions skipped during system failure 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.
Process monitor (PMON)	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 (described later in this table) and server processes and restarts them if they have failed.
Archiver (ARC*n*)	One or more archiver processes copy the redo log files to archival storage when they are full or a log switch occurs. Archiver processes are the subject of Chapter 13, "Managing Archived Redo Logs".
Recoverer (RECO)	The recoverer process is used to resolve distributed transactions that are pending because of 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 35, "Managing Distributed Transactions".
Dispatcher (D*nnn*)	Dispatchers are optional background processes, present only when the shared server configuration is used. Shared server was discussed previously in "Configuring Oracle Database for Shared Server" on page 5-6.

See Also: *Oracle Database Reference* for a complete list of Oracle Database background processes

Managing Processes for Parallel SQL Execution

Note: The parallel execution feature described in this section is available with the Oracle Database Enterprise Edition.

This section describes how to manage parallel processing of SQL statements. In this configuration Oracle Database 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 the database

An example of using parallel SQL execution is contained in "Parallelizing Table Creation" on page 20-15.

The following topics are contained in this section:

- About Parallel Execution Servers
- Altering Parallel Execution for a Session

See Also:

- *Oracle Database Performance Tuning Guide* for information about using parallel hints

About Parallel Execution Servers

When an instance starts up, Oracle Database creates a pool of parallel execution servers which are available for any parallel operation. A process called 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.

The parallel execution servers are enabled by default, because by default the value for `PARALLEL_MAX_SERVERS` initialization parameter is set >0. The processes are available for use by the various Oracle Database features that are capable of exploiting parallelism. Related initialization parameters are tuned by the database for the majority of users, but you can alter them as needed to suit your environment. For ease of tuning, some parameters can be altered dynamically.

Parallelism can be used by several features, including transaction recovery, replication, and SQL execution. In the case of parallel SQL execution, the topic discussed in this book, parallel 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.

See Also: *Oracle Database VLDB and Partitioning Guide* for more information about using parallel execution

Altering Parallel Execution for a Session

You control parallel SQL execution for a session using the `ALTER SESSION` statement.

Disabling Parallel SQL Execution

You disable parallel SQL execution with an `ALTER SESSION DISABLE PARALLEL DML|DDL|QUERY` statement. All subsequent DML (`INSERT`, `UPDATE`, `DELETE`), DDL (`CREATE`, `ALTER`), or query (`SELECT`) operations are executed serially after such a statement is issued. They will be executed serially regardless of any parallel attribute associated with the table or indexes involved. However, statements with a `PARALLEL` hint override the session settings.

The following statement disables parallel DDL operations:

```
ALTER SESSION DISABLE PARALLEL DDL;
```

Enabling Parallel SQL Execution

You enable parallel SQL execution with an `ALTER SESSION ENABLE PARALLEL DML|DDL|QUERY` statement. Subsequently, when a `PARALLEL` clause or parallel hint is associated with a statement, those DML, DDL, or query statements will execute in parallel. By default, parallel execution is enabled for DDL and query statements.

A DML statement can be parallelized only if you specifically issue an `ALTER SESSION` statement to enable parallel DML:

```
ALTER SESSION ENABLE PARALLEL DML;
```

Forcing Parallel SQL 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. Forcing parallel execution overrides any parallel hints in SQL statements.

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;
```

Managing Processes for External Procedures

This section contains:

- About External Procedures
- DBA Tasks to Enable External Procedure Calls

About External Procedures

External procedures are procedures that are written in C, C++, Java, or other language, compiled and stored outside the database, and then called by user sessions. For example, a PL/SQL program unit could call 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 a libunit in the case of a Java class method, and are registered with the base language. Oracle Database provides a special-purpose interface, the **call specification** (call spec), that enables users to call external procedures in other languages.

When a user session calls an external procedure, the database starts an external procedure agent on the database host computer. The default name of the agent is

extproc. Each session has its own dedicated agent. When a session terminates, the database terminates its agent.

User applications pass to the external procedure agent the name of the DLL or libunit, the name of the external procedure, and any relevant parameters. The external procedure agent then loads the DLL or libunit, runs the external procedure, and passes back to the application any values returned by the external procedure.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for information about external procedures

DBA Tasks to Enable External Procedure Calls

Enabling external procedure calls may involve the following DBA tasks:

- Configuring the listener to start the extproc agent

  By default, the database starts the extproc process. Under the following circumstances, you must change this default configuration so that the listener starts the extproc process:

  - You want to use a multithreaded extproc agent
  - The database is running in shared server mode on Windows
  - An AGENT clause in the LIBRARY specification or an AGENT IN clause in the PROCEDURE or FUNCTION specification redirects external procedures to a different extproc agent

  Instructions for changing the default configuration are found in the subsection entitled "Set Up the Environment" in the chapter "Developing Applications with Multiple Programming Languages" in *Oracle Database Advanced Application Developer's Guide*.

- Managing libraries or granting privileges related to managing libraries

  The database requires DLLs to be accessed through a schema object called a library. For security purposes, by default, only users with the DBA role can create and manage libraries. Therefore, you may be asked to:

  - Use the CREATE LIBRARY statement to create the library objects that the developers need.
  - Grant the following privileges to developers: CREATE LIBRARY, CREATE ANY LIBRARY, ALTER ANY LIBRARY, EXECUTE ANY LIBRARY, and EXECUTE ON *library_name*.

  Only make an explicit grant of these privileges to trusted users, and never to the PUBLIC role. If you plan to create PL/SQL interfaces to libraries, then only grant the EXECUTE privilege to the PL/SQL interface. Do not grant EXECUTE on the underlying library. You must have the EXECUTE object privilege on the library to create the PL/SQL interface. However, users have this privilege automatically in their own schemas. Explicit grants of EXECUTE object privilege on a library are rarely required.

> **See Also:** *Oracle Database PL/SQL Language Reference* for information about the CREATE LIBRARY statement

Terminating Sessions

Sometimes it is necessary 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, any active transactions of the session are rolled back, and resources held by the session (such as locks and memory areas) are immediately released and available to other sessions.

You 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 index number and serial number. To identify the system identifier (SID) and serial number of a session, query the `V$SESSION` dynamic performance view. For example, 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 Database. A session is `INACTIVE` if it is not making a SQL call to the database.

> **See Also:** *Oracle 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 you terminate the session, 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 Database returns the following message:

```
ORA-03114: not connected to ORACLE
```

An active session cannot be interrupted when it is performing network I/O or rolling back a transaction. Such a 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 Database (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, the STATUS of the session 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, and 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.
```

Process and Session Data Dictionary Views

The following are the data dictionary views that can help you manage processes and sessions.

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	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 Database release.
V$SESSION_WAIT	Displays the current or last wait for each session
V$SESSION_WAIT_HISTORY	Lists the last ten wait events for each active session

View	Description
V$WAIT_CHAINS	Displays information about blocked sessions
V$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 areas. Contains one row for each SQL string. Provides statistics about SQL statements that are in memory, parsed, and ready for execution

6

Managing Memory

In this chapter:

- About Memory Management
- Memory Architecture Overview
- Using Automatic Memory Management
- Configuring Memory Manually
- Configuring Database Smart Flash Cache
- Memory Management Reference

About Memory Management

Memory management involves maintaining optimal sizes for the Oracle Database instance memory structures as demands on the database change. The memory structures that must be managed are the system global area (SGA) and the instance program global area (instance PGA).

Oracle Database supports various memory management methods, which are chosen by initialization parameter settings. Oracle recommends that you enable the method known as *automatic memory management*.

Automatic Memory Management

Beginning with Release 11*g*, Oracle Database can manage the SGA memory and instance PGA memory completely automatically. You designate only the total memory size to be used by the instance, and Oracle Database dynamically exchanges memory between the SGA and the instance PGA as needed to meet processing demands. This capability is referred to as *automatic memory management*. With this memory management method, the database also dynamically tunes the sizes of the individual SGA components and the sizes of the individual PGAs.

Manual Memory Management

If you prefer to exercise more direct control over the sizes of individual memory components, you can disable automatic memory management and configure the database for manual memory management. There are a few different methods available for manual memory management. Some of these methods retain some degree of automation. The methods therefore vary in the amount of effort and knowledge required by the DBA. These methods are:

- Automatic shared memory management - for the SGA
- Manual shared memory management - for the SGA

- Automatic PGA memory management - for the instance PGA
- Manual PGA memory management - for the instance PGA

These memory management methods are described later in this chapter.

Note: The easiest way to manage memory is to use the graphical user interface of Oracle Enterprise Manager.

To manage memory with Enterprise Manager:

1. Do one of the following:

   - If you are using Oracle Enterprise Manager Database Control, access the Database Home page. See *Oracle Database 2 Day DBA* for instructions.

   - If you are using Oracle Enterprise Manager Grid Control, go to the desired database target. The Database Home page is displayed.

2. At the top of the page, click **Server** to display the Server page.
3. In the Database Configuration section, click **Memory Advisors**.

See Also: *Oracle Database Concepts* for an introduction to the various automatic and manual methods of managing memory.

Memory Architecture Overview

The basic memory structures associated with Oracle Database include:

- System Global Area (SGA)

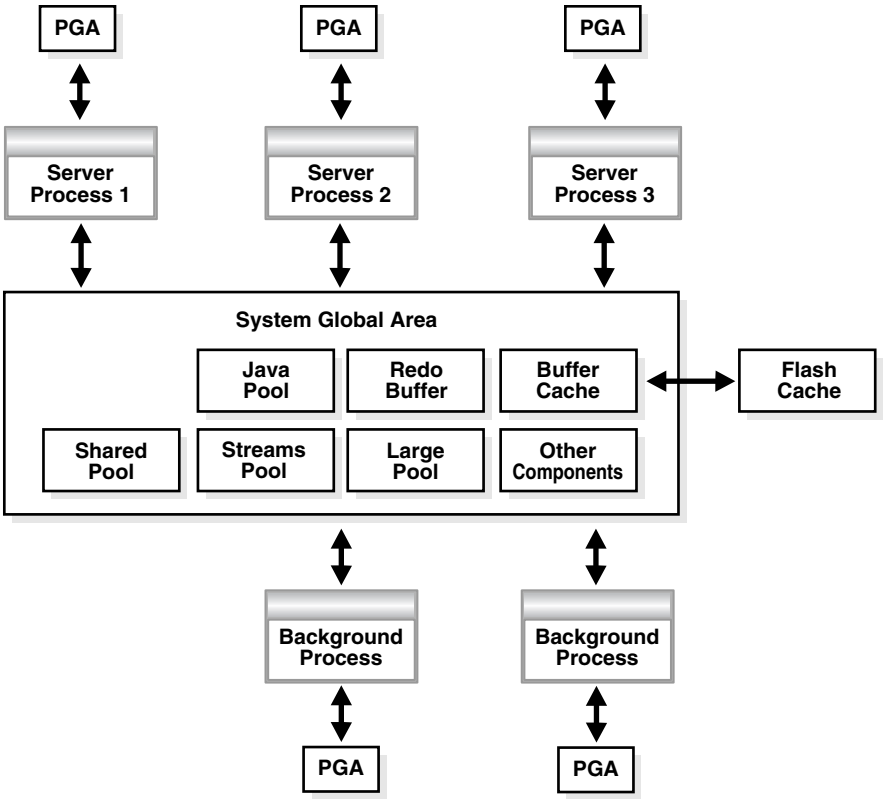
  The SGA is a group of shared memory structures, known as *SGA components*, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

- Program Global Area (PGA)

  A PGA is a memory region that contains data and control information for a server process. It is nonshared memory created by Oracle Database when a server process is started. Access to the PGA is exclusive to the server process. There is one PGA for each server process. Background processes also allocate their own PGAs. The total PGA memory allocated for all background and server processes attached to an Oracle Database instance is referred to as the **total instance PGA memory**, and the collection of all individual PGAs is referred to as the **total instance PGA**, or just **instance PGA**.

Figure 6–1 illustrates the relationships among these memory structures.

Figure 6–1 Oracle Database Memory Structures



If your database is running on Solaris or Oracle Linux, you can optionally add another memory component: Database Smart Flash Cache (the flash cache). The flash cache is an extension of the SGA-resident buffer cache, providing a level 2 cache for database blocks. It can improve response time and overall throughput, especially for read-intensive online transaction processing (OLTP) workloads. The flash cache resides on one or more flash disk devices, which are solid state storage devices that use flash memory.

The flash cache is typically more economical than additional main memory, and is an order of magnitude faster than disk drives.

See Also:

- *Oracle Database Concepts* for more information on memory architecture in an Oracle Database instance
- "Configuring Database Smart Flash Cache" on page 6-21

Using Automatic Memory Management

This section provides background information on the automatic memory management feature of Oracle Database, and includes instructions for enabling this feature. The following topics are covered:

- About Automatic Memory Management
- Enabling Automatic Memory Management
- Monitoring and Tuning Automatic Memory Management

About Automatic Memory Management

The simplest way to manage instance memory is to allow the Oracle Database instance to automatically manage and tune it for you. To do so (on most platforms), you set only a *target* memory size initialization parameter (MEMORY_TARGET) and optionally a *maximum* memory size initialization parameter (MEMORY_MAX_TARGET). The total memory that the instance uses remains relatively constant, based on the value of MEMORY_TARGET, and the instance automatically distributes memory between the system global area (SGA) and the instance program global area (instance PGA). As memory requirements change, the instance dynamically redistributes memory between the SGA and instance PGA.

When automatic memory management is not enabled, you must size both the SGA and instance PGA manually.

Because the MEMORY_TARGET initialization parameter is dynamic, you can change MEMORY_TARGET at any time without restarting the database. MEMORY_MAX_TARGET, which is not dynamic, serves as an upper limit so that you cannot accidentally set MEMORY_TARGET too high, and so that enough memory is set aside for the database instance in case you do want to increase total instance memory in the future. Because certain SGA components either cannot easily shrink or must remain at a minimum size, the instance also prevents you from setting MEMORY_TARGET too low.

If you create your database with Database Configuration Assistant (DBCA) and choose the basic installation option, automatic memory management is enabled. If you choose advanced installation, Database Configuration Assistant (DBCA) enables you to select automatic memory management.

Note: You cannot enable automatic memory management if the LOCK_SGA initialization parameter is TRUE. See *Oracle Database Reference* for information about this parameter.

See Also:

- "Platforms That Support Automatic Memory Management" on page 6-23

Enabling Automatic Memory Management

If you did not enable automatic memory management upon database creation (either by selecting the proper options in DBCA or by setting the appropriate initialization parameters for the CREATE DATABASE SQL statement), you can enable it at a later time. Enabling automatic memory management involves a shutdown and restart of the database.

To enable automatic memory management

1. Start SQL*Plus and connect to the database as SYSDBA.

   See "Connecting to the Database with SQL*Plus" on page 1-7 and "Database Administrator Authentication" on page 1-16 for instructions.

2. Calculate the minimum value for MEMORY_TARGET as follows:

   a. Determine the current sizes of SGA_TARGET and PGA_AGGREGATE_TARGET by entering the following SQL*Plus command:

   ```
   SHOW PARAMETER TARGET
   ```

SQL*Plus displays the values of all initialization parameters with the string TARGET in the parameter name.

```
NAME                           TYPE        VALUE
------------------------------ ----------- ----------------
archive_lag_target             integer     0
db_flashback_retention_target  integer     1440
fast_start_io_target           integer     0
fast_start_mttr_target         integer     0
memory_max_target              big integer 0
memory_target                  big integer 0
pga_aggregate_target           big integer 90M
sga_target                     big integer 272M
```

b. Run the following query to determine the maximum instance PGA allocated since the database was started:

```
select value from v$pgastat where name='maximum PGA allocated';
```

c. Compute the maximum value between the query result from step 2b and PGA_AGGREGATE_TARGET. Add SGA_TARGET to this value.

```
memory_target = sga_target + max(pga_aggregate_target, maximum PGA
allocated)
```

For example, if SGA_TARGET is 272M and PGA_AGGREGATE_TARGET is 90M as shown above, and if the maximum PGA allocated is determined to be 120M, then MEMORY_TARGET should be at least 392M (272M + 120M).

3. Choose the value for MEMORY_TARGET that you want to use.

This can be the minimum value that you computed in step 2, or you can choose to use a larger value if you have enough physical memory available.

4. For the MEMORY_MAX_TARGET initialization parameter, decide on a maximum amount of memory that you would want to allocate to the database for the foreseeable future. That is, determine the maximum value for the sum of the SGA and instance PGA sizes. This number can be larger than or the same as the MEMORY_TARGET value that you chose in the previous step.

5. Do one of the following:

- If you started your Oracle Database instance with a server parameter file, which is the default if you created the database with the Database Configuration Assistant (DBCA), enter the following command:

  ```
  ALTER SYSTEM SET MEMORY_MAX_TARGET = nM SCOPE = SPFILE;
  ```

  where *n* is the value that you computed in Step 4.

  The SCOPE = SPFILE clause sets the value only in the server parameter file, and not for the running instance. You must include this SCOPE clause because MEMORY_MAX_TARGET is not a dynamic initialization parameter.

- If you started your instance with a text initialization parameter file, manually edit the file so that it contains the following statements:

  ```
  memory_max_target = nM
  memory_target = mM
  ```

  where *n* is the value that you determined in Step 4, and *m* is the value that you determined in step 3.

> **Note:** In a text initialization parameter file, if you omit the line for MEMORY_MAX_TARGET and include a value for MEMORY_TARGET, the database automatically sets MEMORY_MAX_TARGET to the value of MEMORY_TARGET. If you omit the line for MEMORY_TARGET and include a value for MEMORY_MAX_TARGET, the MEMORY_TARGET parameter defaults to zero. After startup, you can then dynamically change MEMORY_TARGET to a nonzero value, provided that it does not exceed the value of MEMORY_MAX_TARGET.

6. Shut down and restart the database.

   See Chapter 3, "Starting Up and Shutting Down" on page 3-1 for instructions.

7. If you started your Oracle Database instance with a server parameter file, enter the following commands:

```
ALTER SYSTEM SET MEMORY_TARGET = nM;
ALTER SYSTEM SET SGA_TARGET = 0;
ALTER SYSTEM SET PGA_AGGREGATE_TARGET = 0;
```

   where *n* is the value that you determined in step 3.

> **Note:** The preceding steps instruct you to set SGA_TARGET and PGA_AGGREGATE_TARGET to zero so that the sizes of the SGA and instance PGA are tuned up and down as required, without restrictions. You can omit the statements that set these parameter values to zero and leave either or both of the values as positive numbers. In this case, the values act as minimum values for the sizes of the SGA or instance PGA.

See Also:

- "About Automatic Memory Management" on page 6-4
- "Memory Architecture Overview" on page 6-2
- *Oracle Database SQL Language Reference* for information on the ALTER SYSTEM SQL statement

Monitoring and Tuning Automatic Memory Management

The dynamic performance view V$MEMORY_DYNAMIC_COMPONENTS shows the current sizes of all dynamically tuned memory components, including the total sizes of the SGA and instance PGA.

The view V$MEMORY_TARGET_ADVICE provides tuning advice for the MEMORY_TARGET initialization parameter.

```
SQL>  select * from v$memory_target_advice order by memory_size;

MEMORY_SIZE MEMORY_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR    VERSION
----------- ------------------ ------------ -------------------- ----------
        180                 .5          458                1.344          0
        270                .75          367               1.0761          0
        360                  1          341                    1          0
        450               1.25          335                .9817          0
        540                1.5          335                .9817          0
        630               1.75          335                .9817          0
```

```
720                    2          335                .9817          0
```

The row with the MEMORY_SIZE_FACTOR of 1 shows the current size of memory, as set by the MEMORY_TARGET initialization parameter, and the amount of DB time required to complete the current workload. In previous and subsequent rows, the results show several alternative MEMORY_TARGET sizes. For each alternative size, the database shows the size factor (the multiple of the current size), and the estimated DB time to complete the current workload if the MEMORY_TARGET parameter were changed to the alternative size. Notice that for a total memory size smaller than the current MEMORY_TARGET size, estimated DB time increases. Notice also that in this example, there is nothing to be gained by increasing total memory size beyond 450MB. However, this situation might change if a complete workload has not yet been run.

Enterprise Manager provides an easy-to-use graphical memory advisor to help you select an optimal size for MEMORY_TARGET. See *Oracle Database 2 Day DBA* for details.

See Also:

- *Oracle Database Reference* for more information about these dynamic performance views
- *Oracle Database Performance Tuning Guide* for a definition of DB time.

Configuring Memory Manually

If you prefer to exercise more direct control over the sizes of individual memory components, you can disable automatic memory management and configure the database for manual memory management. There are two different manual memory management methods for the SGA, and two for the instance PGA.

The two manual memory management methods for the SGA vary in the amount of effort and knowledge required by the DBA. With *automatic shared memory management*, you set target and maximum sizes for the SGA. The database then sets the total size of the SGA to your designated target, and dynamically tunes the sizes of many SGA components. With *manual shared memory management*, you set the sizes of several individual SGA components, thereby determining the overall SGA size. You then manually tune these individual SGA components on an ongoing basis.

For the instance PGA, there is *automatic PGA memory management*, in which you set a target size for the instance PGA. The database then sets the size of the instance PGA to your target, and dynamically tunes the sizes of individual PGAs. There is also *manual PGA memory management*, in which you set maximum work area size for each type of SQL operator (such as sort or hash-join). This memory management method, although supported, is not recommended.

The following sections provide details on all of these manual memory management methods:

- Using Automatic Shared Memory Management
- Using Manual Shared Memory Management
- Using Automatic PGA Memory Management
- Using Manual PGA Memory Management

See Also: *Oracle Database Concepts* for an overview of Oracle Database memory management methods.

Using Automatic Shared Memory Management

This section contains the following topics:

- About Automatic Shared Memory Management
- Components and Granules in the SGA
- Setting Maximum SGA Size
- Setting SGA Target Size
- Enabling Automatic Shared Memory Management
- Automatic Shared Memory Management Advanced Topics

See Also:

- *Oracle Database Performance Tuning Guide* for information about tuning the components of the SGA

About Automatic Shared Memory Management

Automatic Shared Memory Management simplifies SGA memory management. You specify the total amount of SGA memory available to an instance using the `SGA_TARGET` initialization parameter and Oracle Database automatically distributes this memory among the various SGA components to ensure the most effective memory utilization.

When automatic shared memory management is enabled, the sizes of the different SGA components are flexible and can adapt to the needs of a workload without requiring any additional configuration. The database automatically distributes the available memory among the various components as required, allowing the system to maximize the use of all available SGA memory.

If you are using a server parameter file (`SPFILE`), the database remembers the sizes of the automatically tuned SGA components across instance shutdowns. As a result, the database instance does not need to learn the characteristics of the workload again each time the instance is started. The instance can begin with information from the previous instance and continue evaluating workload where it left off at the last shutdown.

Components and Granules in the SGA

The SGA comprises several memory **components**, which are pools of memory used to satisfy a particular class of memory allocation requests. Examples of memory components include the shared pool (used to allocate memory for SQL and PL/SQL execution), the java pool (used for java objects and other java execution memory), and the buffer cache (used for caching disk blocks). All SGA components allocate and deallocate space in units of **granules**. Oracle Database tracks SGA memory use in internal numbers of granules for each SGA component.

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 1 GB, then granule size is 4 MB. For SGAs larger than 1 GB, granule size is 16 MB. Some platform dependencies may arise. For example, on 32-bit Windows NT, the granule size is 8 MB for SGAs larger than 1 GB. Consult your operating system specific documentation for more details.

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

If you specify a size for a component that is not a multiple of granule size, Oracle Database rounds 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, the database actually allocates 12 MB.

Setting Maximum SGA Size

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, large pool, Java pool, and streams 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 Database selects a default value that is the sum of all components specified or defaulted at initialization time. If you do specify SGA_MAX_SIZE, and at the time the database is initialized the value is less than the sum of the memory allocated for all components, either explicitly in the parameter file or by default, then the database ignores the setting for SGA_MAX_SIZE and chooses a correct value for this parameter.

Setting SGA Target Size

You enable the automatic shared memory management feature by setting the SGA_TARGET parameter to a nonzero value. This parameter sets the total size of the SGA. It replaces the parameters that control the memory allocated for a specific set of individual components, which are now automatically and dynamically resized (tuned) as needed.

> **Note:** The STATISTICS_LEVEL initialization parameter must be set to TYPICAL (the default) or ALL for automatic shared memory management to function.

Table 6–1 lists the SGA components that are automatically sized when SGA_TARGET is set. For each SGA component, its corresponding initialization parameter is listed.

Table 6–1 Automatically Sized SGA Components and Corresponding Parameters

SGA Component	Initialization Parameter
Fixed SGA and other internal allocations needed by the Oracle Database instance	N/A
The shared pool	SHARED_POOL_SIZE
The large pool	LARGE_POOL_SIZE
The Java pool	JAVA_POOL_SIZE
The buffer cache	DB_CACHE_SIZE
The Streams pool	STREAMS_POOL_SIZE

The manually sized parameters listed in Table 6–2, if they are set, take their memory from SGA_TARGET, leaving what is available for the components listed in Table 6–1.

Table 6–2 Manually Sized SGA Components that Use SGA_TARGET Space

SGA Component	Initialization Parameter
The log buffer	LOG_BUFFER

Table 6–2 (Cont.) Manually Sized SGA Components that Use SGA_TARGET Space

SGA Component	Initialization Parameter
The keep and recycle buffer caches	DB_KEEP_CACHE_SIZE DB_RECYCLE_CACHE_SIZE
Nonstandard block size buffer caches	DB_*n*K_CACHE_SIZE

In addition to setting SGA_TARGET to a nonzero value, you must set to zero all initialization parameters listed in Table 6–1 to enable full automatic tuning of the automatically sized SGA components.

Alternatively, you can set one or more of the automatically sized SGA components to a nonzero value, which is then used as the minimum setting for that component during SGA tuning. This is discussed in detail later in this section.

> **Note:** An easier way to enable automatic shared memory management is to use Oracle Enterprise Manager. When you enable automatic shared memory management and set the Total SGA Size, Enterprise Manager automatically generates the ALTER SYSTEM statements to set SGA_TARGET to the specified size and to set all automatically sized SGA components to zero.
>
> If you use SQL*Plus to set SGA_TARGET, you must then set the automatically sized SGA components to zero or to a minimum value.

SGA and Virtual Memory For optimal performance in most systems, the entire SGA should fit in real memory. If it does not, and if virtual memory is used to store parts of it, then overall database system performance can decrease dramatically. The reason for this is that portions of the SGA are paged (written to and read from disk) by the operating system.

See your operating system documentation for instructions for monitoring paging activity. You can also view paging activity from the Performance property page of the Host page of Enterprise Manager.

Monitoring and Tuning SGA Target Size The V$SGAINFO view provides information on the current tuned sizes of various SGA components.

The V$SGA_TARGET_ADVICE view provides information that helps you decide on a value for SGA_TARGET.

```
SQL> select * from v$sga_target_advice order by sga_size;

  SGA_SIZE SGA_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR ESTD_PHYSICAL_READS
---------- --------------- ------------ ------------------- -------------------
       290              .5       448176              1.6578             1636103
       435             .75       339336              1.2552             1636103
       580               1       270344                   1             1201780
       725            1.25       239038               .8842              907584
       870             1.5       211517               .7824              513881
      1015            1.75       201866               .7467              513881
      1160               2       200703               .7424              513881
```

The information in this view is similar to that provided in the V$MEMORY_TARGET_ADVICE view for automatic memory management. See "Monitoring and Tuning Automatic Memory Management" on page 6-6 for an explanation of that view.

Enterprise Manager provides an easy-to-use graphical memory advisor to help you select an optimal size for SGA_TARGET. See *Oracle Database 2 Day DBA* for details.

> **See Also:** *Oracle Database Reference* for more information about these dynamic performance views

Enabling Automatic Shared Memory Management

The procedure for enabling automatic shared memory management (ASMM) differs depending on whether you are changing to ASMM from manual shared memory management or from automatic memory management.

To change to ASMM from manual shared memory management:

1. Run the following query to obtain a value for SGA_TARGET:

```
SELECT (
   (SELECT SUM(value) FROM V$SGA) -
   (SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY)
   ) "SGA_TARGET"
FROM DUAL;
```

2. Set the value of SGA_TARGET, either by editing the text initialization parameter file and restarting the database, or by issuing the following statement:

```
ALTER SYSTEM SET SGA_TARGET=value [SCOPE={SPFILE|MEMORY|BOTH}]
```

   where *value* is the value computed in step 1 or is some value between the sum of all SGA component sizes and SGA_MAX_SIZE. For more information on the ALTER SYSTEM statement and its SCOPE clause, see *Oracle Database SQL Language Reference*.

3. Do one of the following:
   - For more complete automatic tuning, set the values of the automatically sized SGA components listed in Table 6–1 to zero. Do this by editing the text initialization parameter file or by issuing ALTER SYSTEM statements.
   - To control the minimum size of one or more automatically sized SGA components, set those component sizes to the desired value. (See the next section for details.) Set the values of the other automatically sized SGA components to zero. Do this by editing the text initialization parameter file or by issuing ALTER SYSTEM statements.

To change to ASMM from automatic memory management:

1. Set the MEMORY_TARGET initialization parameter to 0.

```
ALTER SYSTEM SET MEMORY_TARGET = 0;
```

   The database sets SGA_TARGET based on current SGA memory allocation.

2. Do one of the following:
   - For more complete automatic tuning, set the sizes of the automatically sized SGA components listed in Table 6–1 to zero. Do this by editing the text initialization parameter file or by issuing ALTER SYSTEM statements.
   - To control the minimum size of one or more automatically sized SGA components, set those component sizes to the desired value. (See the next section for details.) Set the sizes of the other automatically sized SGA components to zero. Do this by editing the text initialization parameter file or by issuing ALTER SYSTEM statements.

Example For example, suppose you currently have the following configuration of parameters for an instance configured for manual shared memory management and with SGA_MAX_SIZE set to 1200M:

- SHARED_POOL_SIZE = 200M
- DB_CACHE_SIZE = 500M
- LARGE_POOL_SIZE=200M

Also assume the following query results:

Query	Result
SELECT SUM(value) FROM V$SGA	1200M
SELECT CURRENT_SIZE FROM V$SGA_DYNAMIC_FREE_MEMORY	208M

You can take advantage of automatic shared memory management by setting Total SGA Size to 992M in Oracle Enterprise Manager, or by issuing the following statements:

```
ALTER SYSTEM SET SGA_TARGET = 992M;
ALTER SYSTEM SET SHARED_POOL_SIZE = 0;
ALTER SYSTEM SET LARGE_POOL_SIZE = 0;
ALTER SYSTEM SET JAVA_POOL_SIZE = 0;
ALTER SYSTEM SET DB_CACHE_SIZE = 0;
ALTER SYSTEM SET STREAMS_POOL_SIZE = 0;
```

where 992M = 1200M minus 208M.

Automatic Shared Memory Management Advanced Topics

This section provides a closer look at automatic shared memory management. It includes the following topics:

- Setting Minimums for Automatically Sized SGA Components
- Dynamic Modification of SGA_TARGET
- Modifying Parameters for Automatically Sized Components
- Modifying Parameters for Manually Sized Components

Setting Minimums for Automatically Sized SGA Components You can exercise some control over the size of the automatically sized SGA components by specifying minimum values for the parameters corresponding to these components. Doing so can be useful if you know that an application cannot function properly without a minimum amount of memory in specific components. You specify the minimum amount of SGA space for a component by setting a value for its corresponding initialization parameter.

Manually limiting the minimum size of one or more automatically sized components reduces the total amount of memory available for dynamic adjustment. This reduction in turn limits the ability of the system to adapt to workload changes. Therefore, this practice is not recommended except in exceptional cases. The default automatic management behavior maximizes both system performance and the use of available resources.

Dynamic Modification of SGA_TARGET The SGA_TARGET parameter can be dynamically increased up to the value specified for the SGA_MAX_SIZE parameter, and it can also be reduced. If you reduce the value of SGA_TARGET, the system identifies one or more automatically tuned components for which to release memory. You can reduce SGA_

TARGET until one or more automatically tuned components reach their minimum size. Oracle Database determines the minimum allowable value for SGA_TARGET taking into account several factors, including values set for the automatically sized components, manually sized components that use SGA_TARGET space, and number of CPUs.

The change in the amount of physical memory consumed when SGA_TARGET is modified depends on the operating system. On some UNIX platforms that do not support dynamic shared memory, the physical memory in use by the SGA is equal to the value of the SGA_MAX_SIZE parameter. On such platforms, there is no real benefit in setting SGA_TARGET to a value smaller than SGA_MAX_SIZE. Therefore, setting SGA_MAX_SIZE on those platforms is not recommended.

On other platforms, such as Solaris and Windows, the physical memory consumed by the SGA is equal to the value of SGA_TARGET.

For example, suppose you have an environment with the following configuration:

- SGA_MAX_SIZE = 1024M
- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

In this example, the value of SGA_TARGET can be resized up to 1024M and can also be reduced until one or more of the automatically sized components reaches its minimum size. The exact value depends on environmental factors such as the number of CPUs on the system. However, the value of DB_8K_CACHE_SIZE remains fixed at all times at 128M

Note: When enabling automatic shared memory management, it is best to set SGA_TARGET to the desired nonzero value before starting the database. Dynamically modifying SGA_TARGET from zero to a nonzero value may not achieve the desired results because the shared pool may not be able to shrink. After startup, you can dynamically tune SGA_TARGET up or down as required.

Modifying Parameters for Automatically Sized Components When SGA_TARGET is not set, the automatic shared memory management feature is not enabled. Therefore the rules governing resize for all component parameters are the same as in earlier releases. However, when automatic shared memory management is enabled, the manually specified sizes of automatically sized components serve as a lower bound for the size of the components. You can modify this limit dynamically by changing the values of the corresponding parameters.

If the specified lower limit for the size of a given SGA component is less than its current size, there is no immediate change in the size of that component. The new setting only limits the automatic tuning algorithm to that reduced minimum size in the future. For example, consider the following configuration:

- SGA_TARGET = 512M
- LARGE_POOL_SIZE = 256M
- Current actual large pool size = 284M

In this example, if you increase the value of LARGE_POOL_SIZE to a value greater than the actual current size of the component, the system expands the component to accommodate the increased minimum size. For example, if you increase the value of LARGE_POOL_SIZE to 300M, then the system increases the large pool incrementally until

it reaches 300M. This resizing occurs at the expense of one or more automatically tuned components.

If you decrease the value of LARGE_POOL_SIZE to 200, there is no immediate change in the size of that component. The new setting only limits the reduction of the large pool size to 200 M in the future.

Modifying Parameters for Manually Sized Components Parameters for manually sized components can be dynamically altered as well. However, rather than setting a minimum size, the value of the parameter specifies the precise size of the corresponding component. When you increase the size of a manually sized component, extra memory is taken away from one or more automatically sized components. When you decrease the size of a manually sized component, the memory that is released is given to the automatically sized components.

For example, consider this configuration:

- SGA_TARGET = 512M
- DB_8K_CACHE_SIZE = 128M

In this example, increasing DB_8K_CACHE_SIZE by 16M to 144M means that the 16M is taken away from the automatically sized components. Likewise, reducing DB_8K_CACHE_SIZE by 16M to 112M means that the 16M is given to the automatically sized components.

Using Manual Shared Memory Management

If you decide not to use automatic memory management or automatic shared memory management, you must manually configure several SGA component sizes, and then monitor and tune these sizes on an ongoing basis as the database workload changes. This section provides guidelines on setting the parameters that control the sizes of these SGA components.

If you create your database with DBCA and choose manual shared memory management, DBCA provides fields where you must enter sizes for the buffer cache, shared pool, large pool, and Java pool. It then sets the corresponding initialization parameters in the server parameter file (SPFILE) that it creates. If you instead create the database with the CREATE DATABASE SQL statement and a text initialization parameter file, you can do one of the following:

- Provide values for the initialization parameters that set SGA component sizes.
- Omit SGA component size parameters from the text initialization file. Oracle Database chooses reasonable defaults for any component whose size you do not set.

This section contains the following topics:

- Enabling Manual Shared Memory Management
- Setting the Buffer Cache Initialization Parameters
- Specifying the Shared Pool Size
- Specifying the Large Pool Size
- Specifying the Java Pool Size
- Specifying the Streams Pool Size
- Specifying the Result Cache Maximum Size
- Specifying Miscellaneous SGA Initialization Parameters

Enabling Manual Shared Memory Management

There is no initialization parameter that in itself enables manual shared memory management. You effectively enable manual shared memory management by disabling both automatic memory management and automatic shared memory management.

To enable manual shared memory management:

1. Set the `MEMORY_TARGET` initialization parameter to 0.
2. Set the `SGA_TARGET` initialization parameter to 0.

You must then set values for the various SGA components, as described in the following sections.

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.

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.

Oracle Database supports multiple block sizes in a database. If you create tablespaces with nonstandard block sizes, you must configure nonstandard block size buffers to accommodate these tablespaces. The standard block size is used for the `SYSTEM` tablespace. You specify the standard block size by setting the initialization parameter `DB_BLOCK_SIZE`. Legitimate values are from 2K to 32K.

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

The sizes and numbers of nonstandard block size buffers are specified by the following 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 cache for the corresponding block size.

Note:

- Platform-specific restrictions regarding the maximum block size apply, so some of these sizes might not be allowed on some platforms.
- A 32K block size is valid only on 64-bit platforms.

See Also: "Specifying Nonstandard Block Sizes for Tablespaces" on page 14-14

Example of Setting Block and Cache Sizes

```
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=1024M
DB_2K_CACHE_SIZE=256M
DB_8K_CACHE_SIZE=512M
```

In the preceding example, the parameter DB_BLOCK_SIZE sets the standard block size of the database to 4K. The size of the cache of standard block size buffers is 1024MB. Additionally, 2K and 8K caches are also configured, with sizes of 256MB and 512MB, respectively.

> **Note:** The DB_*n*K_CACHE_SIZE parameters cannot be used to size the cache for the standard block size. If the value of DB_BLOCK_SIZE is *n*K, it is invalid to set DB_*n*K_CACHE_SIZE. The size of the cache for the standard block size is always determined from the value of DB_CACHE_SIZE.

The cache has a limited size, so not all the data on disk can fit in the cache. When the cache is full, subsequent cache misses cause Oracle Database to write dirty data already in the cache to disk to make room for the new data. (If a buffer is not dirty, it does not need to be written to disk before a new block can be read into the buffer.) Subsequent access to any data that was written to disk and then overwritten results in additional cache misses.

The size of the cache affects the likelihood that a request for data results in a cache hit. If the cache is large, it is more likely to contain the data that is requested. Increasing the size of a cache increases the percentage of data requests that result in cache hits.

You can change the size of the buffer cache while the instance is running, without having to shut down the database. Do this with the ALTER SYSTEM statement.

Use the fixed view V$BUFFER_POOL to track the sizes of the different cache components and any pending resize operations.

Multiple Buffer Pools You can configure the database buffer cache with separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks. Particular schema objects (tables, clusters, indexes, and partitions) can then be assigned to the appropriate buffer pool to control the way their data blocks age out of the cache.

- The KEEP buffer pool retains the schema object's data blocks in memory.
- The RECYCLE buffer pool eliminates data blocks from memory as soon as they are no longer needed.
- The DEFAULT buffer pool contains data blocks from schema objects that are not assigned to any buffer pool, as well as schema objects that are explicitly assigned to the DEFAULT pool.

The initialization parameters that configure the KEEP and RECYCLE buffer pools are DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE.

> **Note:** Multiple buffer pools are only available for the standard block size. Nonstandard block size caches have a single DEFAULT pool.

See Also: *Oracle Database Performance Tuning Guide* for information about tuning the buffer cache and for more information about multiple buffer pools

Specifying the Shared Pool Size

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

In releases before Oracle Database 10*g* Release 1, the amount of shared pool memory that was allocated was equal to the value of the SHARED_POOL_SIZE initialization parameter plus the amount of internal SGA overhead computed during instance startup. The internal SGA overhead refers to memory that is allocated by Oracle Database during startup, based on the values of several other initialization parameters. This memory is used to maintain state for different server components in the SGA. For example, if the SHARED_POOL_SIZE parameter is set to 64MB and the internal SGA overhead is computed to be 12MB, the real size of the shared pool is 64+12=76MB, although the value of the SHARED_POOL_SIZE parameter is still displayed as 64MB.

Starting with Oracle Database 10*g* Release 1, the size of the internal SGA overhead is included in the user-specified value of SHARED_POOL_SIZE. If you are not using automatic memory management or automatic shared memory management, the amount of shared pool memory that is allocated at startup is equal to the value of the SHARED_POOL_SIZE initialization parameter, rounded up to a multiple of the granule size. You must therefore set this parameter so that it includes the internal SGA overhead in addition to the desired value for shared pool size. In the previous example, if the SHARED_POOL_SIZE parameter is set to 64MB at startup, then the available shared pool after startup is 64-12=52MB, assuming the value of internal SGA overhead remains unchanged. In order to maintain an effective value of 64MB for shared pool memory after startup, you must set the SHARED_POOL_SIZE parameter to 64+12=76MB.

When migrating from a release that is earlier than Oracle Database 10*g* Release 1, the Oracle Database 11*g* migration utilities recommend a new value for this parameter based on the value of internal SGA overhead in the pre-upgrade environment and based on the old value of this parameter. Beginning with Oracle Database 10*g*, the exact value of internal SGA overhead, also known as startup overhead in the shared pool, can be queried from the V$SGAINFO view. Also, in manual shared memory management mode, if the user-specified value of SHARED_POOL_SIZE is too small to accommodate even the requirements of internal SGA overhead, then Oracle Database generates an ORA-00371 error during startup, with a suggested value to use for the SHARED_POOL_SIZE parameter.

When you use automatic shared memory management in Oracle Database 11*g*, the shared pool is automatically tuned, and an ORA-00371 error would not be generated.

The Result Cache and Shared Pool Size The result cache takes its memory from the shared pool. Therefore, if you expect to increase the maximum size of the result cache, take this into consideration when sizing the shared pool.

See Also: "Specifying the Result Cache Maximum Size" on page 6-18

Specifying the Large Pool Size

The LARGE_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the large pool component of the SGA. The large pool is an optional component of the SGA. You must specifically set the LARGE_POOL_SIZE

parameter to create a large pool. Configuring the large pool is discussed in *Oracle Database Performance Tuning Guide*.

Specifying the Java Pool Size

The JAVA_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the java pool component of the SGA. Oracle Database selects an appropriate default value. Configuration of the java pool is discussed in *Oracle Database Java Developer's Guide*.

Specifying the Streams Pool Size

The STREAMS_POOL_SIZE initialization parameter is a dynamic parameter that lets you specify or adjust the size of the Streams Pool component of the SGA. If STREAMS_POOL_SIZE is set to 0, then the Oracle Streams product transfers memory from the buffer cache to the Streams Pool when it is needed. For details, see the discussion of the Streams Pool in *Oracle Streams Replication Administrator's Guide*.

Specifying the Result Cache Maximum Size

The RESULT_CACHE_MAX_SIZE initialization parameter is a dynamic parameter that enables you to specify the maximum size of the result cache component of the SGA. Typically, there is no need to specify this parameter, because the default maximum size is chosen by the database based on total memory available to the SGA and on the memory management method currently in use. You can view the current default maximum size by displaying the value of the RESULT_CACHE_MAX_SIZE parameter. To change this maximum size, you can set RESULT_CACHE_MAX_SIZE with an ALTER SYSTEM statement, or you can specify this parameter in the text initialization parameter file. In each case, the value is rounded up to the nearest multiple of 32K.

If RESULT_CACHE_MAX_SIZE is 0 upon instance startup, the result cache is disabled. To reenable it you must set RESULT_CACHE_MAX_SIZE to a nonzero value (or remove this parameter from the text initialization parameter file to get the default maximum size) and then restart the database.

Note that after starting the database with the result cache disabled, if you use an ALTER SYSTEM statement to set RESULT_CACHE_MAX_SIZE to a nonzero value but do not restart the database, querying the value of the RESULT_CACHE_MAX_SIZE parameter returns a nonzero value even though the result cache is still disabled. The value of RESULT_CACHE_MAX_SIZE is therefore not the most reliable way to determine if the result cache is enabled. You can use the following query instead:

```
SELECT dbms_result_cache.status() FROM dual;

DBMS_RESULT_CACHE.STATUS()
------------------------------------------
ENABLED
```

The result cache takes its memory from the shared pool, so if you increase the maximum result cache size, consider also increasing the shared pool size.

The view V$RESULT_CACHE_STATISTICS and the PL/SQL package procedure DBMS_RESULT_CACHE.MEMORY_REPORT display information to help you determine the amount of memory currently allocated to the result cache.

The PL/SQL package function DBMS_RESULT_CACHE.FLUSH clears the result cache and releases all the memory back to the shared pool.

See Also:

- *Oracle Database Performance Tuning Guide* for more information about the result cache
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_RESULT_CACHE` package procedures and functions.
- *Oracle Database Reference* for more information about the `V$RESULT_CACHE_STATISTICS` view.
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on setting `RESULT_CACHE_MAX_SIZE` for a cluster database.

Specifying Miscellaneous SGA Initialization Parameters

You can set a few additional initialization parameters to control how the SGA uses memory.

Physical Memory The `LOCK_SGA` parameter, when set to `TRUE`, locks the entire SGA into physical memory. This parameter cannot be used with automatic memory management or automatic shared memory management.

SGA Starting Address The `SHARED_MEMORY_ADDRESS` and `HI_SHARED_MEMORY_ADDRESS` parameters specify the SGA's starting address at run time. These parameters are rarely used. For 64-bit platforms, `HI_SHARED_MEMORY_ADDRESS` specifies the high order 32 bits of the 64-bit address.

Extended Buffer Cache Mechanism The `USE_INDIRECT_DATA_BUFFERS` parameter enables the use of the extended buffer cache mechanism for 32-bit platforms that can support more than 4 GB of physical memory. On platforms that do not support this much physical memory, this parameter is ignored. This parameter cannot be used with automatic memory management or automatic shared memory management.

See Also:

- *Oracle Database Reference* for more information on these initialization parameters
- "Using Automatic Memory Management" on page 6-3
- "Using Automatic Shared Memory Management" on page 6-8

Using Automatic PGA Memory Management

By default, Oracle Database automatically and globally manages the total amount of memory dedicated to the instance PGA. You can control this amount by setting the initialization parameter `PGA_AGGREGATE_TARGET`. Oracle Database then tries to ensure that the total amount of PGA memory allocated across all database server processes and background processes never exceeds this target.

If you create your database with DBCA, you can specify a value for the total instance PGA. DBCA then sets the `PGA_AGGREGATE_TARGET` initialization parameters in the server parameter file (`SPFILE`) that it creates. If you do not specify the total instance PGA, DBCA chooses a reasonable default.

If you create the database with the CREATE DATABASE SQL statement and a text initialization parameter file, you can provide a value for PGA_AGGREGATE_TARGET. If you omit this parameter, the database chooses a default value.

With automatic PGA memory management, sizing of SQL work areas for all dedicated server sessions is automatic and all *_AREA_SIZE initialization parameters are ignored for these sessions. At any given time, the total amount of PGA memory available to active work areas on the instance is automatically derived from the parameter PGA_AGGREGATE_TARGET. This amount is set to the value of PGA_AGGREGATE_TARGET minus the PGA memory allocated for other purposes (for example, session memory). The resulting PGA memory is then allotted to individual active work areas based on their specific memory requirements.

There are dynamic performance views that provide PGA memory use statistics. Most of these statistics are enabled when PGA_AGGREGATE_TARGET is set.

- Statistics on allocation and use of work area memory can be viewed in the following dynamic performance views:

  ```
  V$SYSSTAT
  V$SESSTAT
  V$PGASTAT
  V$SQL_WORKAREA
  V$SQL_WORKAREA_ACTIVE
  ```

- The following three columns in the V$PROCESS view report the PGA memory allocated and used by an Oracle Database process:

  ```
  PGA_USED_MEM
  PGA_ALLOCATED_MEM
  PGA_MAX_MEM
  ```

Note: The automatic PGA memory management method applies to work areas allocated by both dedicated and shared server process. See *Oracle Database Concepts* for information about PGA memory allocation in dedicated and shared server modes.

See Also:

- *Oracle Database Reference* for information about views mentioned in this section
- *Oracle Database Performance Tuning Guide* for information about using these views

Using Manual PGA Memory Management

Oracle Database supports manual PGA memory management, in which you manually tune SQL work areas.

In releases earlier than Oracle Database 10*g*, the database administrator controlled the maximum size of SQL work areas by setting the following parameters: SORT_AREA_SIZE, HASH_AREA_SIZE, BITMAP_MERGE_AREA_SIZE and CREATE_BITMAP_AREA_SIZE. Setting these parameters is difficult, because the maximum work area size is ideally selected from the data input size and the total number of work areas active in the system. These two factors vary greatly from one work area to another and from one

time to another. Thus, the various `*_AREA_SIZE` parameters are difficult to tune under the best of circumstances.

For this reason, Oracle strongly recommends that you leave automatic PGA memory management enabled.

If you decide to tune SQL work areas manually, you must set the `WORKAREA_SIZE_POLICY` initialization parameter to `MANUAL`.

> **Note:** The initialization parameter `WORKAREA_SIZE_POLICY` is a session- and system-level parameter that can take only two values: `MANUAL` or `AUTO`. The default is `AUTO`. You can set `PGA_AGGREGATE_TARGET`, and then switch back and forth from auto to manual memory management mode. When `WORKAREA_SIZE_POLICY` is set to `AUTO`, your settings for `*_AREA_SIZE` parameters are ignored.

Configuring Database Smart Flash Cache

This section contains the following topics on configuring Database Smart Flash Cache (the flash cache):

- When to Configure the Flash Cache
- Sizing the Flash Cache
- Tuning Memory for the Flash Cache
- Flash Cache Initialization Parameters
- Flash Cache in an Oracle Real Applications Clusters Environment

> **See Also:** "Memory Architecture Overview" on page 6-2 for a description of the flash cache

When to Configure the Flash Cache

Consider adding the flash cache when all of the following are true:

- Your database is running on the Solaris or Oracle Linux operating systems. The flash cache is supported on these operating systems only.
- The Buffer Pool Advisory section of your Automatic Workload Repository (AWR) report or STATSPACK report indicates that doubling the size of the buffer cache would be beneficial.
- `db file sequential read` is a top wait event.
- You have spare CPU.

> **Note:** You cannot share the flash cache among multiple instances.

Sizing the Flash Cache

As a general rule, size the flash cache to be between 2 times and 10 times the size of the buffer cache. Any multiplier less than two would not provide any benefit. If you are using automatic shared memory management, make the flash cache between 2 times and 10 times the size of `SGA_TARGET`. Using 80% of the size of `SGA_TARGET` instead of the full size would also suffice for this calculation.

Tuning Memory for the Flash Cache

For each database block moved from the buffer cache to the flash cache, a small amount of metadata about the block is kept in the buffer cache. For a single instance database, the metadata consumes approximately 100 bytes. For an Oracle Real Application Clusters (Oracle RAC) database, it is closer to 200 bytes. You must therefore take this extra memory requirement into account when adding the flash cache.

- If you are managing memory manually, increase the size of the buffer cache by an amount approximately equal to the number of database blocks that fit into the flash cache multiplied by 100 (or 200 for Oracle RAC).
- If you are using automatic memory management, increase the size of `MEMORY_TARGET` using the algorithm described above. You may first have to increase the size of `MEMORY_MAX_TARGET`.
- If you are using automatic shared memory management, increase the size of `SGA_TARGET`.

Note: You can choose to not increase the buffer cache size to account for the flash cache. In this case, the effective size of the buffer cache is reduced. However, you can offset this loss by using a larger flash cache.

See Also: "About Memory Management" on page 6-1

Flash Cache Initialization Parameters

Table 6–3 describes the initialization parameters that you use to configure the flash cache.

Table 6–3 Flash Cache Initialization Parameters

Parameter	Description
`db_flash_cache_file`	Specifies the path and file name for the file to contain the flash cache, in either the operating system file system or an Oracle Automatic Storage Management disk group. If the file does not exist, the database creates it during startup. The file must reside on a flash disk device. If you configure the flash cache on a disk drive (spindle), performance may suffer. The following is an example of a valid value for `db_flash_cache_file`: `/dev/fioa1`
`db_flash_cache_size`	Specifies the size of the flash cache. Must be less than or equal to the physical memory size of the flash disk device. Expressed as *n*G, indicating the number of gigabytes (GB). For example, to specify a 16 GB flash cache, set `db_flash_cache_size` to 16G.

You can use `ALTER SYSTEM` to set `db_flash_cache_size` to zero to disable the flash cache. You can also use `ALTER SYSTEM` to set the flash cache back to its original size to reenable it. However, dynamically changing the size of the flash cache is not supported.

Flash Cache in an Oracle Real Applications Clusters Environment

You must configure a flash cache on either all or none of the instances in an Oracle Real Application Clusters environment.

Memory Management Reference

This section contains the following reference topics for memory management:

- Platforms That Support Automatic Memory Management
- Memory Management Data Dictionary Views

Platforms That Support Automatic Memory Management

The following platforms support automatic memory management—the Oracle Database ability to automatically tune the sizes of the SGA and PGA, redistributing memory from one to the other on demand to optimize performance:

- Linux
- Solaris
- Windows
- HP-UX
- AIX

Memory Management Data Dictionary Views

The following dynamic performance views provide information on memory management:

View	Description
`V$SGA`	Displays summary information about the system global area (SGA).
`V$SGAINFO`	Displays size information about the SGA, including the sizes of different SGA components, the granule size, and free memory.
`V$SGASTAT`	Displays detailed information about how memory is allocated within the shared pool, large pool, Java pool, and Streams pool.
`V$PGASTAT`	Displays PGA memory usage statistics as well as statistics about the automatic PGA memory manager when it is enabled (that is, when `PGA_AGGREGATE_TARGET` is set). Cumulative values in `V$PGASTAT` are accumulated since instance startup.
`V$MEMORY_DYNAMIC_COMPONENTS`	Displays information on the current size of all automatically tuned and static memory components, with the last operation (for example, grow or shrink) that occurred on each.
`V$SGA_DYNAMIC_COMPONENTS`	Displays the current sizes of all SGA components, and the last operation for each component.
`V$SGA_DYNAMIC_FREE_MEMORY`	Displays information about the amount of SGA memory available for future dynamic SGA resize operations.

View	Description
V$MEMORY_CURRENT_RESIZE_OPS	Displays information about resize operations that are currently in progress. A resize operation is an enlargement or reduction of the SGA, the instance PGA, or a dynamic SGA component.
V$SGA_CURRENT_RESIZE_OPS	Displays information about dynamic SGA component resize operations that are currently in progress.
V$MEMORY_RESIZE_OPS	Displays information about the last 800 completed memory component resize operations, including automatic grow and shrink operations for SGA_TARGET and PGA_AGGREGATE_TARGET.
V$SGA_RESIZE_OPS	Displays information about the last 800 completed SGA component resize operations.
V$MEMORY_TARGET_ADVICE	Displays information that helps you tune MEMORY_TARGET if you enabled automatic memory management.
V$SGA_TARGET_ADVICE	Displays information that helps you tune SGA_TARGET.
V$PGA_TARGET_ADVICE	Displays information that helps you tune PGA_AGGREGATE_TARGET.

See Also: *Oracle Database Reference* for detailed information on memory management views.

7

Managing Users and Securing the Database

In this chapter:

- The Importance of Establishing a Security Policy for Your Database
- Managing Users and Resources
- Managing User Privileges and Roles
- Auditing Database Use
- Predefined User Accounts

The Importance of Establishing a Security Policy for Your Database

It is important to develop a security policy for every database. The security policy establishes methods for protecting your database from accidental or malicious destruction of data or damage to the database infrastructure.

Each database can have an administrator, referred to as the security administrator, who is responsible for implementing and maintaining the database security policy If the database system is small, the database administrator can have the responsibilities of the security administrator. However, if the database system is large, a designated person or group of people may have sole responsibility as security administrator.

For information about establishing security policies for your database, see *Oracle Database Security Guide*.

Managing Users and Resources

To connect to the database, each user must specify a valid user name that has been previously defined to the database. An account must have been established for the user, with information about the user being stored in the data dictionary.

When you create a database user (account), you specify the following attributes of the user:

- User name
- Authentication method
- Default tablespace
- Temporary tablespace
- Other tablespaces and quotas
- User profile

To learn how to create and manage users, see *Oracle Database Security Guide*.

Managing User Privileges and Roles

Privileges and roles are used to control user access to data and the types of SQL statements that can be executed. The table that follows describes the three types of privileges and roles:

Type	Description
System privilege	A system-defined privilege usually granted only by administrators. These privileges allow users to perform specific database operations.
Object privilege	A system-defined privilege that controls access to a specific object.
Role	A collection of privileges and other roles. Some system-defined roles exist, but most are created by administrators. Roles group together privileges and other roles, which facilitates the granting of multiple privileges and roles to users.

Privileges and roles can be granted to other users by users who have been granted the privilege to do so. The granting of roles and privileges starts at the administrator level. At database creation, the administrative user `SYS` is created and granted all system privileges and predefined Oracle Database roles. User `SYS` can then grant privileges and roles to other users, and also grant those users the right to grant specific privileges to others.

To learn how to administer privileges and roles for users, see *Oracle Database Security Guide*.

Auditing Database Use

You can monitor and record selected user database actions, including those performed by administrators. There are several reasons why you might want to implement database auditing. Complete background information and instructions for database auditing are found in *Oracle Database Security Guide*.

Predefined User Accounts

Oracle Database includes several predefined user accounts. The three types of predefined accounts are:

- Administrative accounts (`SYS`, `SYSTEM`, `SYSMAN`, and `DBSNMP`)

  `SYS` and `SYSTEM` are described in "About Database Administrator Security and Privileges" on page 1-14. `SYSMAN` is used to perform Oracle Enterprise Manager administration tasks. The management agent of Enterprise Manager uses the `DBSNMP` account to monitor and manage the database. You must not delete these accounts.

- Sample schema accounts

  These accounts are used for examples in Oracle Database documentation and instructional materials. Examples are `HR`, `SH`, and `OE`. You must unlock these accounts and reset their passwords before using them.

- Internal accounts.

These accounts are created so that individual Oracle Database features or components can have their own schemas. You must not delete internal accounts, and you must not attempt to log in with them.

> **See Also:** *Oracle Database 2 Day + Security Guide* for a table of predefined accounts.

8

Monitoring Database Operations

It is important that you monitor the operation of your database on a regular basis. Doing so not only informs you of errors that have not yet come to your attention but also gives you a better understanding of the normal operation of your database. Being familiar with normal behavior in turn helps you recognize when something is wrong.

In this chapter:

- Monitoring Errors and Alerts
- Monitoring Performance

Monitoring Errors and Alerts

The following sections explain how to monitor database errors and alerts. It contains the following topics:

- Monitoring Errors with Trace Files and the Alert Log
- Monitoring Database Operations with Server-Generated Alerts

> **Note:** The easiest and best way to monitor the database for errors and alerts is with the Database Home page in Enterprise Manager. This section provides alternate methods for monitoring, using data dictionary views, PL/SQL packages, and other command-line facilities.

Monitoring Errors with Trace Files and the Alert Log

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, and other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

> **Note:** Critical errors also create incidents and incident dumps in the Automatic Diagnostic Repository. See Chapter 9, "Managing Diagnostic Data" on page 9-1 for more information.

The **alert log** is a chronological log of messages and errors, and includes the following items:

- All internal errors (ORA-00600), block corruption errors (ORA-01578), and deadlock errors (ORA-00060) that occur
- Administrative operations, such as CREATE, ALTER, and DROP statements and STARTUP, SHUTDOWN, and ARCHIVELOG statements
- 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 that had nondefault values at the time the database and instance start

Oracle Database uses the alert log to record these operations as an alternative to displaying the information on an operator's console (although some systems also display information on the console). If an operation is successful, a "completed" message is written in the alert log, along with a timestamp.

The alert log is maintained as both an XML-formatted file and a text-formatted file. You can view either format of the alert log with any text editor or you can use the ADRCI utility to view the XML-formatted version of the file with the XML tags stripped.

Check the alert log and trace files of an instance periodically to learn whether the background processes have encountered errors. For example, when the log writer process (LGWR) cannot write to a member of a log group, an error message indicating the nature of the problem is written to the LGWR trace file and the alert log. Such an error message means that a media or I/O problem has occurred and should be corrected immediately.

Oracle Database also writes values of initialization parameters to the alert log, in addition to other important statistics.

The alert log and all trace files for background and server processes are written to the Automatic Diagnostic Repository, the location of which is specified by the DIAGNOSTIC_DEST initialization parameter. 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:

- Chapter 9, "Managing Diagnostic Data" on page 9-1 for information on the Automatic Diagnostic Repository.
- "Alert Log" on page 9-5 for additional information about the alert log.
- "Viewing the Alert Log" on page 9-20
- *Oracle Database Utilities* for information on the ADRCI utility.
- 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 log) using the initialization parameter MAX_DUMP_FILE_SIZE, which limits the file to the specified number of operating system blocks. To control the size of an alert log, you must manually delete the file when you no longer need it. Otherwise the database continues to append to the file.

You can safely delete the alert log while the instance is running, although you should consider making an archived copy of it first. This archived copy could prove valuable if you should have a future problem that requires investigating the history of an instance.

Controlling When Oracle Database 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 behavior is described in "Controlling Trace Output Generated by the Archivelog Process" on page 13-13. Other background processes do not have this flexibility.

Trace files are written on behalf of server processes whenever critical 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 Automatic Diagnostic Repository.

Optionally, you can request that trace files be generated for server processes. 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;
```

Use the `DBMS_SESSION` or the `DBMS_MONITOR` packages to control SQL tracing for a session.

Caution: The SQL trace facility for server processes can cause significant system overhead resulting in severe performance impact, so you should enable this feature only when collecting statistics.

See Also:

- Chapter 9, "Managing Diagnostic Data" on page 9-1 for more information on how the database handles critical errors, otherwise known as "incidents."

Reading the Trace File for Shared Server Sessions

If shared server is enabled, each session using a dispatcher is routed to a shared server process, and trace information is written to the server 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 trace files. To help you, Oracle provides a command line utility program, `trcsess`, which consolidates all trace information pertaining to a user session in one place and orders the information by time.

See Also: *Oracle Database Performance Tuning Guide* for information about using the SQL trace facility and using `TKPROF` and `trcsess` to interpret the generated trace files

Monitoring Database Operations with Server-Generated Alerts

A server-generated alert is a notification from the Oracle Database server of an impending problem. The notification may contain suggestions for correcting the problem. Notifications are also provided when the problem condition has been cleared.

Alerts are automatically generated when a problem occurs or when data does not match expected values for metrics, such as the following:

- Physical Reads Per Second
- User Commits Per Second
- SQL Service Response Time

Server-generated alerts can be based on threshold levels or can issue simply because an event has occurred. Threshold-based alerts can be triggered at both threshold warning and critical levels. The value of these levels can be customer-defined or internal values, and some alerts have default threshold levels which you can change if appropriate. For example, by default a server-generated alert is generated for tablespace space usage when the percentage of space usage exceeds either the 85% warning or 97% critical threshold level. Examples of alerts not based on threshold levels are:

- `Snapshot Too Old`
- `Resumable Session Suspended`
- `Recovery Area Space Usage`

An alert message is sent to the predefined persistent queue `ALERT_QUE` owned by the user `SYS`. Oracle Enterprise Manager reads this queue and provides notifications about outstanding server alerts, and sometimes suggests actions for correcting the problem. The alerts are displayed on the Enterprise Manager Database Home page and can be configured to send email or pager notifications to selected administrators. If an alert cannot be written to the alert queue, a message about the alert is written to the Oracle Database alert log.

Background processes periodically flush the data to the Automatic Workload Repository to capture a history of metric values. The alert history table and `ALERT_QUE` are purged automatically by the system at regular intervals.

Setting and Retrieving Thresholds for Server-Generated Alerts

You can view and change threshold settings for the server alert metrics using the `SET_THRESHOLD` and `GET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package. Examples of using these procedures are provided in the following sections:

- Setting Threshold Levels
- Retrieving Threshold Information

Note: The most convenient way to set and retrieve threshold values is to use the graphical interface of Enterprise Manager. See *Oracle Database 2 Day DBA* for instructions.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_SERVER_ALERT` package

Setting Threshold Levels The following example shows how to set thresholds with the SET_THRESHOLD procedure for CPU time for each user call for an instance:

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, DBMS_SERVER_ALERT.OPERATOR_GE, '8000',
 DBMS_SERVER_ALERT.OPERATOR_GE, '10000', 1, 2, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
```

In this example, a warning alert is issued when CPU time exceeds 8000 microseconds for each user call and a critical alert is issued when CPU time exceeds 10,000 microseconds for each user call. The arguments include:

- CPU_TIME_PER_CALL specifies the metric identifier. For a list of support metrics, see *Oracle Database PL/SQL Packages and Types Reference*.
- The observation period is set to 1 minute. This period specifies the number of minutes that the condition must deviate from the threshold value before the alert is issued.
- The number of consecutive occurrences is set to 2. This number specifies how many times the metric value must violate the threshold values before the alert is generated.
- The name of the instance is set to inst1.
- The constant DBMS_ALERT.OBJECT_TYPE_SERVICE specifies the object type on which the threshold is set. In this example, the service name is main.regress.rdbms.dev.us.example.com.

Retrieving Threshold Information To retrieve threshold values, use the GET_THRESHOLD procedure. For example:

```
DECLARE
 warning_operator         BINARY_INTEGER;
 warning_value            VARCHAR2(60);
 critical_operator        BINARY_INTEGER;
 critical_value           VARCHAR2(60);
 observation_period       BINARY_INTEGER;
 consecutive_occurrences  BINARY_INTEGER;
BEGIN
 DBMS_SERVER_ALERT.GET_THRESHOLD(
 DBMS_SERVER_ALERT.CPU_TIME_PER_CALL, warning_operator, warning_value,
    critical_operator, critical_value, observation_period,
    consecutive_occurrences, 'inst1',
 DBMS_SERVER_ALERT.OBJECT_TYPE_SERVICE, 'main.regress.rdbms.dev.us.example.com');
 DBMS_OUTPUT.PUT_LINE('Warning operator:       ' || warning_operator);
 DBMS_OUTPUT.PUT_LINE('Warning value:          ' || warning_value);
 DBMS_OUTPUT.PUT_LINE('Critical operator:      ' || critical_operator);
 DBMS_OUTPUT.PUT_LINE('Critical value:         ' || critical_value);
 DBMS_OUTPUT.PUT_LINE('Observation_period:     ' || observation_period);
 DBMS_OUTPUT.PUT_LINE('Consecutive occurrences:' || consecutive_occurrences);
END;
/
```

You can also check specific threshold settings with the DBA_THRESHOLDS view. For example:

```
SELECT metrics_name, warning_value, critical_value, consecutive_occurrences
   FROM DBA_THRESHOLDS
   WHERE metrics_name LIKE '%CPU Time%';
```

Viewing Server-Generated Alerts

The easiest way to view server-generated alerts is by accessing the Database Home page of Enterprise Manager. The following discussion presents other methods of viewing these alerts.

If you use your own tool rather than Enterprise Manager to display alerts, you must subscribe to the ALERT_QUE, read the ALERT_QUE, and display an alert notification after setting the threshold levels for an alert. To create an agent and subscribe the agent to the ALERT_QUE, use the CREATE_AQ_AGENT and ADD_SUBSCRIBER procedures of the DBMS_AQADM package.

Next you must associate a database user with the subscribing agent, because only a user associated with the subscribing agent can access queued messages in the secure ALERT_QUE. You must also assign the enqueue privilege to the user. Use the ENABLE_DB_ACCESS and GRANT_QUEUE_PRIVILEGE procedures of the DBMS_AQADM package.

Optionally, you can register with the DBMS_AQ.REGISTER procedure to receive an asynchronous notification when an alert is enqueued to ALERT_QUE. The notification can be in the form of email, HTTP post, or PL/SQL procedure.

To read an alert message, you can use the DBMS_AQ.DEQUEUE procedure or OCIAQDeq call. After the message has been dequeued, use the DBMS_SERVER_ALERT.EXPAND_MESSAGE procedure to expand the text of the message.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_AQ, and DBMS_AQADM packages

Server-Generated Alerts Data Dictionary Views

The following data dictionary views provide information about server-generated alerts.

View	Description
DBA_THRESHOLDS	Lists the threshold settings defined for the instance
DBA_OUTSTANDING_ALERTS	Describes the outstanding alerts in the database
DBA_ALERT_HISTORY	Lists a history of alerts that have been cleared
V$ALERT_TYPES	Provides information such as group and type for each alert
V$METRICNAME	Contains the names, identifiers, and other information about the system metrics
V$METRIC	Contains system-level metric values
V$METRIC_HISTORY	Contains a history of system-level metric values

> **See Also:** *Oracle Database Reference* for information on static data dictionary views and dynamic performance views

Monitoring Performance

Monitoring database performance is covered in detail in *Oracle Database Performance Tuning Guide*. Here are some additional topics with details that are not covered in that guide:

- Monitoring Locks
- Monitoring Wait Events

- Performance Monitoring Data Dictionary Views

Monitoring Locks

Locks are mechanisms that prevent destructive interaction between transactions accessing the same resource. The resources can be either user objects, such as tables and rows, or system objects not visible to users, such as shared data structures in memory and data dictionary rows. Oracle Database automatically obtains and manages necessary locks when executing SQL statements, so you need not be concerned with such details. However, the database also lets you lock data manually.

A deadlock can occur when two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work. Oracle Database automatically detects deadlock situations and resolves them by rolling back one of the statements involved in the deadlock, thereby releasing one set of the conflicting row locks.

Oracle Database is designed to avoid deadlocks, and they are not common. Most often they occur when transactions explicitly override the default locking of the database. Deadlocks can affect the performance of your database, so Oracle provides some scripts and views that enable you to monitor locks.

The `utllockt.sql` script displays, in a tree fashion, the sessions in the system that are waiting for locks and the locks that they are waiting for. The location of this script file is operating system dependent.

A second script, `catblock.sql`, creates the lock views that `utllockt.sql` needs, so you must run it before running `utllockt.sql`.

See Also:

- "Performance Monitoring Data Dictionary Views" on page 8-7
- *Oracle Database Concepts* contains more information about locks.

Monitoring Wait Events

Wait events are statistics that are incremented by a server process to indicate that it had to wait for an event to complete before being able to continue processing. A session could wait for a variety of reasons, including waiting for more input, waiting for the operating system to complete a service such as a disk write, or it could wait for a lock or latch.

When a session is waiting for resources, it is not doing any useful work. A large number of waits is a source of concern. Wait event data reveals various symptoms of problems that might be affecting performance, such as latch contention, buffer contention, and I/O contention.

Oracle provides several views that display wait event statistics. A discussion of these views and their role in instance tuning is contained in *Oracle Database Performance Tuning Guide*.

Performance Monitoring Data Dictionary Views

This section lists some of the data dictionary views that you can use to monitor an Oracle Database instance. These views are general in their scope. Other views, more specific to a process, are discussed in the section of this book where the process is described.

View	Description
V$LOCK	Lists the locks currently held by Oracle Database and outstanding requests for a lock or latch
DBA_BLOCKERS	Displays a session if it is holding a lock on an object for which another session is waiting
DBA_WAITERS	Displays a session if it is waiting for a locked object
DBA_DDL_LOCKS	Lists all DDL locks held in the database and all outstanding requests for a DDL lock
DBA_DML_LOCKS	Lists all DML locks held in the database and all outstanding requests for a DML lock
DBA_LOCK	Lists all locks or latches held in the database and all outstanding requests for a lock or latch
DBA_LOCK_INTERNAL	Displays a row for each lock or latch that is being held, and one row for each outstanding request for a lock or latch
V$LOCKED_OBJECT	Lists all locks acquired by every transaction on the system
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 nonparent latches and summary statistics for parent latches

See Also: *Oracle Database Reference* for detailed descriptions of these views

9

Managing Diagnostic Data

Beginning with Release 11*g*, Oracle Database includes an advanced fault diagnosability infrastructure for collecting and managing diagnostic data. **Diagnostic data** includes the trace files, dumps, and core files that are also present in previous releases, plus new types of diagnostic data that enable customers and Oracle Support to identify, investigate, track, and resolve problems quickly and effectively.

In this chapter:

- About the Oracle Database Fault Diagnosability Infrastructure
- Investigating, Reporting, and Resolving a Problem
- Viewing Problems with the Enterprise Manager Support Workbench
- Creating a User-Reported Problem
- Viewing the Alert Log
- Finding Trace Files
- Running Health Checks with Health Monitor
- Repairing SQL Failures with the SQL Repair Advisor
- Repairing Data Corruptions with the Data Recovery Advisor
- Creating, Editing, and Uploading Custom Incident Packages

About the Oracle Database Fault Diagnosability Infrastructure

This section contains background information on the Oracle Database fault diagnosability infrastructure. It contains the following topics:

- Fault Diagnosability Infrastructure Overview
- About Incidents and Problems
- Fault Diagnosability Infrastructure Components
- Structure, Contents, and Location of the Automatic Diagnostic Repository

Fault Diagnosability Infrastructure Overview

The fault diagnosability infrastructure aids in preventing, detecting, diagnosing, and resolving problems. The problems that are targeted in particular are critical errors such as those caused by code bugs, metadata corruption, and customer data corruption.

When a critical error occurs, it is assigned an incident number, and diagnostic data for the error (such as trace files) are immediately captured and tagged with this number. The data is then stored in the Automatic Diagnostic Repository (ADR)—a file-based repository outside the database—where it can later be retrieved by incident number and analyzed.

The goals of the fault diagnosability infrastructure are the following:

- First-failure diagnosis
- Problem prevention
- Limiting damage and interruptions after a problem is detected
- Reducing problem diagnostic time
- Reducing problem resolution time
- Simplifying customer interaction with Oracle Support

The keys to achieving these goals are the following technologies:

- **Automatic capture of diagnostic data upon first failure**—For critical errors, the ability to capture error information at first-failure greatly increases the chance of a quick problem resolution and reduced downtime. An always-on memory-based tracing system proactively collects diagnostic data from many database components, and can help isolate root causes of problems. Such proactive diagnostic data is similar to the data collected by airplane "black box" flight recorders. When a problem is detected, alerts are generated and the fault diagnosability infrastructure is activated to capture and store diagnostic data. The data is stored in a repository that is outside the database (and therefore available when the database is down), and is easily accessible with command line utilities and Enterprise Manager.
- **Standardized trace formats**—Standardizing trace formats across all database components enables DBAs and Oracle Support personnel to use a single set of tools for problem analysis. Problems are more easily diagnosed, and downtime is reduced.
- **Health checks**—Upon detecting a critical error, the fault diagnosability infrastructure can run one or more health checks to perform deeper analysis of a critical error. Health check results are then added to the other diagnostic data collected for the error. Individual health checks look for data block corruptions, undo and redo corruption, data dictionary corruption, and more. As a DBA, you can manually invoke these health checks, either on a regular basis or as required.
- **Incident packaging service (IPS) and incident packages**—The IPS enables you to automatically and easily gather the diagnostic data—traces, dumps, health check reports, and more—pertaining to a critical error and package the data into a zip file for transmission to Oracle Support. Because all diagnostic data relating to a critical error are tagged with that error's incident number, you do not have to search through trace files and other files to determine the files that are required for analysis; the incident packaging service identifies the required files automatically and adds them to the zip file. Before creating the zip file, the IPS first collects diagnostic data into an intermediate logical structure called an incident package (package). Packages are stored in the Automatic Diagnostic Repository. If you choose to, you can access this intermediate logical structure, view and modify its contents, add or remove additional diagnostic data at any time, and when you are ready, create the zip file from the package. After these steps are completed, the zip file is ready to be uploaded to Oracle Support.

- **Data Recovery Advisor**—The Data Recovery Advisor integrates with database health checks and RMAN to display data corruption problems, assess the extent of each problem (critical, high priority, low priority), describe the impact of a problem, recommend repair options, conduct a feasibility check of the customer-chosen option, and automate the repair process.
- **SQL Test Case Builder**—For many SQL-related problems, obtaining a reproducible test case is an important factor in problem resolution speed. The SQL Test Case Builder automates the sometimes difficult and time-consuming process of gathering as much information as possible about the problem and the environment in which it occurred. After quickly gathering this information, you can upload it to Oracle Support to enable support personnel to easily and accurately reproduce the problem.

See Also:

- *Oracle Database Performance Tuning Guide* for more information on SQL Test Case Builder

About Incidents and Problems

To facilitate diagnosis and resolution of critical errors, the fault diagnosability infrastructure introduces two concepts for Oracle Database: problems and incidents.

A **problem** is a critical error in a database instance, Oracle Automatic Storage Management (Oracle ASM) instance, or other Oracle product or component. Critical errors manifest as internal errors, such as `ORA-00600`, or other severe errors, such as `ORA-07445` (operating system exception) or `ORA-04031` (out of memory in the shared pool). Problems are tracked in the ADR. Each problem has a *problem key*, which is a text string that describes the problem. It includes an error code (such as `ORA 600`) and in some cases, one or more error parameters.

An **incident** is a single occurrence of a problem. When a problem (critical error) occurs multiple times, an incident is created for each occurrence. Incidents are timestamped and tracked in the Automatic Diagnostic Repository (ADR). Each incident is identified by a numeric incident ID, which is unique within the ADR. When an incident occurs, the database:

- Makes an entry in the alert log.
- Sends an *incident alert* to Oracle Enterprise Manager (Enterprise Manager).
- Gathers first-failure diagnostic data about the incident in the form of dump files (incident dumps).
- Tags the incident dumps with the incident ID.
- Stores the incident dumps in an ADR subdirectory created for that incident.

Diagnosis and resolution of a critical error usually starts with an incident alert. Incident alerts are displayed on the Enterprise Manager Database Home page or Oracle Automatic Storage Management Home page. The Database Home page also displays in its Related Alerts section any critical alerts in the Oracle ASM instance or other Oracle products or components. After viewing an alert, you can then view the problem and its associated incidents with Enterprise Manager or with the ADRCI command-line utility.

The following sections provide more information about incidents and problems:

- Incident Flood Control
- Related Problems Across the Topology

See Also:

- "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18
- "Investigating, Reporting, and Resolving a Problem" on page 9-11
- "ADRCI Command-Line Utility" on page 9-7

Incident Flood Control

It is conceivable that a problem could generate dozens or perhaps hundreds of incidents in a short period of time. This would generate too much diagnostic data, which would consume too much space in the ADR and could possibly slow down your efforts to diagnose and resolve the problem. For these reasons, the fault diagnosability infrastructure applies *flood control* to incident generation after certain thresholds are reached. A **flood-controlled incident** is an incident that generates an alert log entry, is recorded in the ADR, but does not generate incident dumps. Flood-controlled incidents provide a way of informing you that a critical error is ongoing, without overloading the system with diagnostic data. You can choose to view or hide flood-controlled incidents when viewing incidents with Enterprise Manager or the ADRCI command-line utility.

Threshold levels for incident flood control are predetermined and cannot be changed. They are defined as follows:

- After five incidents occur for the same problem key in one hour, subsequent incidents for this problem key are flood-controlled. Normal (non-flood-controlled) recording of incidents for that problem key begins again in the next hour.
- After 25 incidents occur for the same problem key in one day, subsequent incidents for this problem key are flood-controlled. Normal recording of incidents for that problem key begins again on the next day.

In addition, after 50 incidents for the same problem key occur in one hour, or 250 incidents for the same problem key occur in one day, subsequent incidents for this problem key are not recorded at all in the ADR. In these cases, the database writes a message to the alert log indicating that no further incidents will be recorded. As long as incidents continue to be generated for this problem key, this message is added to the alert log every ten minutes until the hour or the day expires. Upon expiration of the hour or day, normal recording of incidents for that problem key begins again.

Related Problems Across the Topology

For any problem identified in a database instance, the diagnosability framework can identify related problems across the topology of your Oracle Database installation. In a single instance environment, a related problem could be identified in the local Oracle ASM instance. In an Oracle RAC environment, a related problem could be identified in any database instance or Oracle ASM instance on any other node. When investigating problems, you are able to view and gather information on any related problems.

A problem is related to the original problem if it occurs within a designated time period or shares the same execution context identifier. An **execution context identifier (ECID)** is a globally unique identifier used to tag and track a single call through the Oracle software stack, for example, a call to Oracle Fusion Middleware that then calls into Oracle Database to retrieve data. The ECID is typically generated in the middle tier and is passed to the database as an Oracle Call Interface (OCI) attribute. When a single call has failures on multiple tiers of the Oracle software stack, problems that are generated are tagged with the same ECID so that they can be correlated. You can then determine the tier on which the originating problem occurred.

Fault Diagnosability Infrastructure Components

The following are the key components of the fault diagnosability infrastructure:

- Automatic Diagnostic Repository (ADR)
- Alert Log
- Trace Files, Dumps, and Core Files
- Other ADR Contents
- Enterprise Manager Support Workbench
- ADRCI Command-Line Utility

Automatic Diagnostic Repository (ADR)

The ADR is a file-based repository for database diagnostic data such as traces, dumps, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products. Beginning with Release 11*g*, the database, Oracle Automatic Storage Management (Oracle ASM), the listener, and other Oracle products or components store all diagnostic data in the ADR. Each instance of each product stores diagnostic data underneath its own home directory within the ADR. For example, in an Oracle Real Application Clusters environment with shared storage and Oracle ASM, each database instance and each Oracle ASM instance has an ADR home directory. ADR's unified directory structure, consistent diagnostic data formats across products and instances, and a unified set of tools enable customers and Oracle Support to correlate and analyze diagnostic data across multiple instances.

Note: Beginning with Release 11*g* of Oracle Database, because all diagnostic data, including the alert log, are stored in the ADR, the initialization parameters `BACKGROUND_DUMP_DEST` and `USER_DUMP_DEST` are deprecated. They are replaced by the initialization parameter `DIAGNOSTIC_DEST`, which identifies the location of the ADR.

See Also: "Structure, Contents, and Location of the Automatic Diagnostic Repository" on page 9-7 for more information on the `DIAGNOSTIC_DEST` parameter and on ADR homes.

Alert Log

The alert log is an XML file that is a chronological log of database messages and errors. It is stored in the ADR and includes messages about the following:

- Critical errors (incidents)
- Administrative operations, such as starting up or shutting down the database, recovering the database, creating or dropping a tablespace, and others.
- Errors during automatic refresh of a materialized view
- Other database events

You can view the alert log in text format (with the XML tags stripped) with Enterprise Manager and with the ADRCI utility. There is also a text-formatted version of the alert log stored in the ADR for backward compatibility. However, Oracle recommends that any parsing of the alert log contents be done with the XML-formatted version, because the text format is unstructured and may change from release to release.

See Also:

- "ADRCI Command-Line Utility" on page 9-7
- "Viewing the Alert Log" on page 9-20

Trace Files, Dumps, and Core Files

Trace files, dumps, and core files contain diagnostic data that are used to investigate problems. They are stored in the ADR.

Trace Files Each server and background process can write to an associated trace file. Trace files are updated periodically over the life of the process and can contain information on the process environment, status, activities, and errors. In addition, when a process detects a critical error, it writes information about the error to its trace file. The SQL trace facility also creates trace files, which provide performance information on individual SQL statements. You can enable SQL tracing for a session or an instance.

Trace file names are platform-dependent. Typically, database background process trace file names contain the Oracle SID, the background process name, and the operating system process number, while server process trace file names contain the Oracle SID, the string "ora", and the operating system process number. The file extension is `.trc`. An example of a server process trace file name is orcl_ora_344.trc. Trace files are sometimes accompanied by corresponding trace map (`.trm`) files, which contain structural information about trace files and are used for searching and navigation.

Oracle Database includes tools that help you analyze trace files. For more information on application tracing, SQL tracing, and tracing tools, see *Oracle Database Performance Tuning Guide*.

See Also: "Finding Trace Files" on page 9-21

Dumps A dump is a specific type of trace file. A dump is typically a one-time output of diagnostic data in response to an event (such as an incident), whereas a trace tends to be continuous output of diagnostic data. When an incident occurs, the database writes one or more dumps to the incident directory created for the incident. Incident dumps also contain the incident number in the file name.

Core Files A core file contains a memory dump, in an all-binary, port-specific format. Core file names include the string "core" and the operating system process ID. Core files are useful to Oracle Support engineers only. Core files are not found on all platforms.

Other ADR Contents

In addition to files mentioned in the previous sections, the ADR contains health monitor reports, data repair records, SQL test cases, incident packages, and more. These components are described later in the chapter.

Enterprise Manager Support Workbench

The Enterprise Manager Support Workbench (Support Workbench) is a facility that enables you to investigate, report, and in some cases, repair problems (critical errors), all with an easy-to-use graphical interface. The Support Workbench provides a self-service means for you to gather first-failure diagnostic data, obtain a support request number, and upload diagnostic data to Oracle Support with a minimum of

effort and in a very short time, thereby reducing time-to-resolution for problems. The Support Workbench also recommends and provides easy access to Oracle advisors that help you repair SQL-related problems, data corruption problems, and more.

ADRCI Command-Line Utility

The ADR Command Interpreter (ADRCI) is a utility that enables you to investigate problems, view health check reports, and package first-failure diagnostic data, all within a command-line environment. You can then upload the package to Oracle Support. ADRCI also enables you to view the names of the trace files in the ADR, and to view the alert log with XML tags stripped, with and without content filtering.

For more information on ADRCI, see *Oracle Database Utilities*.

Structure, Contents, and Location of the Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a directory structure that is stored outside of the database. It is therefore available for problem diagnosis when the database is down.

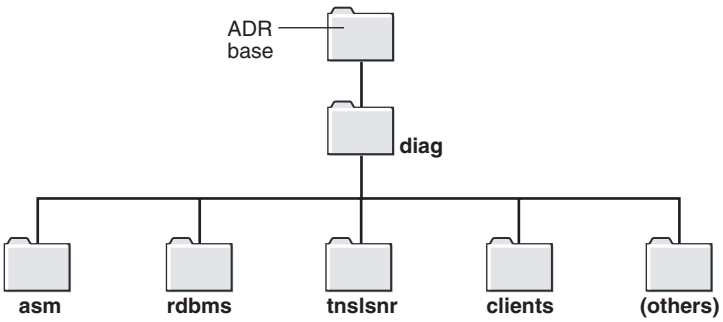
The ADR root directory is known as **ADR base**. Its location is set by the `DIAGNOSTIC_DEST` initialization parameter. If this parameter is omitted or left null, the database sets `DIAGNOSTIC_DEST` upon startup as follows:

- If environment variable `ORACLE_BASE` is set, `DIAGNOSTIC_DEST` is set to the directory designated by `ORACLE_BASE`.
- If environment variable `ORACLE_BASE` is not set, `DIAGNOSTIC_DEST` is set to *ORACLE_HOME*/log.

Within ADR base, there can be multiple ADR homes, where each ADR home is the root directory for all diagnostic data—traces, dumps, the alert log, and so on—for a particular instance of a particular Oracle product or component. For example, in an Oracle Real Application Clusters environment with Oracle ASM, each database instance, Oracle ASM instance, and listener has an ADR home.

ADR homes reside in ADR base subdirectories that are named according to the product or component type. Figure 9–1 illustrates these top-level subdirectories.

Figure 9–1 Product/Component Type Subdirectories in the ADR



The location of each ADR home is given by the following path, which starts at the ADR base directory:

```
diag/product_type/product_id/instance_id
```

As an example, Table 9–1 lists the values of the various path components for an Oracle Database instance.

Table 9–1 ADR Home Path Components for Oracle Database

Path Component	Value for Oracle Database
product_type	rdbms
product_id	*DB_UNIQUE_NAME*
instance_id	*SID*

For example, for a database with a SID and database unique name both equal to `orclbi`, the ADR home would be in the following location:

```
ADR_base/diag/rdbms/orclbi/orclbi/
```

Similarly, the ADR home path for the Oracle ASM instance in a single-instance environment would be:

```
ADR_base/diag/asm/+asm/+asm/
```

ADR Home Subdirectories

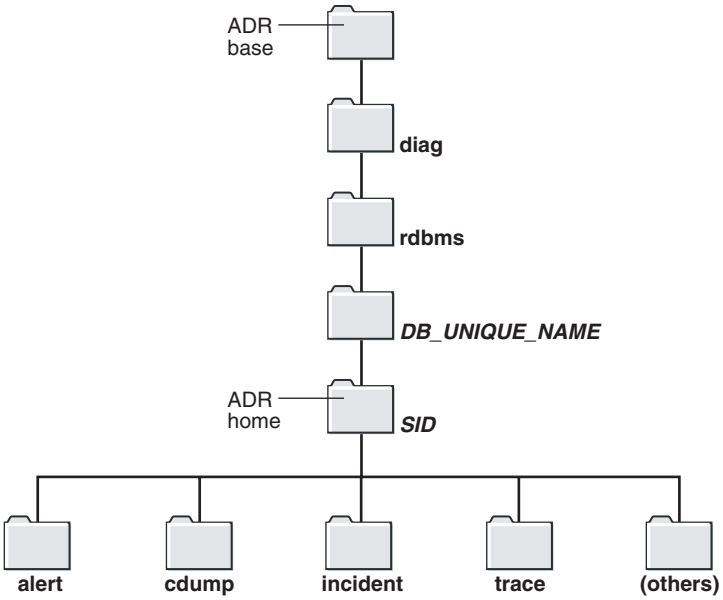
Within each ADR home directory are subdirectories that contain the diagnostic data. Table 9–2 lists some of these subdirectories and their contents.

Table 9–2 ADR Home Subdirectories

Subdirectory Name	Contents
alert	The XML-formatted alert log
cdump	Core files
incident	Multiple subdirectories, where each subdirectory is named for a particular incident, and where each contains dumps pertaining only to that incident
trace	Background and server process trace files, SQL trace files, and the text-formatted alert log
(others)	Other subdirectories of ADR home, which store incident packages, health monitor reports, and other information

Figure 9–2 illustrates the complete directory hierarchy of the ADR for a database instance.

Figure 9–2 ADR Directory Structure for a Database Instance



ADR in an Oracle Real Application Clusters Environment

In an Oracle Real Application Clusters (Oracle RAC) environment, each node can have ADR base on its own local storage, or ADR base can be set to a location on shared storage. You can use ADRCI to view aggregated diagnostic data from all instances on a single report.

ADR in Oracle Client

Each installation of Oracle Client includes an ADR for diagnostic data associated with critical failures in any of the Oracle Client components. The ADRCI utility is installed with Oracle Client so that you can examine diagnostic data and package it to enable it for upload to Oracle Support.

Viewing ADR Locations with the V$DIAG_INFO View

The `V$DIAG_INFO` view lists all important ADR locations for the current Oracle Database instance.

```
SELECT * FROM V$DIAG_INFO;

INST_ID NAME                 VALUE
------- -------------------- ------------------------------------------------------------
      1 Diag Enabled         TRUE
      1 ADR Base             /u01/oracle
      1 ADR Home             /u01/oracle/diag/rdbms/orclbi/orclbi
      1 Diag Trace           /u01/oracle/diag/rdbms/orclbi/orclbi/trace
      1 Diag Alert           /u01/oracle/diag/rdbms/orclbi/orclbi/alert
      1 Diag Incident        /u01/oracle/diag/rdbms/orclbi/orclbi/incident
      1 Diag Cdump           /u01/oracle/diag/rdbms/orclbi/orclbi/cdump
      1 Health Monitor       /u01/oracle/diag/rdbms/orclbi/orclbi/hm
      1 Default Trace File   /u01/oracle/diag/rdbms/orclbi/orclbi/trace/orcl_ora_22769.trc
      1 Active Problem Count 8
      1 Active Incident Count 20
```

The following table describes some of the information displayed by this view.

Table 9–3 Data in the V$DIAG_INFO View

Name	Description
ADR Base	Path of ADR base
ADR Home	Path of ADR home for the current database instance
Diag Trace	Location of background process trace files, server process trace files, SQL trace files, and the text-formatted version of the alert log
Diag Alert	Location of the XML-formatted version of the alert log
Default Trace File	Path to the trace file for the current session

Viewing Critical Errors with the V$DIAG_CRITICAL_ERROR View

The `V$DIAG_CRITICAL_ERROR` view lists all of the non-internal errors designated as critical errors for the current Oracle Database release. The view does not list internal errors because internal errors are always designated as critical errors.

The following example shows the output for the `V$DIAG_CRITICAL_ERROR` view in Oracle Database 11*g* Release 2 (11.2.0.2):

```
SELECT * FROM V$DIAG_CRITICAL_ERROR;

FACILITY   ERROR
---------- ----------------------------------------------------------------
ORA        7445
ORA        4030
ORA        4031
ORA        29740
ORA        255
ORA        355
ORA        356
ORA        239
ORA        240
ORA        494
ORA        3137
ORA        227
ORA        353
ORA        1578
ORA        32701
ORA        32703
ORA        29770
ORA        29771
ORA        445
ORA        25319
OCI        3106
OCI        3113
OCI        3135
```

The following table describes the information displayed by this view.

Table 9–4 Data in the V$DIAG_CRITICAL_ERROR View

Column	Description
`FACILITY`	The facility that can report the error, such as Oracle Database (ORA) or Oracle Call Interface (OCI)
`ERROR`	The error number

See Also: "About Incidents and Problems" on page 9-3 for more information about internal errors

Investigating, Reporting, and Resolving a Problem

This section describes how to use the Enterprise Manager Support Workbench (Support Workbench) to investigate and report a problem (critical error), and in some cases, resolve the problem. The section begins with a "roadmap" that summarizes the typical set of tasks that you must perform.

Note: The tasks described in this section are all Enterprise Manager–based. You can also accomplish all of these tasks (or their equivalents) with the `ADRCI` command-line utility, with PL/SQL packages such as `DBMS_HM` and `DBMS_SQLDIAG`, and with other software tools. See *Oracle Database Utilities* for more information on the `ADRCI` utility, and see *Oracle Database PL/SQL Packages and Types Reference* for information on PL/SQL packages.

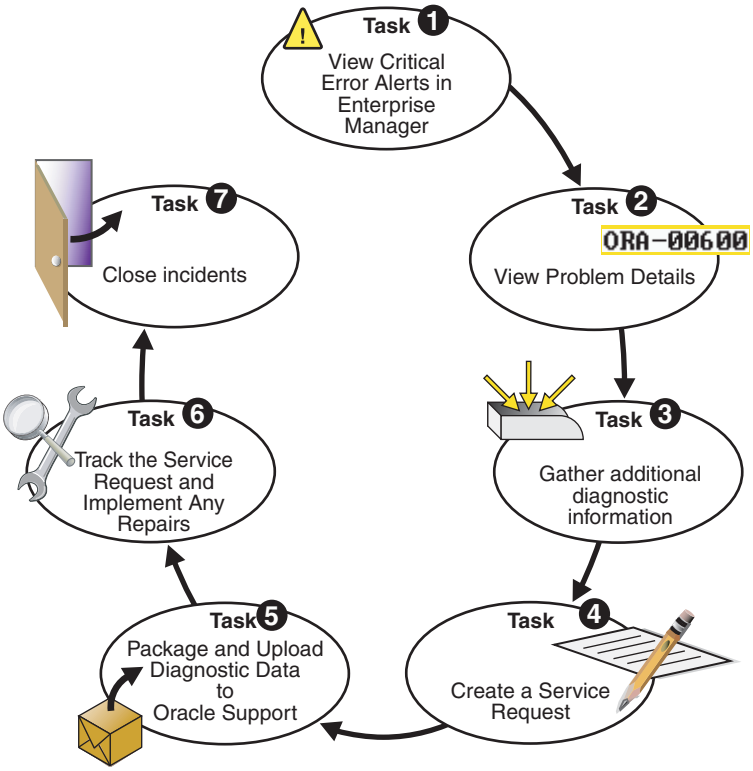
See Also: "About the Oracle Database Fault Diagnosability Infrastructure" on page 9-1 for more information on problems and their diagnostic data

Roadmap—Investigating, Reporting, and Resolving a Problem

You can begin investigating a problem by starting from the Support Workbench home page in Enterprise Manager. However, the more typical workflow begins with a critical error alert on the Database Home page. This section provides an overview of that workflow.

Figure 9–3 illustrates the tasks that you complete to investigate, report, and in some cases, resolve a problem.

Figure 9–3 Workflow for Investigating, Reporting, and Resolving a Problem



The following are task descriptions. Subsequent sections provide details for each task.

- **Task 1 – View Critical Error Alerts in Enterprise Manager** on page 9-13

  Start by accessing the Database Home page in Enterprise Manager, and reviewing critical error alerts. Select an alert for which to view details, and then go to the Problem Details page.

- **Task 2 –View Problem Details** on page 9-14

  Examine the problem details and view a list of all incidents that were recorded for the problem. Display findings from any health checks that were automatically run.

- **Task 3 – (Optional) Gather Additional Diagnostic Information** on page 9-14

  Optionally run additional health checks or other diagnostics. For SQL-related errors, optionally invoke the SQL Test Case Builder, which gathers all required data related to a SQL problem and packages the information in a way that enables the problem to be reproduced at Oracle Support.

- **Task 4 – (Optional) Create a Service Request** on page 9-14

  Optionally create a service request with My Oracle Support and record the service request number with the problem information. If you skip this step, you can create a service request later, or the Support Workbench can create one for you.

- **Task 5 – Package and Upload Diagnostic Data to Oracle Support** on page 9-15

  Invoke a guided workflow (a *wizard*) that automatically packages the gathered diagnostic data for a problem and uploads the data to Oracle Support.

- **Task 6 – Track the Service Request and Implement Any Repairs** on page 9-16

  Optionally maintain an activity log for the service request in the Support Workbench. Run Oracle advisors to help repair SQL failures or corrupted data.

- **Task 7 – Close Incidents** on page 9-17

  Set status for one, some, or all incidents for the problem to Closed.

  See Also: "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18

Task 1 – View Critical Error Alerts in Enterprise Manager

You begin the process of investigating problems (critical errors) by reviewing critical error alerts on the Database Home page or Oracle Automatic Storage Management Home page.

To view critical error alerts:

1. Access the Database Home page in Enterprise Manager.

   For Oracle Enterprise Manager Database Control, see *Oracle Database 2 Day DBA* for instructions. For Oracle Enterprise Manager Grid Control, go to the desired database target.

2. Do one of the following to view critical error alerts. (Critical error alerts are indicated by a red × in the Severity column, and the text "Incident" in the Category column.)

   - View the alerts in the Alerts section.

     You may have to click the hide/show icon next to the Alerts heading to display the alerts.

   - View the alerts in the Related Alerts section.

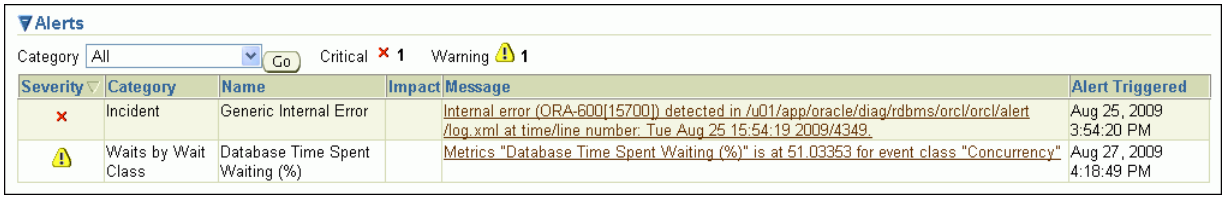
     The Target Name indicates the Oracle product or component that experienced the critical error. See "Related Problems Across the Topology" on page 9-4 for more information.

   - View critical alerts for the Oracle ASM instance by completing these steps:

     a. In the General section, click the link next to the label **ASM**.

     b. On the Oracle Automatic Storage Management Home page, scroll down to view the Alerts section.

Figure 9–4 Alerts Section of the Database Home Page



3. In the Message column, click the message of the critical error alert that you want investigate.

   The Incident page or Data Failure page appears. This page includes:

   - Problem information, including the number of incidents for the problem
   - A Performance and Critical Error graphical timeline for the 24-hour period in which the critical error occurred (database instance only).
   - Alert details, including severity, timestamp, and message

- Controls that enable you to clear the alert or record a comment about it.

4. Review the Performance and Critical Error graphical timeline if present, and note any time correlation between performance issues and the critical error. Optionally clear the alert or leave a comment about it.

Task 2 –View Problem Details

You continue your investigation by viewing the Problem Details page.

To view problem details:

1. On the Incident page or Data Failure page, click **View Problem Details**.

   The Problem Details page appears, showing the Incidents subpage.

2. (Optional) Do one or both of the following:

   - In the Investigate and Resolve section, on the Self Service tab, under Diagnose, click **Related Problems Across Topology**.

     A page appears showing any related problems in the local Oracle Automatic Storage Management (Oracle ASM) instance, or in the database or Oracle ASM instances on other nodes in an Oracle Real Application Clusters environment. This step is recommended if any critical alerts appear in the Related Alerts section on the Enterprise Manager Database Home page.

     See "Related Problems Across the Topology" on page 9-4 for more information.

   - View incident details by completing these steps:

     a. In the Incidents subpage, select an incident, and then click **View**.

     b. (Optional) On the Incident Details page, click **Checker Findings** to view the Checker Findings subpage.

        This page displays findings from any health checks that were automatically run when the critical error was detected.

        See "Running Health Checks with Health Monitor" on page 9-21 for more information.

Task 3 – (Optional) Gather Additional Diagnostic Information

You can perform the following activities to gather additional diagnostic information for a problem. This additional information is then automatically included in the diagnostic data that is uploaded to Oracle Support. If you are unsure about performing these activities, then check with your Oracle Support representative.

- Manually invoke additional health checks

  See "Running Health Checks with Health Monitor" on page 9-21

- Invoke the SQL Test Case Builder

  See *Oracle Database Performance Tuning Guide* for instructions.

Task 4 – (Optional) Create a Service Request

At this point, you can create an Oracle Support service request and record the service request number with the problem information. If you choose to skip this task, the Support Workbench will automatically create a draft service request for you in Task 5.

To create a service request:

1. On the Problem Details page, in the Investigate and Resolve section, click **Go to My Oracle Support and Research**.

   The My Oracle Support Sign In and Registration page appears in a new browser window.

   > **Note:** See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions for returning to the Problem Details page if you are not already there.

2. Log in to My Oracle Support and create a service request in the usual manner.

   (Optional) Remember the service request number (SR#) for the next step.

3. (Optional) Return to the Problem Details page, and then do the following:

   a. In the Summary section, click the **Edit** button that is adjacent to the SR# label.

   b. Enter the SR#, and then click **OK**.

   The SR# is recorded in the Problem Details page. This is for your reference only.

Task 5 – Package and Upload Diagnostic Data to Oracle Support

For this task, you use the quick packaging process of the Support Workbench to package and upload the diagnostic information for the problem to Oracle Support. Quick packaging has a minimum of steps, organized in a guided workflow (a wizard). The wizard assists you with creating an incident package (package) for a single problem, creating a zip file from the package, and uploading the file. With quick packaging, you are not able to edit or otherwise customize the diagnostic information that is uploaded. However, quick packaging is the more direct, straightforward method to package and upload diagnostic data.

To edit or remove sensitive data from the diagnostic information, enclose additional user files (such as application configuration files or scripts), or perform other customizations before uploading, you must use the custom packaging process, which is a more manual process and has more steps. See "Creating, Editing, and Uploading Custom Incident Packages" on page 9-32 for instructions. If you choose to follow those instructions instead of the instructions here in Task 5, do so now and then continue with Task 6 – Track the Service Request and Implement Any Repairs on page 9-16 when you are finished.

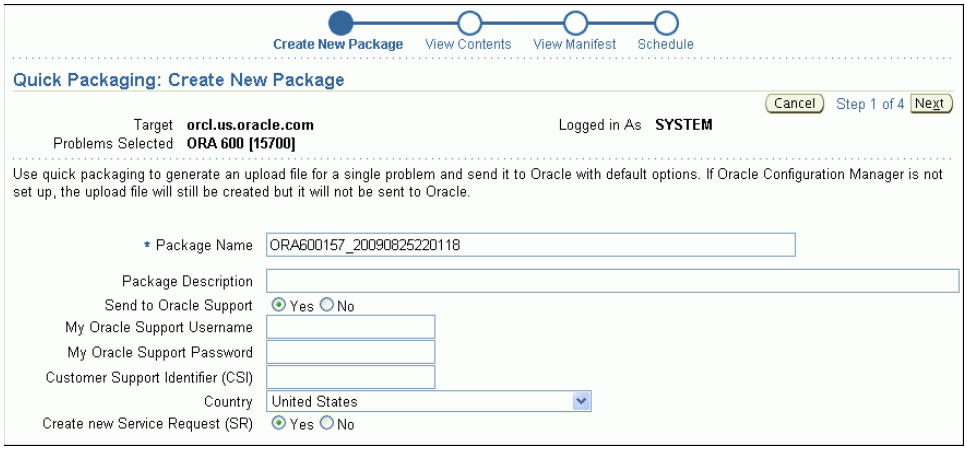
> **Note:** The Support Workbench uses Oracle Configuration Manager to upload the diagnostic data. If Oracle Configuration Manager is not installed or properly configured, the upload may fail. In this case, a message is displayed with a request that you upload the file to Oracle Support manually. You can upload manually with My Oracle Support.
>
> For more information about Oracle Configuration Manager, see *Oracle Configuration Manager Installation and Administration Guide*.

To package and upload diagnostic data to Oracle Support:

1. On the Problem Details page, in the Investigate and Resolve section, click **Quick Package**.

   The Create New Package page of the Quick Packaging wizard appears.

Note: See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions for returning to the Problem Details page if you are not already there.



2. Optionally enter a package name and description.
3. Fill in the remaining fields on the page. If you already created a service request for this problem, select **No** next to Create new Service Request (SR).

   If you select Yes, the Quick Packaging wizard creates a draft service request on your behalf. You must later log in to My Oracle Support and fill in the details of the service request.
4. Click **Next**, and then proceed with the remaining pages of the Quick Packaging wizard.

   When the Quick Packaging wizard is complete, the package that it creates remains available in the Support Workbench. You can then modify it with custom packaging operations (such as adding new incidents) and reupload at a later time. See "Viewing and Modifying Incident Packages" on page 9-39.

Task 6 – Track the Service Request and Implement Any Repairs

After uploading diagnostic information to Oracle Support, you might perform various activities to track the service request, to collect additional diagnostic information, and to implement repairs. Among these activities are the following:

- Adding an Oracle bug number to the problem information.

  To do so, on the Problem Details page, click the **Edit** button that is adjacent to the Bug# label. This is for your reference only.
- Adding comments to the problem activity log.

  You may want to do this to share problem status or history information with other DBAs in your organization. For example you could record the results of your conversations with Oracle Support. To add comments, complete the following steps:

  1. Access the Problem Details page for the problem, as described in "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18.
  2. Click **Activity Log** to display the Activity Log subpage.

3. In the Comment field, enter a comment, and then click **Add Comment**.

   Your comment is recorded in the activity log.

- As new incidents occur, adding them to the package and reuploading.

  For this activity, you must use the custom packaging method described in "Creating, Editing, and Uploading Custom Incident Packages" on page 9-32.

- Running health checks.

  See "Running Health Checks with Health Monitor" on page 9-21.

- Running a suggested Oracle advisor to implement repairs.

  Access the suggested advisor in one of the following ways:

  – **Problem Details page**—In the Self-Service tab of the Investigate and Resolve section

  – **Support Workbench home page**—on the Checker Findings subpage

  – **Incident Details page**—on the Checker Findings subpage

  Table 9–5 lists the advisors that help repair critical errors.

Table 9–5 Oracle Advisors that Help Repair Critical Errors

Advisor	Critical Errors Addressed	See
Data Recovery Advisor	Corrupted blocks, corrupted or missing files, and other data failures	"Repairing Data Corruptions with the Data Recovery Advisor" on page 9-31
SQL Repair Advisor	SQL statement failures	"Repairing SQL Failures with the SQL Repair Advisor" on page 9-28

See Also: "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions for viewing the Checker Findings subpage of the Incident Details page

Task 7 – Close Incidents

When a particular incident is no longer of interest, you can close it. By default, closed incidents are not displayed on the Problem Details page.

All incidents, whether closed or not, are purged after 30 days. You can disable purging for an incident on the Incident Details page.

To close incidents:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. Select the desired problem, and then click **View**.

   The Problem Details page appears.

3. Select the incidents to close and then click **Close**.

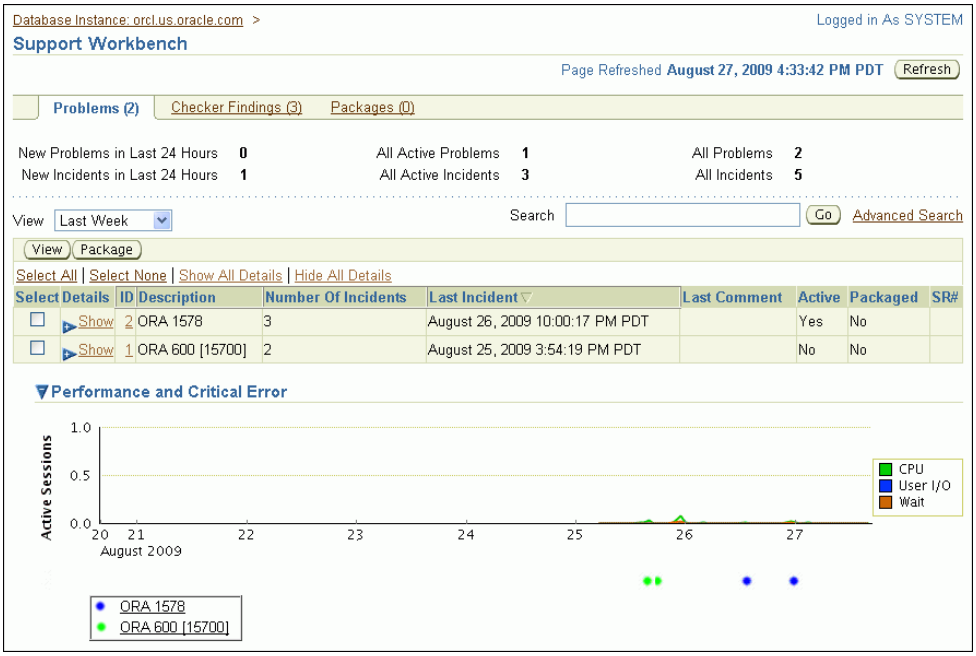
   A confirmation page appears.

4. Enter an optional comment and click **OK**.

Viewing Problems with the Enterprise Manager Support Workbench

You use the Enterprise Manager Support Workbench home page (Figure 9–5 on page 9-18) to view all problems or only those within a specified time period.

Figure 9–5 Enterprise Manager Support Workbench Home Page



To access the Support Workbench home page (database or Oracle ASM):

1. Access the Database Home page in Enterprise Manager.

   See *Oracle Database 2 Day DBA* for the instructions for Oracle Enterprise Manager Database Control. For Oracle Enterprise Manager Grid Control, go to the desired database target.

2. Do one of the following:

   - In the Diagnostic Summary section, click the numeric link next to the label **Active Incidents**.
   - At the top of the page, click **Software and Support**, and then under Support, click **Support Workbench**.

   The Support Workbench home page for the database instance appears, showing the Problems subpage. By default the problems from the last 24 hours are displayed.

3. To view the Support Workbench home page for the Oracle ASM instance, click the link **Support Workbench (+ASM_*hostname*)** in the Related Links section.

To view problems and incidents:

1. On the Support Workbench home page, select the desired time period from the View list. To view all problems, select **All**.

2. (Optional) If the Performance and Critical Error section is hidden, click the **Show/Hide** icon adjacent to the section heading to show the section.

This section enables you to view any correlation between performance changes and incident occurrences.

3. (Optional) Under the Details column, click **Show** to display a list of all incidents for a problem, and then click an incident ID to display the Incident Details page.

To view details for a particular problem:

1. On the Support Workbench home page, select the problem, and then click **View**.

   The Problem Details page appears, showing the Incidents subpage. The incidents subpage shows all incidents that are open and that generated dumps—that is, that were not flood-controlled.

2. (Optional) To view both open and closed incidents, select **All Incidents** in the Status list. To view both normal and flood-controlled incidents, select **All Incidents** in the Data Dumped list.

3. (Optional) To view details for an incident, select the incident, and then click **View**.

   The Incident Details page appears.

4. (Optional) On the Incident Details page, to view checker findings for the incident, click **Checker Findings**.

5. (Optional) On the Incident Details page, to view the user actions that are available to you for the incident, click **Additional Diagnostics**. Each user action provides a way for you to gather additional diagnostics for the incident or its problem.

See Also: "Incident Flood Control" on page 9-4

Creating a User-Reported Problem

System-generated problems—critical errors generated internally to the database—are automatically added to the Automatic Diagnostic Repository (ADR) and tracked in the Support Workbench. From the Support Workbench, you can gather additional diagnostic data on these problems, upload diagnostic data to Oracle Support, and in some cases, resolve the problems, all with the easy-to-use workflow that is explained in "Investigating, Reporting, and Resolving a Problem" on page 9-11.

There may be a situation in which you want to manually add a problem that you noticed to the ADR so that you can put that problem through that same workflow. An example of such a situation might be a global database performance problem that was not diagnosed by Automatic Diagnostic Database Monitor (ADDM). The Support Workbench includes a mechanism for you to create and work with such a user-reported problem.

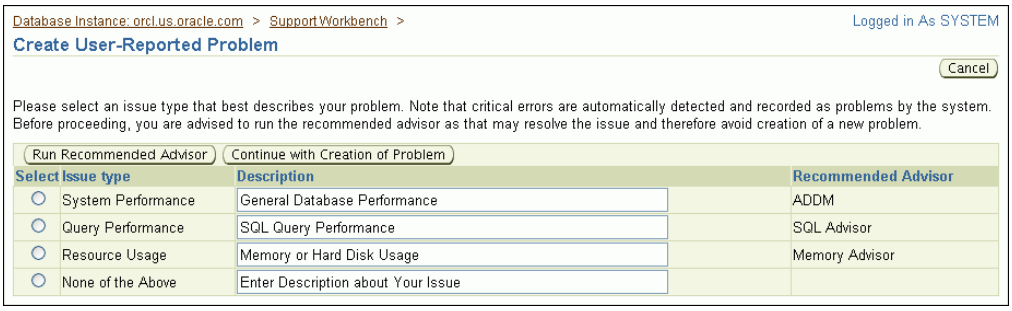
To create a user-reported problem:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. Under Related Links, click **Create User-Reported Problem**.

   The Create User-Reported Problem page appears.



3. If your problem matches one of the listed issue types, select the issue type, and then click **Run Recommended Advisor** to attempt to solve the problem with an Oracle advisor.

4. If the recommended advisor did not solve the problem, or if you did not run an advisor, do one of the following:

   - If your problem matches one of the listed issue types, select the issue type, and then click **Continue with Creation of Problem**.
   - If your problem does not match one of the listed issue types, select the issue type **None of the Above**, enter a description, and then click **Continue with Creation of Problem**.

   The Problem Details page appears.

5. Follow the instructions on the Problem Details page.

   See "Investigating, Reporting, and Resolving a Problem" on page 9-11 for more information.

   See Also: "About the Oracle Database Fault Diagnosability Infrastructure" on page 9-1 for more information on problems and the ADR

Viewing the Alert Log

You can view the alert log with a text editor, with Enterprise Manager, or with the ADRCI utility.

To view the alert log with Enterprise Manager:

1. Access the Database Home page in Enterprise Manager.

   For Oracle Enterprise Manager Database Control, see *Oracle Database 2 Day DBA* for instructions. For Oracle Enterprise Manager Grid Control, go to the desired database target.

2. Under Related Links, click **Alert Log Contents**.

   The View Alert Log Contents page appears.

3. Select the number of entries to view, and then click **Go**.

To view the alert log with a text editor:

1. Connect to the database with SQL*Plus or another query tool, such as SQL Developer.

2. Query the `V$DIAG_INFO` view as shown in "Viewing ADR Locations with the V$DIAG_INFO View" on page 9-9.

3. To view the text-only alert log, without the XML tags, complete these steps:
   a. In the `V$DIAG_INFO` query results, note the path that corresponds to the `Diag Trace` entry, and change directory to that path.
   b. Open file alert_*SID*.log with a text editor.
4. To view the XML-formatted alert log, complete these steps:
   a. In the `V$DIAG_INFO` query results, note the path that corresponds to the `Diag Alert` entry, and change directory to that path.
   b. Open the file log.xml with a text editor.

   See Also: *Oracle Database Utilities* for information about using the ADRCI utility to view a text version of the alert log (with XML tags stripped) and to run queries against the alert log

Finding Trace Files

Trace files are stored in the Automatic Diagnostic Repository (ADR), in the `trace` directory under each ADR home. To help you locate individual trace files within this directory, you can use data dictionary views. For example, you can find the path to your current session's trace file or to the trace file for each Oracle Database process.

To find the trace file for your current session:

- Submit the following query:

```
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Default Trace File';
```

  The full path to the trace file is returned.

To find all trace files for the current instance:

- Submit the following query:

```
SELECT VALUE FROM V$DIAG_INFO WHERE NAME = 'Diag Trace';
```

  The path to the ADR trace directory for the current instance is returned.

To determine the trace file for each Oracle Database process:

- Submit the following query:

```
SELECT PID, PROGRAM, TRACEFILE FROM V$PROCESS;
```

  See Also:

  - "Structure, Contents, and Location of the Automatic Diagnostic Repository" on page 9-7
  - The ADRCI `SHOW TRACEFILE` command in *Oracle Database Utilities*

Running Health Checks with Health Monitor

This section describes the Health Monitor and includes instructions on how to use it. The following topics are covered:

- About Health Monitor
- Running Health Checks Manually
- Viewing Checker Reports

- Health Monitor Views
- Health Check Parameters Reference

About Health Monitor

Beginning with Release 11*g*, Oracle Database includes a framework called Health Monitor for running diagnostic checks on the database.

About Health Monitor Checks

Health Monitor checks (also known as checkers, health checks, or checks) examine various layers and components of the database. Health checks detect file corruptions, physical and logical block corruptions, undo and redo corruptions, data dictionary corruptions, and more. The health checks generate reports of their findings and, in many cases, recommendations for resolving problems. Health checks can be run in two ways:

- **Reactive**—The fault diagnosability infrastructure can run health checks automatically in response to a critical error.
- **Manual**—As a DBA, you can manually run health checks using either the `DBMS_HM` PL/SQL package or the Enterprise Manager interface. You can run checkers on a regular basis if desired, or Oracle Support may ask you to run a checker while working with you on a service request.

Health Monitor checks store findings, recommendations, and other information in the Automatic Diagnostic Repository (ADR).

Health checks can run in two modes:

- **DB-online** mode means the check can be run while the database is open (that is, in `OPEN` mode or `MOUNT` mode).
- **DB-offline** mode means the check can be run when the instance is available but the database itself is closed (that is, in `NOMOUNT` mode).

All the health checks can be run in DB-online mode. Only the Redo Integrity Check and the DB Structure Integrity Check can be used in DB-offline mode.

See Also: "Automatic Diagnostic Repository (ADR)" on page 9-5

Types of Health Checks

Health monitor runs the following checks:

- **DB Structure Integrity Check**—This check verifies the integrity of database files and reports failures if these files are inaccessible, corrupt or inconsistent. If the database is in mount or open mode, this check examines the log files and datafiles listed in the control file. If the database is in `NOMOUNT` mode, only the control file is checked.
- **Data Block Integrity Check**—This check detects disk image block corruptions such as checksum failures, head/tail mismatch, and logical inconsistencies within the block. Most corruptions can be repaired using Block Media Recovery. Corrupted block information is also captured in the `V$DATABASE_BLOCK_CORRUPTION` view. This check does not detect inter-block or inter-segment corruption.

- **Redo Integrity Check**—This check scans the contents of the redo log for accessibility and corruption, as well as the archive logs, if available. The Redo Integrity Check reports failures such as archive log or redo corruption.
- **Undo Segment Integrity Check**—This check finds logical undo corruptions. After locating an undo corruption, this check uses PMON and SMON to try to recover the corrupted transaction. If this recovery fails, then Health Monitor stores information about the corruption in `V$CORRUPT_XID_LIST`. Most undo corruptions can be resolved by forcing a commit.
- **Transaction Integrity Check**—This check is identical to the Undo Segment Integrity Check except that it checks only one specific transaction.
- **Dictionary Integrity Check**—This check examines the integrity of core dictionary objects, such as `tab$` and `col$`. It performs the following operations:
  - Verifies the contents of dictionary entries for each dictionary object.
  - Performs a cross-row level check, which verifies that logical constraints on rows in the dictionary are enforced.
  - Performs an object relationship check, which verifies that parent-child relationships between dictionary objects are enforced.

  The Dictionary Integrity Check operates on the following dictionary objects:

  `tab$`, `clu$`, `fet$`, `uet$`, `seg$`, `undo$`, `ts$`, `file$`, `obj$`, `ind$`, `icol$`, `col$`, `user$`, `con$`, `cdef$`, `ccol$`, `bootstrap$`, `objauth$`, `ugroup$`, `tsq$`, `syn$`, `view$`, `typed_view$`, `superobj$`, `seq$`, `lob$`, `coltype$`, `subcoltype$`, `ntab$`, `refcon$`, `opqtype$`, `dependency$`, `access$`, `viewcon$`, `icoldep$`, `dual$`, `sysauth$`, `objpriv$`, `defrole$`, and `ecol$`.

Running Health Checks Manually

Health Monitor provides two ways to run health checks manually:

- By using the `DBMS_HM` PL/SQL package
- By using the Enterprise Manager interface, found on the Checkers subpage of the Advisor Central page

Running Health Checks Using the DBMS_HM PL/SQL Package

The `DBMS_HM` procedure for running a health check is called `RUN_CHECK`. To call `RUN_CHECK`, supply the name of the check and a name for the run, as follows:

```
BEGIN
  DBMS_HM.RUN_CHECK('Dictionary Integrity Check', 'my_run');
END;
/
```

To obtain a list of health check names, run the following query:

```
SELECT name FROM v$hm_check WHERE internal_check='N';

NAME
----------------------------------------------------------------
DB Structure Integrity Check
Data Block Integrity Check
Redo Integrity Check
Transaction Integrity Check
Undo Segment Integrity Check
Dictionary Integrity Check
```

Most health checks accept input parameters. You can view parameter names and descriptions with the V$HM_CHECK_PARAM view. Some parameters are mandatory while others are optional. If optional parameters are omitted, defaults are used. The following query displays parameter information for all health checks:

```
SELECT c.name check_name, p.name parameter_name, p.type,
p.default_value, p.description
FROM v$hm_check_param p, v$hm_check c
WHERE p.check_id = c.id and c.internal_check = 'N'
ORDER BY c.name;
```

Input parameters are passed in the input_params argument as name/value pairs separated by semicolons (;). The following example illustrates how to pass the transaction ID as a parameter to the Transaction Integrity Check:

```
BEGIN
  DBMS_HM.RUN_CHECK (
   check_name   => 'Transaction Integrity Check',
   run_name     => 'my_run',
   input_params => 'TXN_ID=7.33.2');
END;
/
```

See Also:

- "Health Check Parameters Reference" on page 9-28
- *Oracle Database PL/SQL Packages and Types Reference* for more examples of using DBMS_HM.

Running Health Checks Using Enterprise Manager

Enterprise Manager provides an interface for running Health Monitor checkers.

To run a Health Monitor Checker using Enterprise Manager:

1. On the Database Home page, in the Related Links section, click **Advisor Central**.
2. Click **Checkers** to view the Checkers subpage.
3. In the Checkers section, click the checker you want to run.
4. Enter values for input parameters or, for optional parameters, leave them blank to accept the defaults.
5. Click **Run**, confirm your parameters, and click **Run** again.

Viewing Checker Reports

After a checker has run, you can view a report of its execution. The report contains findings, recommendations, and other information. You can view reports using Enterprise Manager, the ADRCI utility, or the DBMS_HM PL/SQL package. The following table indicates the report formats available with each viewing method.

Report Viewing Method	Report Formats Available
Enterprise Manager	HTML
DBMS_HM PL/SQL package	HTML, XML, and text
ADRCI utility	XML

Results of checker runs (findings, recommendations, and other information) are stored in the ADR, but reports are not generated immediately. When you request a report with the DBMS_HM PL/SQL package or with Enterprise Manager, if the report does not yet exist, it is first generated from the checker run data in the ADR, stored as a report file in XML format in the HM subdirectory of the ADR home for the current instance, and then displayed. If the report file already exists, it is just displayed. When using the ADRCI utility, you must first run a command to generate the report file if it does not exist, and then run another command to display its contents.

The preferred method to view checker reports is with Enterprise Manager. The following sections provide instructions for all methods:

- Viewing Reports Using Enterprise Manager
- Viewing Reports Using DBMS_HM
- Viewing Reports Using the ADRCI Utility

See Also: "Automatic Diagnostic Repository (ADR)" on page 9-5

Viewing Reports Using Enterprise Manager

You can also view Health Monitor reports and findings for a given checker run using Enterprise Manager.

To view run findings using Enterprise Manager

1. Access the Database Home page.

   For Oracle Enterprise Manager Database Control, see *Oracle Database 2 Day DBA* for instructions. For Oracle Enterprise Manager Grid Control, go to the desired database target.

2. In the Related Links section, click **Advisor Central**.
3. Click **Checkers** to view the Checkers subpage.
4. Click the run name for the checker run that you want to view.

   The Run Detail page appears, showing the findings for that checker run.

5. Click **Runs** to display the Runs subpage.

   Enterprise Manager displays more information about the checker run.

6. Click **View Report** to view the report for the checker run.

   The report is displayed in a new browser window.

Viewing Reports Using DBMS_HM

You can view Health Monitor checker reports with the DBMS_HM package function GET_RUN_REPORT. This function enables you to request HTML, XML, or text formatting. The default format is text, as shown in the following SQL*Plus example:

```
SET LONG 100000
SET LONGCHUNKSIZE 1000
SET PAGESIZE 1000
SET LINESIZE 512
SELECT DBMS_HM.GET_RUN_REPORT('HM_RUN_1061') FROM DUAL;

DBMS_HM.GET_RUN_REPORT('HM_RUN_1061')
---------------------------------------------------------------------
```

```
 Run Name                      : HM_RUN_1061
 Run Id                        : 1061
 Check Name                    : Data Block Integrity Check
 Mode                          : REACTIVE
 Status                        : COMPLETED
 Start Time                    : 2007-05-12 22:11:02.032292 -07:00
 End Time                      : 2007-05-12 22:11:20.835135 -07:00
 Error Encountered             : 0
 Source Incident Id            : 7418
 Number of Incidents Created   : 0

Input Paramters for the Run
 BLC_DF_NUM=1
 BLC_BL_NUM=64349

Run Findings And Recommendations
 Finding
 Finding Name  : Media Block Corruption
 Finding ID    : 1065
 Type          : FAILURE
 Status        : OPEN
 Priority      : HIGH
 Message       : Block 64349 in datafile 1:
               '/u01/app/oracle/dbs/t_db1.f' is media corrupt
 Message       : Object BMRTEST1 owned by SYS might be unavailable
 Finding
 Finding Name  : Media Block Corruption
 Finding ID    : 1071
 Type          : FAILURE
 Status        : OPEN
 Priority      : HIGH
 Message       : Block 64351 in datafile 1:
               '/u01/app/oracle/dbs/t_db1.f' is media corrupt
 Message       : Object BMRTEST2 owned by SYS might be unavailable
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_HM` package.

Viewing Reports Using the ADRCI Utility

You can create and view Health Monitor checker reports using the ADRCI utility.

To create and view a checker report using ADRCI:

1. Ensure that operating system environment variables (such as `ORACLE_HOME`) are set properly, and then enter the following command at the operating system command prompt:

   ```
   ADRCI
   ```

   The utility starts and displays the following prompt:

   ```
   adrci>>
   ```

   Optionally, you can change the current ADR home. Use the `SHOW HOMES` command to list all ADR homes, and the `SET HOMEPATH` command to change the current ADR home. See *Oracle Database Utilities* for more information.

2. Enter the following command:

   ```
   show hm_run
   ```

This command lists all the checker runs (stored in V$HM_RUN) registered in the ADR repository.

3. Locate the checker run for which you want to create a report and note the checker run name. The REPORT_FILE field contains a filename if a report already exists for this checker run. Otherwise, generate the report with the following command:

   ```
   create report hm_run run_name
   ```

4. To view the report, enter the following command:

   ```
   show report hm_run run_name
   ```

See Also: "Automatic Diagnostic Repository (ADR)" on page 9-5

Health Monitor Views

Instead of requesting a checker report, you can view the results of a specific checker run by directly querying the ADR data from which reports are created. This data is available through the views VHM_RUN, VHM_FINDING, and V$HM_RECOMMENDATION.

The following example queries the V$HM_RUN view to determine a history of checker runs:

```
SELECT run_id, name, check_name, run_mode, src_incident FROM v$hm_run;

    RUN_ID NAME         CHECK_NAME                        RUN_MODE SRC_INCIDENT
---------- ------------ --------------------------------- -------- ------------
         1 HM_RUN_1     DB Structure Integrity Check      REACTIVE            0
       101 HM_RUN_101   Transaction Integrity Check       REACTIVE         6073
       121 TXNCHK       Transaction Integrity Check       MANUAL              0
       181 HMR_tab$     Dictionary Integrity Check        MANUAL              0
          .
          .
          .
       981 Proct_ts$    Dictionary Integrity Check        MANUAL              0
      1041 HM_RUN_1041  DB Structure Integrity Check      REACTIVE            0
      1061 HM_RUN_1061  Data Block Integrity Check        REACTIVE         7418
```

The next example queries the V$HM_FINDING view to obtain finding details for the reactive data block check with RUN_ID 1061:

```
SELECT type, description FROM v$hm_finding WHERE run_id = 1061;

TYPE          DESCRIPTION
------------- ----------------------------------------
FAILURE       Block 64349 in datafile 1: '/u01/app/orac
              le/dbs/t_db1.f' is media corrupt

FAILURE       Block 64351 in datafile 1: '/u01/app/orac
              le/dbs/t_db1.f' is media corrupt
```

See Also:

- "Types of Health Checks" on page 9-22
- *Oracle Database Reference* for more information on the V$HM_* views

Health Check Parameters Reference

The following tables describe the parameters for those health checks that require them. Parameters with a default value of `(none)` are mandatory.

Table 9–6 Parameters for Data Block Integrity Check

Parameter Name	Type	Default Value	Description
`BLC_DF_NUM`	Number	(none)	Block datafile number
`BLC_BL_NUM`	Number	(none)	Data block number

Table 9–7 Parameters for Redo Integrity Check

Parameter Name	Type	Default Value	Description
`SCN_TEXT`	Text	0	SCN of the latest good redo (if known)

Table 9–8 Parameters for Undo Segment Integrity Check

Parameter Name	Type	Default Value	Description
`USN_NUMBER`	Text	(none)	Undo segment number

Table 9–9 Parameters for Transaction Integrity Check

Parameter Name	Type	Default Value	Description
`TXN_ID`	Text	(none)	Transaction ID

Table 9–10 Parameters for Dictionary Integrity Check

Parameter Name	Type	Default Value	Description
`CHECK_MASK`	Text	`ALL`	Possible values are: ■ `COLUMN_CHECKS`—Run column checks only. Verify column-level constraints in the core tables. ■ `ROW_CHECKS`—Run row checks only. Verify row-level constraints in the core tables. ■ `REFERENTIAL_CHECKS`—Run referential checks only. Verify referential constraints in the core tables. ■ `ALL`—Run all checks.
`TABLE_NAME`	Text	`ALL_CORE_TABLES`	Name of a single core table to check. If omitted, all core tables are checked.

Repairing SQL Failures with the SQL Repair Advisor

In the rare case that a SQL statement fails with a critical error, you can run the SQL Repair Advisor to try to repair the failed statement.

This section covers the following topics:

- About the SQL Repair Advisor
- Running the SQL Repair Advisor

- Viewing, Disabling, or Removing a SQL Patch

About the SQL Repair Advisor

You run the SQL Repair Advisor after a SQL statement fails with a critical error. The advisor analyzes the statement and in many cases recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternate execution plan for future executions.

Running the SQL Repair Advisor

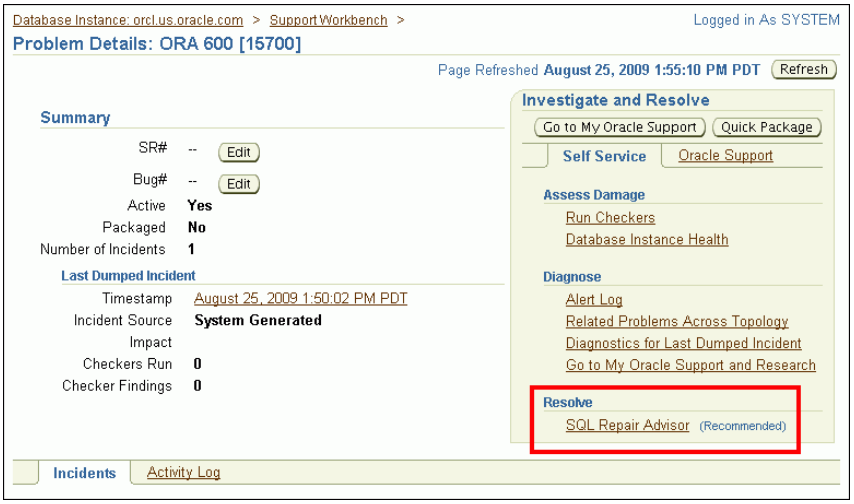
You run the SQL Repair Advisor from the Problem Details page of the Support Workbench. The instructions in this section assume that you were already notified of a critical error caused by your SQL statement and that you followed the workflow described in "Investigating, Reporting, and Resolving a Problem" on page 9-11.

To run the SQL Repair Advisor:

1. Access the Problem Details page for the problem that pertains to the failed SQL statement.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. In the Investigate and Resolve section, under the Self Service tab, under the Resolve heading, click **SQL Repair Advisor**.

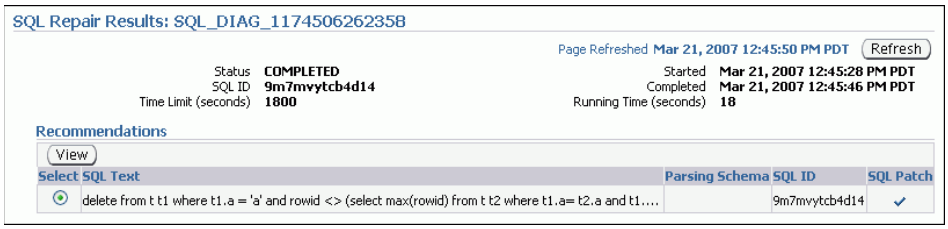


3. On the SQL Repair Advisor page, complete these steps:

   a. Modify the preset task name if desired, optionally enter a task description, modify or clear the optional time limit for the advisor task, and adjust settings to schedule the advisor to run either immediately or at a future date and time.

   b. Click **Submit**.

   A "Processing" page appears. After a short delay, the SQL Repair Results page appears.



A check mark in the SQL Patch column indicates that a recommendation is present. The absence of a check mark in this column means that the SQL Repair Advisor was unable to devise a patch for the SQL statement.

Note: If the SQL Repair Results page fails to appear, then complete these steps to display it:

1. Go to the Database Home page.
2. Under Related Links, click **Advisor Central**.
3. On the Advisor Central page, in the Results list, locate the most recent entry for the SQL Repair Advisor.
4. Select the entry and click **View Result**.

---

4. If a recommendation is present (there is a check mark in the SQL Patch column), click **View** to view the recommendation.

   The Repair Recommendations page appears, showing the recommended patch for the statement.

5. Click **Implement**.

   The SQL Repair Results page returns, showing a confirmation message.

6. (Optional) Click **Verify using SQL Worksheet** to run the statement in the SQL worksheet and verify that the patch successfully repaired the statement.

Viewing, Disabling, or Removing a SQL Patch

After you apply a SQL patch with the SQL Repair Advisor, you may want to view it to confirm its presence, disable it, or remove it. One reason to remove a patch is if you install a later release of Oracle Database that fixes the bug that caused the failure in the patched SQL statement.

To view, disable, or remove a SQL patch:

1. Access the Database Home page in Enterprise Manager.

   For Oracle Enterprise Manager Database Control, see *Oracle Database 2 Day DBA* for instructions. For Oracle Enterprise Manager Grid Control, go to the desired database target.

2. At the top of the page, click **Server** to display the Server page.
3. In the Query Optimizer section, click **SQL Plan Control**.

   The SQL Plan Control page appears. See the online help for information about this page.

4. At the top of the page, click **SQL Patch** to display the SQL Patch subpage.

   The SQL Patch subpage displays all SQL patches in the database.

5. Locate the desired patch by examining the associated SQL text.

   Click the SQL text to view the complete text of the statement.

6. To disable the patch, select it, and then click **Disable**.

   A confirmation message appears, and the patch status changes to `DISABLED`. You can later reenable the patch by selecting it and clicking **Enable**.

7. To remove the patch, select it, and then click **Drop**.

   A confirmation message appears.

See Also: "About the SQL Repair Advisor" on page 9-29

Repairing Data Corruptions with the Data Recovery Advisor

You use the Data Recovery Advisor to repair data block corruptions, undo corruptions, data dictionary corruptions, and more. The Data Recovery Advisor integrates with the Enterprise Manager Support Workbench (Support Workbench), with the Health Monitor, and with the RMAN utility to display data corruption problems, assess the extent of each problem (critical, high priority, low priority), describe the impact of a problem, recommend repair options, conduct a feasibility check of the customer-chosen option, and automate the repair process.

Oracle Database 2 Day DBA provides details on how to use the Data Recovery Advisor. This section describes the various ways to access the advisor from the Support Workbench.

The Data Recovery Advisor is automatically recommended by and accessible from the Support Workbench when you are viewing:

- Problem details for a problem that is related to a data corruption or other data failure.
- Health checker findings that are related to a data corruption or other data failure.

The Data Recovery Advisor is also available from the Advisor Central page. A link to this page can be found in the Related Links section of the Database Home page and of the Performance page.

Note: The Data Recovery Advisor is available only when you are connected as `SYSDBA`.

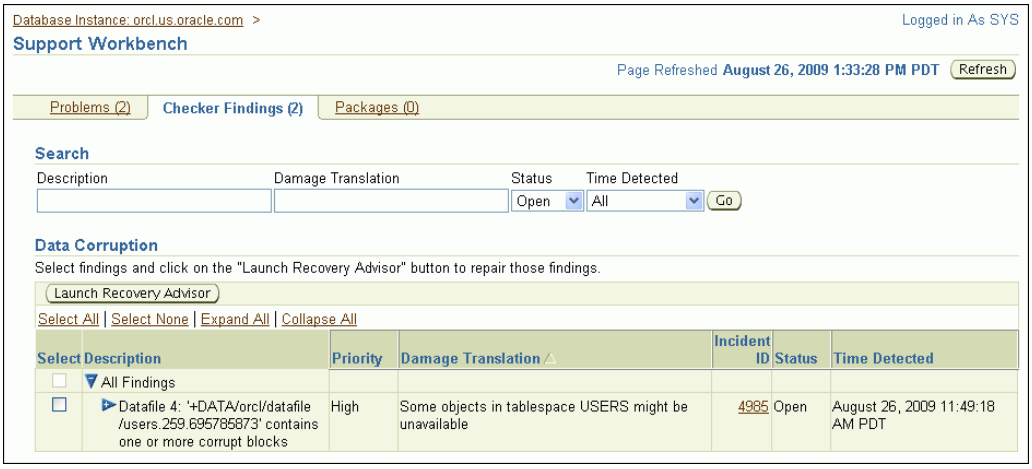
You access the Data Recovery Advisor from the Support Workbench in the following ways:

- From the Problem Details page

Click the **Data Recovery Advisor** link in the Investigate and Resolve section.

See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions on how to access this page.

- From the Checker Findings subpage of the Support Workbench home page



Select one or more data corruption findings and then click **Launch Recovery Advisor**.

See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions on how to access the Support Workbench home page.

> **See Also:** *Oracle Database 2 Day DBA* for instructions for running the Data Recovery Advisor

Creating, Editing, and Uploading Custom Incident Packages

Using the Enterprise Manager Support Workbench (Support Workbench), you can create, edit, and upload custom incident packages. With custom incident packages, you have fine control over the diagnostic data that you send to Oracle Support.

In this section:

- About Incident Packages

- Packaging and Uploading Problems with Custom Packaging
- Viewing and Modifying Incident Packages
- Creating, Editing, and Uploading Correlated Packages
- Deleting Correlated Packages
- Setting Incident Packaging Preferences

See Also: "About the Oracle Database Fault Diagnosability Infrastructure" on page 9-1

About Incident Packages

For the customized approach to uploading diagnostic data to Oracle Support, you first collect the data into an intermediate logical structure called an incident package (package). A **package** is a collection of metadata that is stored in the Automatic Diagnostic Repository (ADR) and that points to diagnostic datafiles and other files both in and out of the ADR. When you create a package, you select one or more problems to add to the package. The Support Workbench then automatically adds to the package the problem information, incident information, and diagnostic data (such as trace files and dumps) associated with the selected problems. Because a problem can have many incidents (many occurrences of the same problem), by default only the first three and last three incidents for each problem are added to the package, excluding any incidents that are over 90 days old. You can change these default numbers on the Incident Packaging Configuration page of the Support Workbench.

After the package is created, you can add any type of external file to the package, remove selected files from the package, or edit selected files in the package to remove sensitive data. As you add and remove package contents, only the package metadata is modified.

When you are ready to upload the diagnostic data to Oracle Support, you first create a zip file that contains all the files referenced by the package metadata. You then upload the zip file through Oracle Configuration Manager.

Note: If you do not have Oracle Configuration Manager installed and properly configured, you must upload the zip file manually through My Oracle Support.

For more information about Oracle Configuration Manager, see *Oracle Configuration Manager Installation and Administration Guide*.

More information about packages is presented in the following sections:

- About Correlated Diagnostic Data in Incident Packages
- About Quick Packaging and Custom Packaging
- About Correlated Packages

See Also:

- "Packaging and Uploading Problems with Custom Packaging" on page 9-35
- "Viewing and Modifying Incident Packages" on page 9-39

About Correlated Diagnostic Data in Incident Packages

To diagnose problem, it is sometimes necessary to examine not only diagnostic data that is directly related to the problem, but also diagnostic data that is *correlated* with the directly related data. Diagnostic data can be correlated by time, by process ID, or by other criteria. For example, when examining an incident, it may be helpful to also examine an incident that occurred five minutes after the original incident. Similarly, while it is clear that the diagnostic data for an incident should include the trace file for the Oracle Database process that was running when the incident occurred, it might be helpful to also include trace files for other processes that are related to the original process.

Thus, when problems and their associated incidents are added to a package, any correlated incidents are added at the same time, with their associated trace files.

During the process of creating the physical file for a package, the Support Workbench calls upon the Incident Packaging Service to finalize the package. **Finalizing** means adding to the package any additional trace files that are correlated by time to incidents in the package, and adding other diagnostic information such as the alert log, health checker reports, SQL test cases, configuration information, and so on. Therefore, the number of files in the zip file may be greater than the number of files that the Support Workbench had previously displayed as the package contents.

The Incident Packaging Service follows a set of rules to determine the trace files in the ADR that are correlated to existing package data. You can modify some of those rules in the Incident Packaging Configuration page in Enterprise Manager.

Because both initial package data and added correlated data may contain sensitive information, it is important to have an opportunity to remove or edit files that contain this information before uploading to Oracle Support. For this reason, the Support Workbench enables you to run a command that finalizes the package as a separate operation. After manually finalizing a package, you can examine the package contents, remove or edit files, and then generate and upload a zip file.

Note: Finalizing a package does not mean closing it to further modifications. You can continue to add diagnostic data to a finalized package. You can also finalize the same package multiple times. Each time that you finalize, any new correlated data is added.

See Also: "Setting Incident Packaging Preferences" on page 9-46

About Quick Packaging and Custom Packaging

The Enterprise Manager Support Workbench provides two methods for creating and uploading an incident package: the quick packaging method and the custom packaging method.

Quick Packaging—This is the more automated method with a minimum of steps, organized in a guided workflow (a wizard). You select a single problem, provide a package name and description, and then schedule upload of the package contents, either immediately or at a specified date and time. The Support Workbench automatically places diagnostic data related to the problem into the package, finalizes the package, creates the zip file, and then uploads the file. With this method, you do not have the opportunity to add, edit, or remove package files or add other diagnostic data such as SQL test cases. However, it is the simplest and quickest way to get first-failure diagnostic data to Oracle Support. Quick packaging is the method used in the workflow described in "Investigating, Reporting, and Resolving a Problem" on page 9-11.

Note that when quick packaging is complete, the package that was created by the wizard remains. You can then modify the package with custom packaging operations at a later time and manually reupload.

Custom Packaging—This is the more manual method, with more steps. It is intended for expert Support Workbench users who want more control over the packaging process. With custom packaging, you can create a new package with one or more problems, or you can add one or more problems to an existing package. You can then perform a variety of operations on the new or updated package, including:

- Adding or removing problems or incidents
- Adding, editing, or removing trace files in the package
- Adding or removing external files of any type
- Adding other diagnostic data such as SQL test cases
- Manually finalizing the package and then viewing package contents to determine if you must edit or remove sensitive data or remove files to reduce package size.

You might conduct these operations over several days, before deciding that you have enough diagnostic information to send to Oracle Support.

With custom packaging, you create the zip file and request upload to Oracle Support as two separate steps. Each of these steps can be performed immediately or scheduled for a future date and time.

> **See Also:** "Task 5 – Package and Upload Diagnostic Data to Oracle Support" on page 9-15 for instructions for the Quick Packaging method

About Correlated Packages

Correlated packages provide a means of packaging and uploading diagnostic data for related problems. A database instance problem can have related problems in other database instances or in Oracle Automatic Storage Management instances, as described in "Related Problems Across the Topology" on page 9-4. After you create and upload a package for one or more database instance problems (the "main package"), you can create and upload one or more correlated packages, each with one or more related problems. You can accomplish this only with the custom packaging workflow in Enterprise Manager Support Workbench.

> **See Also:** "Creating, Editing, and Uploading Correlated Packages" on page 9-45

Packaging and Uploading Problems with Custom Packaging

You use Enterprise Manager Support Workbench (Support Workbench) to create and upload custom incident packages (packages). Before uploading, you can manually add, edit, and remove diagnostic datafiles from the package.

To package and upload problems with custom packaging:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. (Optional) For each problem that you want to include in the package, indicate the service request number (SR#) associated with the problem, if any. To do so, complete the following steps for each problem:

a. In the Problems subpage at the bottom of the Support Workbench home page, select the problem, and then click **View**.

Note: If you do not see the desired problem in the list of problems, or if there are too many problems to scroll through, select a time period from the View list and click **Go**. You can then select the desired problem and click **View**.

The Problem Details page appears.

b. Next to the SR# label, click **Edit**, enter a service request number, and then click **OK**.

The service request number is displayed on the Problem Details page.

c. Return to the Support Workbench home page by clicking **Support Workbench** in the locator links at the top of the page.

3. On the Support Workbench home page, select the problems that you want to package, and then click **Package**.

The Select Packaging Mode page appears.

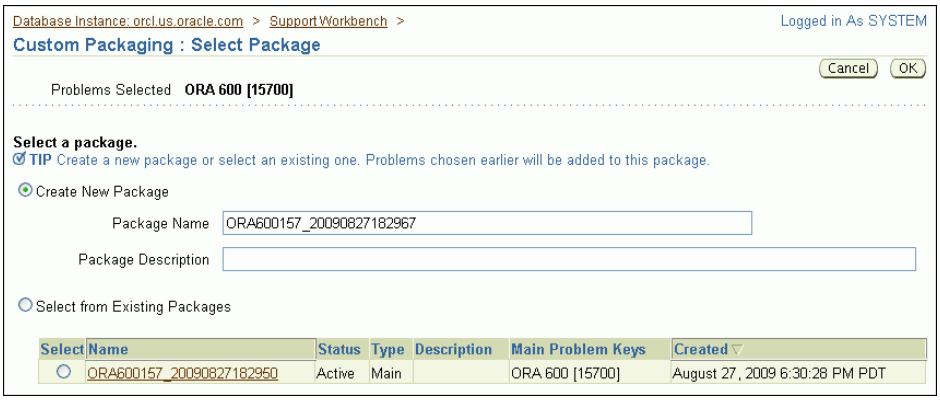
Note: The packaging process may automatically select additional correlated problems to add to the package. An example of a correlated problem is one that occurs within a few minutes of the selected problem. See "About Correlated Diagnostic Data in Incident Packages" on page 9-34 for more information.

---

4. Select the **Custom packaging** option, and then click **Continue**.

The Select Package page appears.

Figure 9–6 Select Package Page



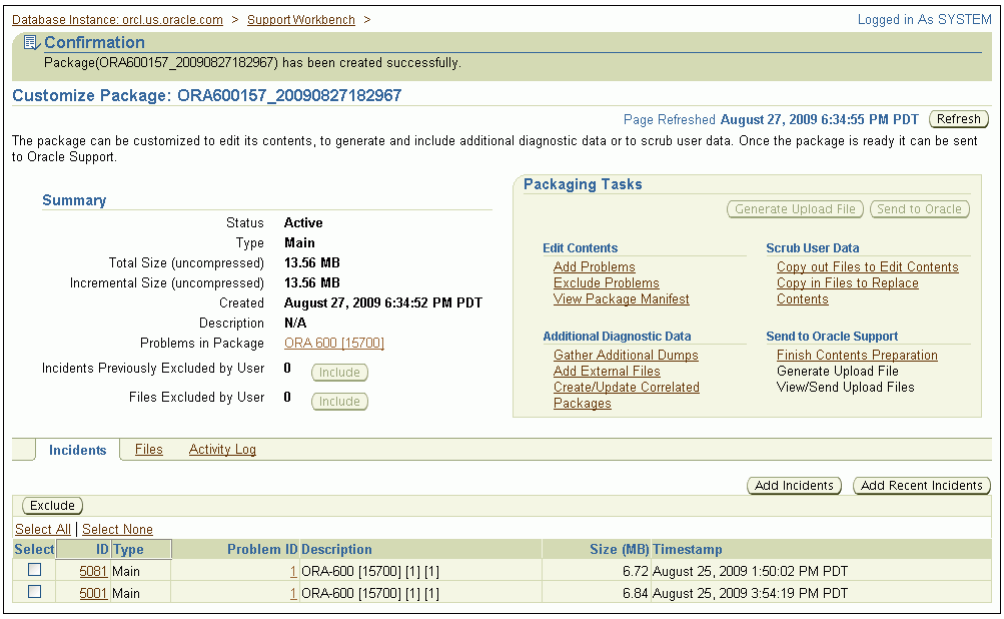
5. Do one of the following:

- To create a new package, select the **Create new package** option, enter a package name and description, and then click **OK**.

- To add the selected problems to an existing package, select the **Select from existing packages** option, select the package to update, and then click **OK**.

The Customize Package page appears. It displays the problems and incidents that are contained in the package, plus a selection of packaging tasks to choose from. You run these tasks against the new package or the updated existing package.

Figure 9–7 Customize Package Page



6. (Optional) In the Packaging Tasks section, click links to perform one or more packaging tasks. Or, use other controls on the Customize Package page and its subpages to manipulate the package. Return to the Customize Package page when you are finished.

   See "Viewing and Modifying Incident Packages" on page 9-39 for instructions for some of the most common packaging tasks.

7. In the Packaging Tasks section of the Customize Package page, under the heading Send to Oracle Support, click **Finish Contents Preparation** to finalize the package.

   A list (or partial list) of files included in the package is displayed. (This may take a while.) The list includes files that were determined to contain correlated diagnostic information and added by the finalization process.

   See "About Correlated Diagnostic Data in Incident Packages" on page 9-34 for a definition of package finalization.

8. Click the **Files** link to view all the files in the package. Examine the list to see if there are any files that might contain sensitive data that you do not want to expose. If you find such files, exclude (remove) or edit them.

   See "Editing Incident Package Files (Copying Out and In)" on page 9-40 and "Removing Incident Package Files" on page 9-44 for instructions for editing and removing files.

   To view the contents of a file, click the eyeglasses icon in the rightmost column in the table of files. Enter host credentials, if prompted.

> **Note:** Trace files are generally for Oracle internal use only.

9. Click **Generate Upload File**.

   The Generate Upload File page appears.

10. Select the **Full** or **Incremental** option to generate a full package zip file or an incremental package zip file.

    For a full package zip file, all the contents of the package (original contents and all correlated data) are always added to the zip file.

    For an incremental package zip file, only the diagnostic information that is new or modified since the last time that you created a zip file for the same package is added to the zip file. For example, if trace information was appended to a trace file since that file was last included in the generated physical file for a package, the trace file is added to the incremental package zip file. Conversely, if no changes were made to a trace file since it was last uploaded for a package, that trace file is not included in the incremental package zip file.

    > **Note:** The Incremental option is dimmed (unavailable) if an upload file was never created for the package.

11. Schedule file creation either immediately or at a future date and time (select **Immediately** or **Later**), and then click **Submit**.

    File creation can use significant system resources, so it may be advisable to schedule it for a period of low system usage.

    A Processing page appears, and creation of the zip file proceeds. A confirmation page appears when processing is complete.

    > **Note:** The package is automatically finalized when the zip file is created.

12. Click **OK**.

    The Customize Package page returns.

13. Click **Send to Oracle**.

    The View/Send Upload Files page appears.

14. (Optional) Click the **Send Correlated Packages** link to create correlated packages and send them to Oracle.

    See "Creating, Editing, and Uploading Correlated Packages" on page 9-45. When you are finished working with correlated packages, return to the View/Send Upload Files page by clicking the **Package Details** link at the top of the page, clicking **Customize Package**, and then clicking **Send to Oracle** again.

15. Select the zip files to upload, and then click **Send to Oracle**.

    The Send to Oracle page appears. The selected zip files are listed in a table.

16. Fill in the requested My Oracle Support information. Next to Create new Service Request (SR), select **Yes** or **No**. If you select Yes, a draft service request is created

for you. You must later log in to My Oracle Support and fill in the service request details. If you select No, enter an existing service request number.

17. Schedule the upload to take place immediately or at a future date and time, and then click **Submit**.

    A Processing page appears. If the upload is completed successfully, a confirmation page appears. If the upload could not complete, an error page appears. The error page may include a message that requests that you upload the zip file to Oracle manually. If so, contact your Oracle Support representative for instructions.

18. Click **OK**.

    The View/Send Upload Files page returns. Under the Time Sent column, check the status of the files that you attempted to upload.

    Note: The Support Workbench uses Oracle Configuration Manager to upload the physical files. If Oracle Configuration Manager is not installed or properly configured, the upload may fail. In this case, a message is displayed with a path to the package zip file and a request that you upload the file to Oracle Support manually. You can upload manually with My Oracle Support.

    For more information about Oracle Configuration Manager, see *Oracle Configuration Manager Installation and Administration Guide*.

19. (Optional) Create and upload correlated packages.

    See "Creating, Editing, and Uploading Correlated Packages" on page 9-45 for instructions.

    See Also:

    - "About Incidents and Problems" on page 9-3
    - "About Incident Packages" on page 9-33
    - "About Quick Packaging and Custom Packaging" on page 9-34

Viewing and Modifying Incident Packages

After creating an incident package with the custom packaging method, you can view or modify the contents of the package before uploading the package to Oracle Support. In addition, after using the quick packaging method to package and upload diagnostic data, you can view or modify the contents of the package that the Support Workbench created, and then reupload the package. To modify a package, you choose from among a selection of *packaging tasks*, most of which are available from the Customize Package page. (See Figure 9–7 on page 9-37.)

This section provides instructions for some of the most common packaging tasks. It includes the following topics:

- Editing Incident Package Files (Copying Out and In)
- Adding an External File to an Incident Package
- Removing Incident Package Files
- Viewing and Updating the Incident Package Activity Log

Also included are the following topics, which explains how to view package details and how to access the Customize Package page for a particular package:

- Viewing Package Details
- Accessing the Customize Package Page

See Also:

- "About Incident Packages" on page 9-33
- "Packaging and Uploading Problems with Custom Packaging" on page 9-35

Viewing Package Details

The Package Details page contains information about the incidents, trace files, and other files in a package, and enables you to view and add to the package activity log.

To view package details:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. Click the **Packages** link to view the Packages subpage.

   A list of packages that are currently in the Automatic Diagnostic Repository (ADR) is displayed.

3. (Optional) To reduce the number of packages displayed, enter text into the **Search** field above the list, and then click **Go**.

   All packages that contain the search text anywhere in the package name are displayed. To view the full list of packages, click the **Packages** link again.

4. Under the Package Name column, click the link for the desired package.

   The Package Details page appears.

Accessing the Customize Package Page

The Customize Package page is used to perform various packaging tasks, such as adding and removing problems; adding, removing, and scrubbing (editing) package files; and generating and uploading the package zip file.

To access the Customize Package page:

1. Access the Package Details page for the desired package, as described in "Viewing Package Details" on page 9-40.

2. Click **Customize Package**.

   The Customize Package page appears. See Figure 9–7 on page 9-37.

Editing Incident Package Files (Copying Out and In)

The Support Workbench enables you to edit one or more files in an incident package. You may want to do this to delete or overwrite sensitive data in the files. To edit package files, you must first copy the files out of the package into a designated directory, edit the files with a text editor or other utility, and then copy the files back into the package, overwriting the original package files.

The following procedure assumes that the package is already created and contains diagnostic data.

To edit incident package files:

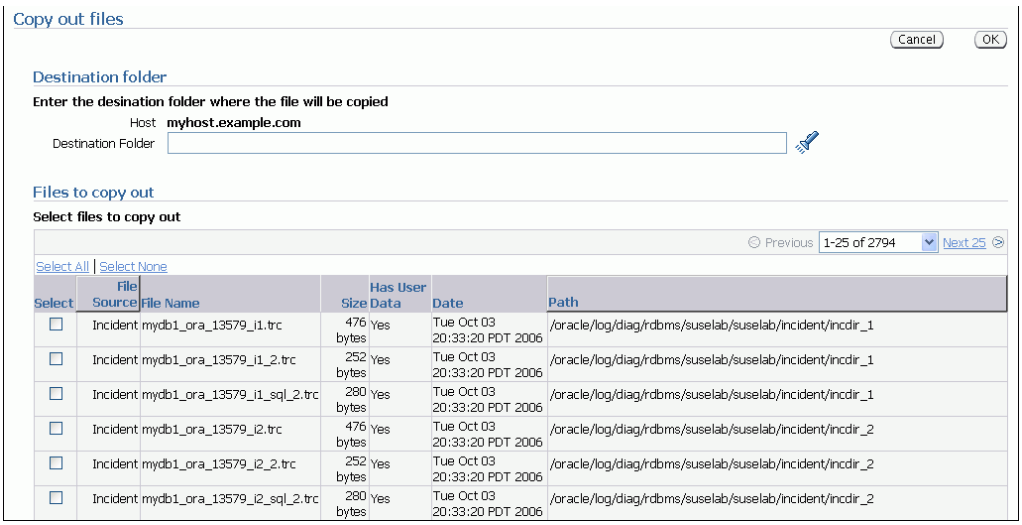
1. Access the Customize Package page for the desired incident package.

   See "Accessing the Customize Package Page" on page 9-40 for instructions.

2. In the Packaging Tasks section, under the Scrub User Data heading, click **Copy out Files to Edit contents**.

   The Copy Out Files page appears. It displays the name of the host to which you can copy files.

Figure 9–8 Copy Out Files Page

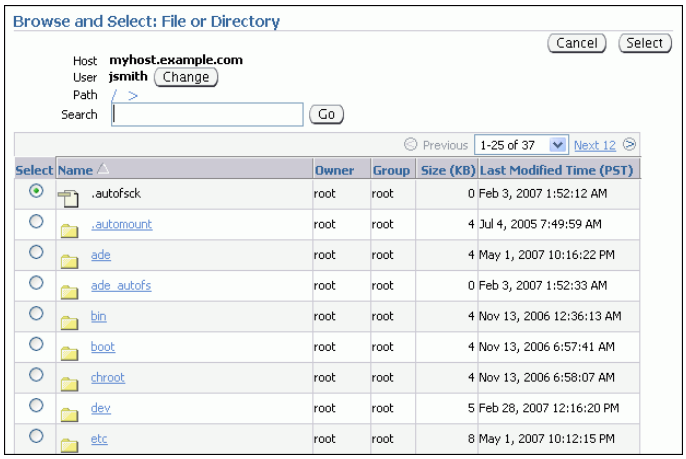


3. Do one of the following to specify a destination directory for the files:

   - Enter a directory path in the **Destination Folder** field.
   - Click the flashlight icon next to the **Destination Folder** field, and then complete the following steps:

     a. If prompted for host credentials, enter credentials for the host to which you want to copy out the files, and then click **OK**. (Select **Save as Preferred Credential** to avoid the prompt for credentials next time.)

     b. In the Browse and Select File or Directory window, click directory links to move down the directory hierarchy, and click directory names next to the **Path** label to move up the directory hierarchy, until you see the desired destination directory.



To reduce the number of directories displayed in the list, enter search text in the **Search** field and click **Go**. All directories that have the search text anywhere in the directory name are displayed.

c. Select the desired destination directory, and then click **Select**.

The Browse and Select File or Directory window closes, and the path to the selected directory appears in the Destination Folder field of the Copy Out Files page.

4. Under Files to Copy Out, select the desired files, and then click **OK**.

Note: If you do not see the desired files, they may be on another page. Click the **Next** link to view the next page. Continue clicking **Next**, or select from the list of file numbers (to the left of the Next link) until you see the desired files. You can then select the files and click **OK**.

The Customize Package page returns, displaying a confirmation message that lists the files that were copied out.

5. Using a text editor or other utility, edit the files.
6. On the Customize Package page, in the Packaging Tasks section, under the Scrub User Data heading, click **Copy in Files to Replace Contents**.

The Copy In Files page appears. It displays the files that you copied out.

7. Select the files to copy in, and then click **OK**.

The files are copied into the package, overwriting the existing files. The Customize Package page returns, displaying a confirmation message that lists the files that were copied in.

Adding an External File to an Incident Package

You can add any type of external file to an incident package.

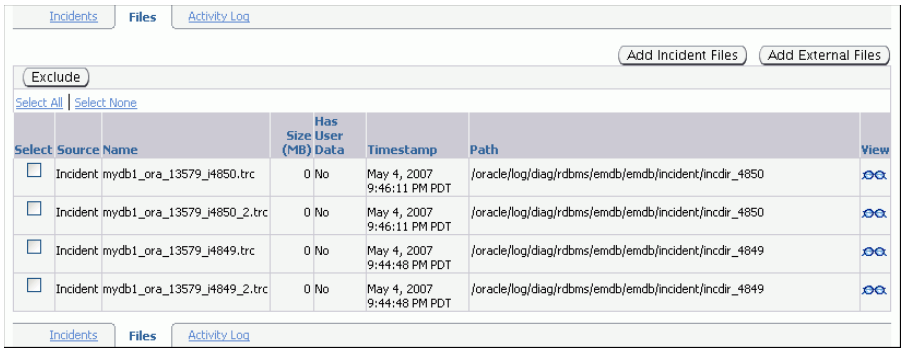
To add an external file to an incident package:

1. Access the Customize Package page for the desired incident package.

See "Accessing the Customize Package Page" on page 9-40 for instructions.

2. Click the **Files** link to view the Files subpage.

Figure 9–9 Files Subpage of Customize Package Page



From this page, you can add and remove files to and from the package.

3. Click **Add external files**.

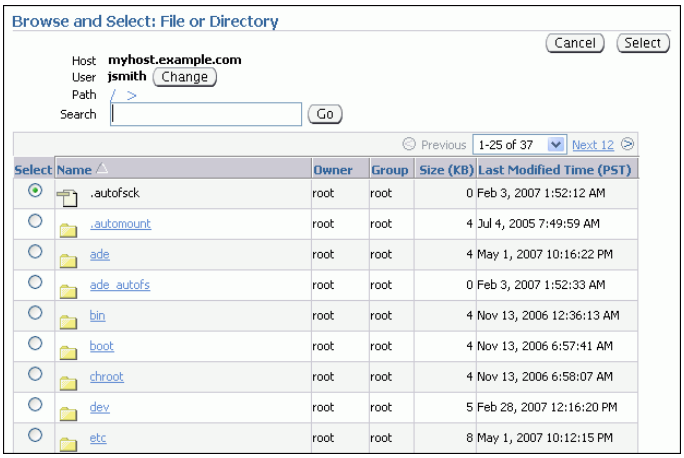
   The Add External File page appears. It displays the host name from which you may select a file.

4. Do one of the following to specify a file to add:

   - Enter the full path to the file in the **File Name** field.
   - Click the flashlight icon next to the **File Name** field, and then complete the following steps:

     a. If prompted for host credentials, enter credentials for the host on which the external file resides, and then click **OK**. (Select **Save as Preferred Credential** to avoid the prompt for credentials next time.)

     b. In the Browse and Select File or Directory window, click directory links to move down the directory hierarchy, and click directory names next to the **Path** label to move up the directory hierarchy, until you see the desired file.



     To reduce the number of files or directories displayed in the list, enter search text in the **Search** field and click **Go**. All files or directories that have the search text anywhere in the file name or directory name are displayed.

     c. In the Select column, click to select the desired file, and then click **Select**.

The Browse and Select window closes, and the path to the selected file appears in the File Name field of the Add External File page.

5. Click **OK**.

   The Customize Package page returns, displaying the Files subpage. The selected file is now shown in the list of files.

Removing Incident Package Files

You can remove one or more files of any type from the incident package.

To remove incident package files:

1. Access the Customize Package page for the desired incident package.

   See "Accessing the Customize Package Page" on page 9-40 for instructions.

2. Click the **Files** link to view the Files subpage.

   A list of files in the package is displayed.

   If you have not yet generated a physical file for this package, all package files are displayed in the list. If you have already generated a physical file, a View list appears above the files list. It enables you to choose between viewing only incremental package contents or the full package contents. The default selection is incremental package contents. This default selection displays only those package files that were created or modified since the last time that a physical file was generated for the package. Select **Full package contents** from the View list to view all package files.

3. Select the files to remove, and then click **Exclude**.

   Note: If you do not see the desired files, they may be on another page. Click the **Next** link to view the next page. Continue clicking **Next**, or select from the list of file numbers (to the left of the Next link) until you see the desired files. You can then select the files and click **Remove**.

Viewing and Updating the Incident Package Activity Log

The Support Workbench maintains an activity log for each incident package. Most activities that you perform on a package, such as adding or removing files or creating a package zip file, are recorded in the log. You can also add your own notes to the log. This is especially useful if multiple database administrators are working with packages.

To view and update the incident package activity log:

1. Access the Package Details page for the desired incident package.

   See "Accessing the Customize Package Page" on page 9-40 for instructions.

2. Click the **Activity Log** link to view the Activity Log subpage.

   The activity log is displayed.

3. To add your own note to the activity log, enter text into the **Note** field, and then click **Add Note**.

   Your note is timestamped and appended to the list.

Creating, Editing, and Uploading Correlated Packages

After you upload a package to Oracle Support, you can create and upload one or more correlated packages. This is recommended if critical alerts appeared in the Related Alerts section of the Database Home page. The correlated packages are associated with the original package, also known as the **main package**. The main package contains problems that occurred in a database instance. Correlated packages contain problems that occurred on other instances (Oracle ASM instances or other database instances) and that are related problems for the problems in the main package. There can be only one correlated package for each related instance.

To create, edit, and upload a correlated package:

1. View the Package Details page for the main package.

   See "Viewing Package Details" on page 9-40 for instructions.

2. On the Package Details page, click **Customize Package**.
3. On the Customize Package page, in the Packaging Tasks section, under Additional Diagnostic Data, click **Create/Update Correlated Packages**.

   See Figure 9–7 on page 9-37.

4. On the Correlated Packages page, under Correlated Packages, select one or more instances that have incidents and click **Create**.

   A confirmation message appears, and the package IDs of the newly created correlated packages appear in the ID column.

5. Select the instance on which you created the correlated package, and click **Finish Contents Preparation**.

   A confirmation message appears.

6. (Optional) View and edit a correlated package by completing these steps:
   a. Click the package ID to view the package.

      If prompted for credentials, enter them and click **Login**.

   b. On the Package Details page, click **Files** to view the files in the package.
   c. Click **Customize Package** and perform any desired customization tasks, as described in "Viewing and Modifying Incident Packages" on page 9-39.
7. For each correlated package to upload, click **Generate Upload File**.
8. For each correlated package to send to Oracle, select the package and click **Send to Oracle**.

   Note: If **Send to Oracle** is unavailable (dimmed), there were no correlated incidents for the instance.

   See Also:

   - "About Correlated Packages" on page 9-35
   - "Related Problems Across the Topology" on page 9-4

Deleting Correlated Packages

You delete a correlated package with the Enterprise Manager Support Workbench for the target for which you created the package. For example, if you created a correlated

package for an Oracle ASM instance target, access the Support Workbench for that Oracle ASM instance.

To delete a correlated package:

1. Access the Support Workbench for the target on which you created the correlated package.

   Tip: See the Related Links section at the bottom of any Support Workbench page. Or, see "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18

2. Click **Packages** to view the Packages subpage.
3. Locate the correlated package in the list. Verify that it is a correlated package by viewing the package description.
4. Select the package and click **Delete**.
5. On the confirmation page, click **Yes**.

   See Also:

   - "About Correlated Packages" on page 9-35
   - "Related Problems Across the Topology" on page 9-4

Setting Incident Packaging Preferences

This section provides instructions for setting incident packaging preferences. Examples of incident packaging preferences include the number of days to retain incident information, and the number of leading and trailing incidents to include in a package for each problem. (By default, if a problem has many incidents, only the first three and last three incidents are packaged.) You can change these and other incident packaging preferences with Enterprise Manager or with the ADRCI utility.

To set incident packaging preferences with Enterprise Manager:

1. Access the Support Workbench home page.

   See "Viewing Problems with the Enterprise Manager Support Workbench" on page 9-18 for instructions.

2. In the Related Links section at the bottom of the page, click **Incident Packaging Configuration**.

   The View Incident Packaging Configuration page appears. Click **Help** to view descriptions of the settings on this page.

3. Click **Edit**.

   The Edit Incident Packaging Configuration page appears.

4. Edit settings, and then click **OK** to apply changes.

See Also:

- "About Incident Packages" on page 9-33
- "About Incidents and Problems" on page 9-3
- "Task 5 – Package and Upload Diagnostic Data to Oracle Support" on page 9-15
- *Oracle Database Utilities* for information on ADRCI

10

Managing Oracle Enterprise Manager Database Control

In this chapter:

- Database Control Overview
- Configuring Database Control During and After Installation
- Administering Database Control
- Managing Database Control in an Oracle RAC Environment
- Deleting and Creating Database Control Objects
- EMCA Troubleshooting Tips

Database Control Overview

This chapter describes the functionality and administration of Oracle Enterprise Manager Database Control (Database Control), the Web-based interface and primary tool for managing an Oracle database. Using Database Control, you can perform administrative tasks such as creating schema objects (tables, views, indexes, and so on), managing user security, managing database memory and storage, backing up and recovering your database, and importing and exporting data. You can also view performance and status information about your database.

After you have installed the Oracle Database software, created or upgraded a database, and configured the network, you can use Database Control to manage your database. Database Control also provides an interface for both performance advisors and Oracle utilities such as SQL*Loader and Recovery Manager (RMAN).

Oracle also offers separately licensed Oracle Enterprise Manager management packs, management plug-ins, and other products that you can purchase to enhance the capabilities of Oracle Enterprise Manager in specific environments.

Understanding the Enterprise Manager Directories

When you install Oracle Database, you also install Database Control by default. As a result, the Oracle Database installation procedure installs a set of Enterprise Manager directories and files into the `$ORACLE_HOME` directory.

In particular, the `emctl` commands required to control Database Control are installed into the `$ORACLE_HOME/bin` directory.

The Management Agent and Management Service support files are installed in two locations in an installation of Oracle Database:

- Files that are common and shared among all instances of the database are stored in the following directory:
  - `$ORACLE_HOME/sysman`

    For example, the administration files, which define the supported target types and the scripts used to perform Management Agent configuration tasks are stored in the `$ORACLE_HOME/sysman/admin` directory
- Files that are unique to each instance of the database are stored in the following directory:
  - `$ORACLE_HOME/`*`hostname_dbuniquename`*`/` (for a single-instance database)
  - `$ORACLE_HOME/`*`nodename_dbuniquename`*`/` (for an Oracle Real Applications Cluster (Oracle RAC) database)

    Note: In Oracle Database 10*g*, files that are unique to each instance of the database are stored in following directory of the Oracle Database 10*g* home:

    - `$ORACLE_HOME/`*`hostname_sid`*`/` (for a single-instance database)
    - `$ORACLE_HOME/`*`nodename_sid`*`/` (for an Oracle RAC database)

Throughout the rest of this chapter, `$ORACLE_HOME/`*`hostname_dbuniquename`*`/` and `$ORACLE_HOME/`*`nodename_dbuniquename`* may be used interchangeably. (For release 10*g* of the database, the directories are `$ORACLE_HOME/`*`hostname_sid`*`/` and `$ORACLE_HOME/`*`nodename_sid`*`/`.) Both paths refer to the same concept: the Enterprise Manager directory for the specific database instance. The difference is that `$ORACLE_HOME/`*`hostname_dbuniquename`*`/` is used for single-instance databases, while `$ORACLE_HOME/`*`nodename_dbuniquename`*`/` is used for cluster (Oracle RAC) databases. In cluster databases, nodename refers to the public name of the node, as specified during Cluster Ready Services (CRS) configuration for cluster environments.

For example, if the database host name is `mgmt1.example.com` and the system identifier for the database name is `db11`, the log files for the Management Agent and Management Service for that instance are installed in the following directory:

```
$ORACLE_HOME/mgmt1.example.com_db11/sysman/log/
```

If a *`hostname_dbuniquename`* directory does not exist in the Oracle home directory, then Database Control was never configured for the database instance. Note that in Oracle Database Release 10*g*, the directory is *`hostname_sid`*.

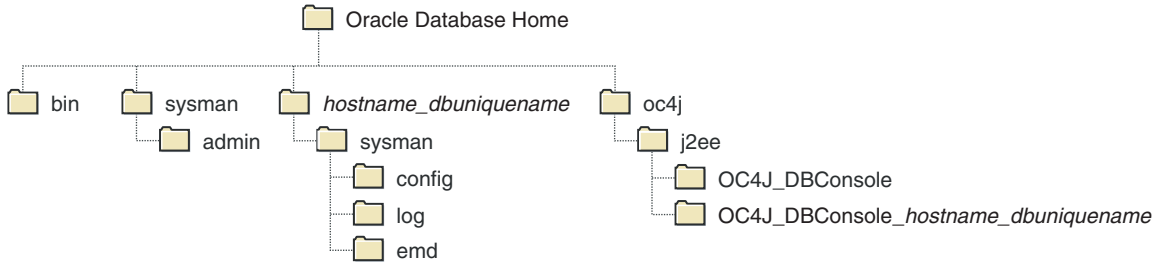
In addition, the files required to deploy the Database Control as a J2EE application are installed into the `$ORACLE_HOME/oc4j/j2ee/` directory structure. Database Control is a J2EE application that is deployed using the standalone version of Oracle Application Server Containers for J2EE (OC4J). The `OC4J_DBConsole` directory contains the template files that are used to create database-specific deployment directories for each Database Control instance deployed in the Oracle home.

The installation and configuration files are stored in the `$ORACLE_HOME` directory in the following sub-directories:

- `cfgtoollogs/dbua`
- `cfgtoollogs/rconfig`
- `cfgtoollogs/dbca`
- `cfgtoollogs/emca`

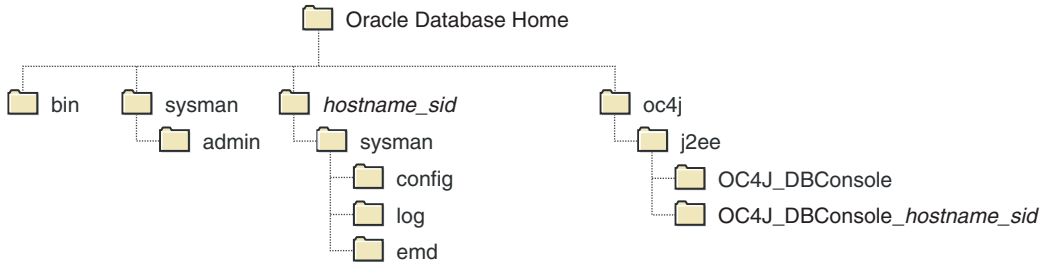
Figure 10–1 summarizes the location of the important Enterprise Manager directories in a typical Oracle home directory for Oracle Database 11*g* Release 2 or later. Note that references to `hostname_dbuniquename` are for single-instance databases; cluster databases have paths of the form `nodename_dbuniquename`.

Figure 10–1 Directory Structure in the Management Service Home for Release 11g or Later



Similarly, Figure 10–2 displays the directory structure of Enterprise Manager directories in the Oracle home directory for Oracle Database Release 10*g*.

Figure 10–2 Directory Structure in the Management Home for Release 10g



Configuring Database Control During and After Installation

The following sections describe how Database Control is configured during installation, and how you can configure Database Control after installation.

Overview of Configuration Options

The following are the products, tools, and utilities you, as a database administrator, can use to configure the Database Control environment:

- Oracle Universal Installer

  Oracle Universal Installer (OUI) is a utility that installs your Oracle software and options. It can automatically start Oracle Database Configuration Assistant to install a database.

- DBCA

  Oracle Database Configuration Assistant (DBCA) is a utility that creates a database from templates that are supplied by Oracle. It also enables you to create your own database by copying a preconfigured seed database, thus saving the time and effort needed to generate and customize a new database.

- Database Upgrade Assistant

  The Database Upgrade Assistant is a tool that guides you through the upgrade of your existing database to a new Oracle Database release.

- Net Configuration Assistant

  Net Configuration Assistant (NETCA) is a utility that enables you to configure listeners and naming methods, which are critical components of the Oracle Database network.

Configuring Database Control During Installation

If you create a database while installing Oracle Database, you have the option of configuring your database so it can be managed by Oracle Enterprise Manager Grid Control Console or by Database Control Console.

To select Grid Control Console as your management option, the Oracle Management Service must be installed on a network host. In addition, the Oracle Management Agent must be installed on the host where you are installing the database. Otherwise, the Grid Control Console option is unavailable and you must instead choose to manage your database with Database Control.

For most of the Oracle installation types, you must choose either Database Control or Grid Control as your management option when you create a database during the installation.

However, if you create a database using one of the following methods, you can choose not to configure Database Control:

- Choosing the Advanced database configuration option during an Enterprise or Standard Edition installation
- Running DBCA after the installation

Note that if you do not configure Database Control during installation of an Oracle Database 11*g* instance, a `hostname_dbuniquename` directory will not be created in the resulting Oracle home directory. When installing a release 10*g* database, a hostname_sid directory will not be created in the resulting Oracle home directory.

Configuring Database Control with DBCA

The primary method for configuring an existing Oracle database so it can be managed with Database Control is to use DBCA. You can use DBCA to create a new database or to reconfigure an existing database.

> **See Also:** "Installing Oracle Software and Building the Database" in the *Oracle Database 2 Day DBA* book for more information about using DBCA to create a new database instance, and *Oracle Database 2 Day + Real Application Clusters Guide* for Oracle RAC examples using DBCA.

To use DBCA to reconfigure your database so it can be managed with Database Control:

1. Log in to the database host as a member of the administrative group that is authorized to install Oracle software and create and run the database.
2. Start DBCA, change directory to the `$ORACLE_HOME/bin` directory, then enter the following command:

   ```
   $ ./dbca
   ```

   The DBCA Welcome page appears.
3. Advance to the Operations page and select **Configure Database Options**.

4. Advance to the Database page and select the database that you want to configure.
5. Advance to the Management Options page and select the following options:
   - **Configure the Database with Enterprise Manager**
   - **Use Database Control for Database Management**
6. Select any desired options for enabling e-mail notifications and enabling daily backups.

   For more information about Enterprise Manager notifications and daily backups, click **Help** on the Management Options page.
7. Advance until the Finish button is available.
8. Click **Finish** to reconfigure the database to use Database Control.

   After DBCA reconfigures the database, a new subdirectory appears in the Oracle home. In release 11*g* of the database and later, this directory is named using the following format and contains Database Control configuration and state files specific to the database you just configured:

   hostname_dbuniquename

   Note that for cluster databases, the directories are named:

   nodename_dbuniquename.

   In Oracle Database 10*g*, the new subdirectory is named:

   hostname_sid

   Likewise in Oracle Database 10*g*, for cluster databases the directories are named:

   nodename_sid

   For example:

   `mycluster1.example.com_myNewDB`

Configuring Database Control Using EMCA

When you use DBCA to configure Oracle 11g Release 2 (11.2), DBCA provides a graphical user interface to help you select Database Control options and to configure other aspects of your database.

However, to use the operating system command line to configure Database Control, you can use the Enterprise Manager Configuration Assistant (EMCA).

> **Caution:** When you configure Database Control using EMCA, the database may become unavailable and users may not be able to connect to the database or perform operations on the database.

To configure Database Control with EMCA:

1. Set the `ORACLE_HOME` environment variable to specify the Oracle home for the database that you want to manage.
2. Change directory to the `$ORACLE_HOME/bin` directory.
3. Start EMCA by entering the following command with any of the optional command-line arguments shown in Table 10–1, " EMCA Command-Line Operations":

```
$ ./emca
```

Depending on the arguments that you include on the EMCA command line, EMCA prompts you for the information required to configure Database Control.

For example, enter the following command to configure Database Control so that it will perform automatic daily backups of your database:

```
$ ./emca -config dbcontrol db -backup
```

EMCA commands are in the following form:

```
emca [operation] [mode] [flags] [parameters]
```

Note: To configure Database Control for single-instance database using Oracle Automatica Storage Management (ASM), no extra parameters need to be passed along with the EMCA command. Run the following command to configure the Database Control which will automatically detect the Oracle ASM instance:

```
$ emca -config dbcontrol db -repos create
```

Table 10–1 describes the valid execution operations and modes and lists the optional parameters in brackets. Table 10–2 discusses the flags and their behavior and Table 10–3 defines the optional parameters in detail. EMCA parameters are in the form `[-parameterName parameterValue]`. Multiple parameters can be used in combination at the command line.

Table 10–1 EMCA Command-Line Operations

Command	Description
`emca -h | --h | -help | --help`	Use this option to display help messages for the EMCA utility. The options described in Table 10–1, Table 10–2, and Table 10–3, and the valid parameters you may include are listed.
`emca -version`	Prints the version information associated with EMCA.
`emca -config dbcontrol db [-repos (create | recreate)] [-cluster] [-silent] [-backup] [parameters]`	Configures Database Control for a database. Options include creating (or re-creating) the Database Control repository, configuring automatic backups, and performing these operations on a cluster database.
`emca -config centralAgent (db | asm) [-cluster] [-silent] [parameters]`	Configures central agent management for a database or an Oracle ASM instance. Options include performing this operation on a cluster environment. This operation will configure the database so that it can be centrally managed by the Oracle Enterprise Manager Grid Control Console. To use this option, you must have previously installed the Oracle Management Service component of Enterprise Manager on a network host. In addition, the Oracle Management Agent must be installed on the host on which you are running the database.
`emca -config all db [-repos (create | recreate)] [-cluster] [-silent] [-backup] [parameters]`	Configures both Database Control and central agent management for a database. The possible configuration options are similar to those described above.

Table 10–1 (Cont.) EMCA Command-Line Operations

Command	Description
`emca -deconfig dbcontrol db [-repos drop] [-cluster] [-silent] [parameters]`	Deconfigures Database Control for a database. Options include dropping the Database Control repository and performing these operations on a cluster database. For example, you might use this command to remove the Database Control configuration from a database you are planning to delete. In such a scenario, remove the Database Control configuration before physically deleting the database. This operation does not remove the actual database or its datafiles.
`emca -deconfig centralAgent (db | asm) [-cluster] [-silent] [parameters]`	Deconfigures central agent management for a database or an Oracle ASM instance. Options include performing this operation on a cluster environment. For example, you might use this command to remove the central agent management configuration from a database you are planning to delete. In such a scenario, remove the central agent management configuration before physically deleting the database. This operation does not remove the actual database or its datafiles.
`emca -deconfig all db [-repos drop] [-cluster] [-silent] [parameters]`	Deconfigures both Database Control and central agent management for a database. The possible deconfiguration options are similar to those described above.
`emca -addNode (db | asm) [-silent] [parameters]`	Configures Enterprise Manager for a new cluster instance of a database or ASM storage. For more information, see "Using EMCA With Oracle RAC" on page 10-12. This option applies only to Release 11*g*.
`emca -deleteNode (db | asm) [-silent] [parameters]`	Deconfigures Enterprise Manager for a node being deleted. For more information, see "Using EMCA With Oracle RAC" on page 10-12. This option applies only to Oracle Database Release 11*g*.
`emca -addInst (db | asm) [-silent] [parameters]`	Configures Enterprise Manager for a newly added cluster instance of an administrator-managed database or Oracle ASM storage. This parameter takes a SID for the instance being added. For more information, see "Using EMCA With Oracle RAC" on page 10-12.
`emca -deleteInst (db | asm) [-silent] [parameters]`	Deconfigures Enterprise Manager for a newly deleted cluster instance of an administrator-managed database or Oracle ASM storage. This parameter takes a SID for the instance being deleted. This is discussed further below, in "Using EMCA With Oracle RAC" on page 10-12.
`emca -reconfig ports [-cluster] [parameters]`	Explicitly reassigns Database Control ports. Options include performing this operation on a cluster environment. For more information, see "Specifying the Ports Used by Database Control" on page 10-11.
`emca -reconfig dbcontrol -cluster [-silent] [parameters]`	Reconfigures Database Control deployment for a cluster database. Note that this command must be used with the `-cluster` option. For more information, see "Using EMCA With Oracle RAC" on page 10-12.
`emca -displayConfig dbcontrol -cluster [-silent] [parameters]`	Displays information about the current deployment configuration of Database Control in a cluster environment. Note that this command must be used with the `-cluster` option. For more information, see "Using EMCA With Oracle RAC" on page 10-12.

Table 10–1 (Cont.) EMCA Command-Line Operations

Command	Description
`emca -upgrade (db | asm | db_asm) [-cluster] [-silent] [parameters]`	Upgrades the configuration of an earlier release of Enterprise Manager to the current release. This operation can be performed for Oracle Database, Oracle ASM, or Oracle Database and Oracle ASM instances together simultaneously. This does not upgrade the actual database or Oracle ASM instances, nor does it upgrade the Enterprise Manager software. Instead, it upgrades the configuration files for the specified instance so that they are compatible with the current version of the Enterprise Manager software. EMCA will attempt to upgrade all instances of the specified database or Oracle ASM target, or both, on the host, across all Oracle homes (because it is likely that certain target properties, such as the listener port or Oracle home, have changed).
`emca -restore (db | asm | db_asm) [-cluster] [-silent] [parameters]`	Restores the current version of Enterprise Manager configuration to an earlier version. This is the inverse of the `-upgrade` option (and will reverse any changes that result from this operation).
`emca -migrate -from dbcontrol -to centralAgent [-repos drop] [-cluster] [-silent] [parameters]`	Migrates the Enterprise Manager configuration from Database Control to Central Agent.

Table 10–2 EMCA Command-Line Flags

Flag	Description
`db`	Performs the operation for a database (including cluster databases). Use this option for databases that use Oracle ASM to store the datafiles. If a database is using Oracle ASM, all the configuration operations and modes described above (except for `-upgrade` and `-restore`) will detect this automatically and apply the changes to both the database and Oracle ASM instances.
`asm`	Performs the operation for an Oracle ASM-only instance (including cluster Oracle ASM instances).
`db_asm`	This flag can only be used in `-upgrade` and `-restore` mode. Performs the upgrade or restore operation for an Oracle Database and an Oracle ASM instance together. Database and Oracle ASM instances may be upgraded or restored separately (that is, upgrading an Oracle ASM instance does not require upgrading the database instances it services). Therefore, the Enterprise Manager configuration can be upgraded or restored separately for a database and its respective Oracle ASM instance.
`-repos create`	Creates a new Database Control management repository.
`-repos drop`	Drops the current Database Control management repository.
`-repos recreate`	Drops the current Database Control management repository and then re-creates a new one.
`-cluster`	Performs the operation for a cluster database or Oracle ASM instance.

Table 10–2 (Cont.) EMCA Command-Line Flags

Flag	Description
-silent	Performs the operation without prompting for additional information. If this mode is specified, all the required parameters must be entered at the command line or specified in an input file using the -respFile argument. You can view a list of the available parameters by entering emca -help at the command line.
-backup	Configures automatic backup for a database. EMCA will prompt for daily automatic backup options. The default Enterprise Manager settings will be used to backup the database files. **Note:** If you use this option, EMCA will use the value of the db_recovery_file_dest initialization parameter to identify the flashback recovery area for the automated backups. If that parameter is not set, EMCA will generate an error. You can modify these settings later using the Maintenance page in Database Control. For more information, see the Database Control online help.

Table 10–3 EMCA Command-Line Parameters

Parameter	Description
-respFile	Specifies the path of an input file listing parameters for EMCA to use while performing its configuration operation. For more information, see "Using an Input File for EMCA Parameters" on page 10-10.
-SID	Database system identifier.
-PORT	Port number for the listener servicing the database.
-ORACLE_HOME	Database Oracle home, as an absolute path.
-ORACLE_HOSTNAME	Local database hostname.
-LISTENER_OH	Oracle home from which the listener is running. If the listener is running from an Oracle home other than the one on which the database is running, the parameter LISTENER_OH must be specified.
-HOST_USER	Host system user name (for automatic backup).
-HOST_USER_PWD	Host system user password (for automatic backup).
-BACKUP_SCHEDULE	Schedule in the form of "HH:MM" for daily automatic backups.
-EMAIL_ADDRESS	E-mail address for notifications.
-MAIL_SERVER_NAME	Outgoing Mail (SMTP) server for notifications.
-ASM_OH	Oracle ASM Oracle home.
-ASM_SID	System identifier for Oracle ASM instance.
-ASM_PORT	Port number for the listener servicing the Oracle ASM instance.
-ASM_USER_ROLE	User role for connecting to the Oracle ASM instance.
-ASM_USER_NAME	User name for connecting to the Oracle ASM instance.
-ASM_USER_PWD	Password for connecting to the Oracle ASM instance.
-DBSNMP_PWD	Password for the DBSNMP user.
-SYSMAN_PWD	Password for the SYSMAN user.
-SYS_PWD	Password for the SYS user.

Table 10–3 (Cont.) EMCA Command-Line Parameters

Parameter	Description
-SRC_OH	Oracle home of the database with Enterprise Manager configuration to be upgraded or restored.
-DBCONTROL_HTTP_PORT	Use this parameter to specify the port you use to display the Database Control Console in your Web browser. For more information, see "Specifying the Ports Used by Database Control" on page 10-11.
-AGENT_PORT	Use this parameter to specify the Management Agent port for Database Control. For more information, see "Specifying the Ports Used by Database Control" on page 10-11.
-RMI_PORT	Use this parameter to specify the RMI port for Database Control. For more information, see "Specifying the Ports Used by Database Control" on page 10-11.
-JMS_PORT	Use this parameter to specify the JMS port for Database Control. For more information, see "Specifying the Ports Used by Database Control" on page 10-11.
-CLUSTER_NAME	Cluster name (for cluster databases).
-DB_UNIQUE_NAME	Database unique name (for cluster databases).
-SERVICE_NAME	Database service name (for cluster databases).
-EM_NODE	Node from which Database Control console is to be run (for cluster databases). For more information, see "Using EMCA With Oracle RAC" on page 10-12.
-EM_NODE_LIST	Comma-delimited list of NODEs for agent-only configurations, uploading data to -EM_NODE. For more information, see "Using EMCA With Oracle RAC" on page 10-12.
-EM_SWLIB_STAGE_LOC	Software library location.
-PORTS_FILE	Path to a static file specifying the ports to use. The default value is: ${ORACLE_HOME}/install/staticports.ini.

Using an Input File for EMCA Parameters

Instead of answering a series of prompts when you run EMCA, you can use the -respFile argument to specify an input file. The input file you create must be in a format similar to the following example:

```
PORT=1521
SID=DB
DBSNMP_PWD=xpE234D
SYSMAN_PWD=KDOdk432
```

After you create an EMCA input file, you can use it on the command line as follows:

```
$ ./emca -config dbcontrol db -respFile input_file_path
```

For example, to configure the Database Control to perform daily backups and create the Database Control Management Repository, create an input file similar to the one shown in Example 10–1 and enter the following command at the operating system prompt:

```
$ ./emca -config dbcontrol db -repos create -backup -respFile input_file_path
```

Example 10–1 Sample EMCA Input File

```
PORT=1521
```

```
SID=DB
DBSNMP_PWD=dow31224
SYSMAN_PWD=squN3243
HOST_USER=johnson
HOST_USER_PWD=diTf32of
SYS_PWD=qlKj4352
BACKUP_SCHEDULE=06:30
```

Specifying the Ports Used by Database Control

When you initially install Oracle Database or configure Database Control with EMCA, Database Control uses a set of default system ports. For example, by default, you access Database Control using port 1158, as shown in the following example:

```
http://host.domain:1158/em
```

This is the default port assigned to Database Control by the Internet Assigned Numbers Authority (IANA). Likewise, the default Database Control Agent port, as assigned by the IANA, is 3938.

To use ports other than the default ports, use the following EMCA command-line arguments when you initially configure the Database Control with EMCA. Alternatively, you can explicitly assign ports after configuring Database Control using the following command:

```
$ emca -reconfig ports [-cluster]
```

> **Note:** You can also use the following EMCA command-line arguments to configure Database Control after you have installed and configured Oracle Database.

The following list summarizes the EMCA command-line arguments that control the standard Database Control port assignments:

- `-DBCONTROL_HTTP_PORT` *`port_number`*

  This port is used in the Database Control Console URL. For example, if you set this port to 5570, you can then use the following URL to display the Database Control Console:

  ```
  http://host.domain:5570/em
  ```

- `-RMI_PORT` *`port_number`*

  This port is used by the Remote Method Invocation (RMI) system, which is part of the J2EE software required by Database Control. The default port can be changed if you want to configure a specific port for Database Control. When a port other than the default port (1521) is used, use the `-RMI_PORT` or `-JMS_PORT` options along with the `emca reconfig` command.

- `-JMS_PORT` *`port_number`*

  This port is used by the OC4J Java Message Service (JMS), which is part of the J2EE software required by Database Control. The default port can be changed if the user wants to configure a specific port for Database Control. When a port other than the default port (1521) is used, use the `-RMI_PORT` or `-JMS_PORT` options along with the `emca reconfig` command.

- `-AGENT_PORT` *`port_number`*

This port is used by the Database Control Management Agent, which is monitoring and administering the database for the Database Control.

Using EMCA With Oracle RAC

Each EMCA command can be used in Oracle RAC environments, and certain commands are only applicable in cluster setups. To perform cluster commands using EMCA, use the cluster flag that is available in almost every EMCA operational mode.

When you use EMCA to configure Database Control for Oracle RAC, you configure the Database Control for each instance in the cluster. However, by default the Database Control Console will only start on the local node. On every other node of the cluster, only the Enterprise Manager agent will start. This is because the Database Control Console opens several connections to the database. If an instance of the console is running on every host in the cluster, then you may easily exceed the maximum number of permitted open connections on a 32-node or 64-node environment.

To remedy this situation, the Database Control is started only on the local node. On every other node, the commands `emctl start dbconsole` and `emctl stop dbconsole` only start and stop the agent. Each of the remote agents will upload their respective data to the console running on the local node, from which you can monitor and manage all the targets in the cluster. The following subdirectory is created on each instance of the Oracle RAC database where *nodename* is the name of a node in the cluster, and *dbuniquename* is the database instance name of the database for release 11*g* and later:

```
$ORACLE_HOME/nodename_dbuniquename
```

For Oracle Database release 10*g*, *nodename* is the name of the node in the cluster and *sid* is the database instance name:

```
$ORACLE_HOME/nodename_sid
```

However, note that if you upgrade an existing Oracle RAC database configured with Database Control to the current release, then the existing Database Control configuration is retained. The existing Database Control has a Database Control running on each cluster node. The console will still be started on each individual node.

To modify the existing configuration in a release 11*g* and later database, use the following command where *nodename* is the public name of the node and *node_list* is a comma-delimited list of nodes where the database is configured:

```
$ emca -reconfig dbcontrol -cluster -EM_NODE nodename -EM_NODE_LIST node_list
```

To modify the existing configuration in a release 10*g* database, use the following command where `nodename` is the public name of the node and *SID_list* is a comma-delimited list of database system identifiers:

```
$ emca -reconfig dbcontrol -cluster -EM_NODE nodename -EM_SID_LIST SID_list
```

Note: SCAN listener configuration is mandatory when configuring Database Control in an Oracle RAC environment.

This command reconfigures the current Database Control setup and performs the following actions in sequence:

Note: Unless otherwise specified, all command options in the following actions represent release 11*g* and later databases. Release 10*g* database differences are called out in the sequence.

1. Starts a Database Control Console on *nodename*, if one has not been started yet.
2. Redirects the agents monitoring the database instances in *node_list* (*SID_list* in release 10*g* of the database) so that they upload their data to the console running on *nodename*. Also, agents monitoring database instances on *nodename* will also upload their data to the local console. Note that if you do not enter the command options `-EM_NODE` or `-EM_NODE_LIST` (`-EM_SID_LIST` for release 10*g* of the database) at the command line, you will be prompted to enter them.

`-EM_NODE` defaults to the local node if it is not specified when prompted. If not specified, `-EM_NODE_LIST` (`-EM_SID_LIST` for release 10*g* of the database) defaults to all nodes where the database is configured.

You can use this command to start the console on multiple nodes. For instance, on an 8-node cluster with nodes `node1`, `node2`, `node3`, `node4`, `node5`, `node6`, `node7`, and `node8`, you can run the following commands in succession:

```
$ emca -reconfig dbcontrol -cluster -EM_NODE node1 -EM_NODE_LIST node2,node3,node4
$ emca -reconfig dbcontrol -cluster -EM_NODE node5 -EM_NODE_LIST node6,node7,node8
```

For release 10*g* databases, run the following commands in succession:

```
$ emca -reconfig dbcontrol -cluster -EM_NODE node1 -EM_SID_LIST sid2,sid3,sid4
$ emca -reconfig dbcontrol -cluster -EM_NODE node5 -EM_SID_LIST sid6,sid7,sid8
```

In this scenario, there are two Database Control consoles running, one on node1 and the other on node5. From either of these consoles, you can manage and monitor all targets in the cluster.

For information about the current cluster configuration, run the following command:

```
$ emca -displayConfig dbcontrol -cluster
```

The preceding command prompts for the database unique name for the cluster database. This will print the current configuration onto the screen, indicating the nodes that have consoles running on them and the consoles where each agent is uploading.

To configure Enterprise Manager for a newly added node of a cluster, enter the following command:

```
$ emca -addNode db
```

To configure Enterprise Manager for a new cluster instance of an administrator-managed database or Oracle ASM storage, enter the following command:

```
$ emca -addInst db
```

For Oracle Database release 11*g*, on cluster policy managed databases another common operation is the addition and deletion of cluster nodes. After you add a new node, you can run EMCA to configure Database Control or central agent management for that node using the command `emca -addNode db`. Running EMCA only configures Enterprise Manager so that you can manage the instance in a way consistent with the rest of the cluster database instances.

For Oracle Database releases 10*g* and 11, on cluster databases you can create and delete database instances. After you create a new instance, you can run EMCA to configure Database Control or central agent management for that instance using the command `emca -addInst db`. Running EMCA does not create the actual database instance; it only configures Enterprise Manager so that you can manage the instance in a way consistent with the rest of the cluster database instances.

On release 11*g* and later, when configuring Enterprise Manager for a new node, run the EMCA command only after you have added the node. Also, run the command from a node in the cluster that already has Enterprise Manager configured for it, because the Enterprise Manager configuration settings on the existing node are propagated to the new node when you run the command. Because the `-addNode` option can be used only in an Oracle RAC environment, you do not need to use the `-cluster` option on the command line.

After running the command `emca -addNode db`, enter the node name and database unique name for the node and database. For example:

```
Node name: node2
Database Unique Name: EM102
```

Similarly in releases 10*g* and 11*g* of the database, when configuring Enterprise Manager for a new instance, run the EMCA command only after you have created the instance. Also, run the command from a node in the cluster that already has Enterprise Manager configured for its associated database instance, as these configuration settings will be propagated to the new instance. Do not run this command from the node on which the new instance was created. Note that this option can be used only in an Oracle RAC environment so you do not need to use the `-cluster` option on the command line. After running the command `emca -addInst db`, enter the following information for the node and database:

```
Node name: node2
Database Unique Name: EM102
Database SID: EM1022
```

For release 11*g* and later, to deconfigure Enterprise Manager for a specific node you are deleting from the cluster, log in to a different node from the one you plan to delete, then use the command `emca -deleteNode db`. Running this command removes the Enterprise Manager configuration so that you will no longer be able to manage the instance with Enterprise Manager on that node. Ensure that you run the EMCA command before you delete the actual cluster node. Because this option can be used only in an Oracle RAC environment, you do not need to use the `-cluster` option on the command line.

For Oracle Database release 10*g*, to deconfigure Enterprise Manager for a specific database instance (typically before the database instance is deleted), use the inverse command, `emca -deleteInst db`. Running EMCA does not delete the database instance; it only removes the Enterprise Manager configuration so that you will no longer be able to manage the instance with Enterprise Manager.

Ensure that you run the EMCA command before you delete the actual cluster database instance. Also, ensure that you run the command from a different node and not from the node on which the database instance will be deleted. Note that this option can be used only in an Oracle RAC environment, so you do not need to use the `-cluster` option on the command line.

For more information, see Table 10–1 which describes EMCA command line operations.

Note: If you use the command `emca -cluster` to configure Database Control for Oracle RAC, then check the `TNS_ADMIN` environment variable on all cluster nodes. If you have a different TNS_ADMIN set for each node, then the listener for the target cannot be configured correctly. In this case, set the value of TNS_ADMIN the same on all cluster nodes before running the `emca -cluster` command.

Administering Database Control

This section provides a brief description of the activities that you can perform while administering your database using Database Control. The following sections describe some of the tasks available for managing an Oracle database:

- Starting and Stopping the Database Control Process
- Accessing the Database Home Page
- Creating Database Control Administrative Users
- Granting Access to Database Control for Non-Administrative Users
- Setting Database Control Preferences

Starting and Stopping the Database Control Process

To access Database Control from a browser, the Database Control console process `dbconsole` must be running on the database host computer. The `dbconsole` process is automatically started after installation. However, if the system must be restarted, you can start the process manually at the command line or start it as a service in Windows. You can also run commands to stop the process or view its status. If the process is stopped, then it must be manually restarted before you use Database Control.

To start or stop the database or to view the status of the `dbconsole` process from the command line:

1. Configure the operating system environment variables, as described in the "Configuring the Operating System Environment Variables" chapter in the *Oracle Database 2 Day DBA* guide.
2. Do one of the following:
   - To start the process, run the following command:

     ```
     $ ./emctl start dbconsole
     ```

   - To stop the process, run the following command:

     ```
     $ ./emctl stop dbconsole
     ```

   - To view the status of the process, run the following command:

     ```
     $ ./emctl status dbconsole
     ```

     Note: If you are prompted to set the `ORACLE_UNQNAME` environment variable, then set this variable and run the emctl command again. Set the variable to the database unique name (the value of the `DB_UNIQUE_NAME` database parameter). A typical value for `DB_UNIQUE_NAME` is `orcl`.

To start or stop Database Control as a service on Microsoft Windows:

1. Do one of the following:
   - On Windows XP and Windows Server 2003, click **Start**, then select **Control Panel**.
   - On Windows 2000 Server, click **Start**, select **Settings**, then select **Control Panel**.

   The Control Panel window opens.
2. Double-click the **Administrative Tools** icon, and then double-click the **Services** icon.

   The Services window opens.

   Oracle Database services begin with `Oracle`. The service is listed as `OracleDBConsole`*`sid`*, where *`sid`* is the system identifier (SID) for your database instance. The status of the process is listed in the Status column.
3. Double-click the **OracleDBConsole*sid*** service (for Release 10*g*). Double-click the **OracleDBConsole*dbuniquename*** service (for Release 11*g*).

   The Service Properties window opens.
4. In the Startup Type list, ensure that either **Manual** or **Automatic** is selected.
5. Do one of the following:
   - To start the service, click **Start** then click **OK**
   - To stop the service, click **Stop** then click **OK**.

Accessing the Database Home Page

The Database Home page is the main database management page in Database Control.

To access the Database Home page:

1. Ensure that the `dbconsole` process is running on the database host computer.

   See the "Starting and Stopping the Database Control Process" section on page 10-15.
2. In your Web browser, enter the following URL:

   ```
   https://hostname:portnumber/em
   ```

   For example, if you installed the database on a host computer named `comp42.example.com`, and the installer indicated that your Enterprise Manager Console HTTP port number is `1158`, then enter the following URL:

   ```
   https://comp42.example.com:1158/em
   ```

   If you have recently installed Oracle Database, you can determine the port number for Database Control on Linux and UNIX systems by viewing the `$ORACLE_HOME/install/portlist.ini` file. Otherwise, you can determine the port number for Database Control by searching for `REPOSITORY_URL` in the `ORACLE_$HOME/`*`Oracle_sid`*`/sysman/config/emd.properties` file, where *`Oracle_sid`* is the system identifier (SID) for your database instance.

   You must do the following to start the database:

   a. Click **Startup/Shutdown**, enter the host login user name and password, then enter the database login user name and password.

For the database user name and password, enter **SYS** and the password that you specified during installation.

b. Click **OK** to start the database instance, then in the Confirmation page, click **Yes** to start the database in open mode.

3. Log in to the database with a user account that is authorized to access Database Control.

   This user initially could be `SYS` or `SYSTEM`, with the password that you specified during database installation.

   While the `SYSTEM` account can be used to perform day-to-day administrative tasks, Oracle strongly recommends creating named user accounts for administering the Oracle database to enable monitoring of database activity. To back up, recover, or upgrade the database, you must log in as a `SYSDBA` user.

   Database Control displays the Database Home page.

The subpage links across the top of the page enable you to access performance, availability, and other administration pages to manage your database. The features provided by these pages are discussed in other sections of this documentation.

The various sections of the Database Home page provide information about the environment and status of the database. For example, the Alerts and Diagnostic Summary sections warn you of errors and performance problems that are impacting the operation of your database. You can click the provided links to see more detail about the problem areas, and, in some cases, to obtain recommendations for resolving the problems. These topics are discussed in the *2 Day DBA* guide, in the "Monitoring and Tuning the Database" chapter.

Creating Database Control Administrative Users

When you log in to Database Control using the `SYS`, `SYSTEM`, or `SYSMAN` user accounts, you are logging in as the Database Control super user. These are the only accounts that are automatically granted the roles and privileges required to administer Database Control itself. Examples of Database Control administration tasks include the following:

- Creating other Database Control administrators.
- Configuring e-mail and other notification methods.
- Setting alert thresholds for database metrics, such as tablespace space usage percentage exceeded or SQL response time exceeded.
- Selecting database policies to apply, so that Database Control can show alerts if a policy is violated. (An example policy is "A nonsystem user cannot use the SYSTEM or SYSAUX tablespace as its default tablespace.")
- Defining blackouts, which are time periods in which database monitoring is suspended so that maintenance operations do not skew monitoring data or generate needless alerts.

You can create Database Control administrative users who have enough privileges to administer Database Control itself, but lack the high-level database administration privileges of the `SYS` and `SYSTEM` users. This practice enables you to assign the minimum privileges required for other Database Control administrators to do their jobs, which is a best practice for database security. You can also create a Database Control administrative account for yourself, thus avoiding logging in as `SYS` or `SYSTEM` until you must perform database administration tasks.

Using the following procedure, you can assign Database Control administrative privileges to an existing database user or create a new Database Control administrative user. When you create a new Database Control administrative user, a user account is created for that user for the database. You must then decide which system privileges, object privileges, or roles to grant the user, if any, to perform database administration tasks.

To create a Database Control administrative user:

1. On any Database Control page, at the top of the page, click **Setup**.

   The Enterprise Manager Configuration page appears, showing the Overview of Setup page.

2. In the left navigation bar, click **Administrators**.

   The Administrators page appears.

3. Click **Create**.

   The Create Administrator: Properties page appears.

4. In the **Name** field, enter the name of an existing database user, or click the flashlight icon next to the field and select an existing database user.

5. Enter one or more e-mail addresses for this administrator only if you plan to set up e-mail notifications for the database.

6. Click **Review** to view a page that summarizes the information that you entered.

7. Click **Finish** to configure the database user as a Database Control administrative user.

   The Administrators page appears, showing the new administrator in the list.

Granting Access to Database Control for Non-Administrative Users

As a database administrator, you can log in to Database Control with the `SYS` or `SYSTEM` user account to perform administrative and other tasks. Non-administrative users may also want to log in to Database Control. For example, application developers may want to take advantage of the Database Control interface to create or modify tables, indexes, views, and so on. You must grant access to Database Control to these users before they can log in.

For non administrative users to have access to Database Control, they must be granted the `SELECT_CATALOG_ROLE` role.

Setting Database Control Preferences

This section discusses setting user preferences for Database Control.

About Database Control Preferences

Database Control enables you to set user preferences in the following areas:

- Notification

  These settings enable Database Control to e-mail you alerts. Alerts are notifications that the database is in an undesirable state and needs your attention. By default, the Database Home page lists all alerts. However, setup is required for e-mail notification. For more information about alerts and setting up notifications, see the "Setting Up Direct Alert Notification" section in the "Monitoring and Tuning the Database" chapter of the *Oracle Database 2 Day DBA* book.

- Blackout Administration

  During blackouts, Database Control does not collect database monitoring data or send alerts. Blackouts enable you to perform scheduled maintenance on the database without receiving needless alerts and without skewing the monitoring data.

  For example, you can stop data collections during a database backup or a hardware upgrade. If you continue monitoring during these periods, then the collected data shows trends and other monitoring information that is not the result of typical day-to-day operations. To get a more accurate, long-term picture of database performance, you can use blackouts to exclude these special-case situations from data analysis. See the "Defining Blackout Periods" section in the "Getting Started with Database Administration" chapter of the *Oracle Database 2 Day DBA* book for more information.

- Preferred Credentials

  Database Control can automatically execute many routine administrative tasks, such as backups. You can use a job scheduling system that is built into Database Control for these routine tasks. To keep your environment secure, setting up tasks for automatic execution in Enterprise Manager requires you to provide login information for the host computer and database. To avoid entering this information every time you create or run a job or task, Database Control enables you to save this information as preferred credentials. Preferred credentials are stored in the database in encrypted mode to protect them from unauthorized use. See the Setting Preferred Credentials section on page 10-15 for more information.

Defining Blackout Periods

You can define one or more one-time or repeating blackout periods where Database Control does not collect database monitoring data or send alerts.

To define a blackout period:

1. On any Database Control page, at the top of the page, click **Setup**.

   The Enterprise Manager Configuration page appears, showing the Overview of Setup page.

2. In the left-hand pane, click **Blackouts**.

   The Blackouts page appears.

3. Click **Create** to start the Create Blackout wizard.

   The Create Blackout: Properties page appears.

4. If desired, replace the default blackout name with one of your choosing.

5. If desired, in the **Comments** field, enter text that describes the purpose of the blackout.

6. In the **Reason** list, select the blackout reason that is most appropriate.

7. In the Available Targets section, in the **Type** list, select **Database Instance**.

   The fully qualified name for your database instance appears in the **Available Targets** list.

8. In the **Available Targets** list, select your instance then click the **Move** icon. You can also double-click the instance name.

   The instance name is moved from the **Available Targets** list to the **Selected Targets** list.

9. Click **Next**.

   The Create Blackout Schedule page appears.

10. On the Create Blackout Schedule page, do the following:

    a. In the Start section, schedule the blackout either immediately or for a later date and time.

    b. In the Duration section, indicate the duration of the blackout.

    c. To repeat the blackout periodically, in the Repeating section, select a repeat frequency from the **Repeat** list. Otherwise, use the default value of **Do Not Repeat**.

11. Click **Next**.

    The Review page appears.

    Review what you have entered. You can click **Back** to change a setting.

12. Click **Finish**.

    The Confirmation heading appears on the Blackouts page, and the new blackout period is shown in the list.

Setting Preferred Credentials

When you set preferred credentials, Database Control automatically fills in host computer and database login credentials for you at times when it usually prompts for these credentials. Database Control also fills in these credentials when it is about to run a job that requires credentials.

To set preferred credentials for the database:

1. From any Database Control page, at the top of the page, click **Preferences**.

   The Preferences page appears.

2. Click **Preferred Credentials** in the left-hand pane.

   The Preferred Credentials page appears, showing a table of targets.

3. In the table row for the Database Instance, click the icon under **Set Credentials**.

   The Database Preferred Credentials page appears.

4. Enter the following credentials:

   – Database credentials for Username/Password and SYSDBA Username/SYSDBA Password

   – Host credentials for Host Username/Host Password

   For example, you might typically connect as the user `SYSTEM`, use the `SYS` account for `SYSDBA` access, and use `oracle` for the host user name.

   > **Note:** The host user may require certain host privileges to run background jobs such as database backups. For example, on UNIX and Linux, the host user must belong to the OSDBA group (typically `dba`). On Windows, the host user must be a member of the Administrators group and must be granted the Log on as Batch Job log in right. See your platform documentation for more information.

5. Click **Test** to test your credentials.

A confirmation message is displayed if your credentials can be verified.

6. Click **Apply** to apply the changes.

Managing Database Control in an Oracle RAC Environment

When you create an Oracle RAC database and choose Database Control for your database management, Database Control is installed and configured automatically.

In this section:

- Verifying that Database Control is Running in an Oracle RAC Environment
- Backing Up the emkey.ora File

Verifying that Database Control is Running in an Oracle RAC Environment

To verify that Database Control has been started in your new Oracle RAC environment:

1. Ensure that the `ORACLE_UNQNAME` environment variable is set to the unique name of the database to which you want to connect, for example `orcl`. Also make sure the `ORACLE_HOME` environment variable is set to the location of the installed Oracle Database software, for example:

```
$ export ORACLE_UNQNAME=orcl
$ echo $ORACLE_HOME
/u01/app/oracle/product/11.2.0/dbhome_1

$ echo $ORACLE_UNQNAME
orcl
```

2. Go to the `$ORACLE_HOME/bin` directory.
3. Run the following command as the `oracle` user:

```
./emctl status dbconsole
```

The Enterprise Manager Control (EMCTL) utility displays the current status of the Database Control console on the current node, for example:

```
Oracle Enterprise Manager 11g Database Control Release 11.2.0.1.0
Copyright (c) 1996, 2009 Oracle Corporation.  All rights reserved.
https://racnode1.example.com:1158/em/console/aboutApplication
Oracle Enterprise Manager 11g is running.
------------------------------------------------------------------
Logs are generated in directory /u01/app/oracle/product/11.2.0/dbhome_
1/racnode1_orcl/sysman/log
```

4. If the EMCTL utility reports that Database Control is not started, use the following command to start it:

```
$ ./emctl start dbconsole
```

Following a typical installation, Database Control serves console pages from the node where the database was created. The console also monitors agents on all nodes of the cluster. However, you can configure Enterprise Manager to have multiple Database Control consoles within a cluster using EMCA.

Backing Up the emkey.ora File

During the installation, the Database Control management repository is placed in secure mode. All Enterprise Manager data is encrypted using the encryption key stored in the file `emkey.ora`. If this file is damaged or lost, and cannot be restored from a backup, you will no longer be able to use the existing Enterprise Manager repository.

The `emkey.ora` file is located in the `$ORACLE_HOME/`*`node_name_Database_name`*`/sysman/config` directory. For example, on the `racnode2` server, the encryption key file for the `orcl.example.com` database is located at `/u01/app/oracle/product/11.2.0/dbhome_1/racnode2_orcl/sysman/config/emkey.ora`.

Deleting and Creating Database Control Objects

This section discusses the process of deleting and creating Database Control objects, and how to re-create configuration files and a repository in Database Control.

Deleting Database Control Objects

This section describes the steps required to delete database control objects. There are six possible options to use when deleting Database Control objects:

- Option 1: Deleting Database Control Configuration Files Using EMCA Scripts
- Option 2: Deleting Database Control Configuration Files and Repository Objects Using An EMCA Script
- Option 3: Deleting A Database Control Repository Using EMCA Scripts
- Option 4: Deleting Database Control Configuration Files Manually
- Option 5: Deleting the Database Control Repository Using RepManager

Option 1: Deleting Database Control Configuration Files Using EMCA Scripts

To delete Database Control configuration files using EMCA scripts, run one of the following commands:

For a single-instance database:

```
$ $ORACLE_HOME/bin/emca -deconfig dbcontrol db
```

For an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -deconfig dbcontrol db -cluster
```

The `-deconfig` option removes the Enterprise Manager jobs from Scheduling before Database Control is deconfigured. Otherwise the repository, which is not dropped, could be corrupted or inconsistent if Enterprise Manager jobs were scheduled at the time Database Control is dropped. It could also cause errors during the Database Control deletion.

Option 2: Deleting Database Control Configuration Files and Repository Objects Using An EMCA Script

You can delete Database Control configuration files and repository objects using an EMCA script. Both configuration files and repository objects can be deleted using a single command.

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -deconfig dbcontrol db -repos drop
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -deconfig dbcontrol db -repos drop -cluster
```

> **Note:** This command places the database in Quiesce Mode.

Option 3: Deleting A Database Control Repository Using EMCA Scripts

You can delete a Database Control repository using EMCA scripts by using the commands listed below.

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -repos drop
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -repos drop -cluster
```

> **Note:** This command places the database in Quiesce Mode.

Option 4: Deleting Database Control Configuration Files Manually

In addition to using EMCA scripts to delete Database Control objects, you can delete objects manually, as follows:

Remove the following directories from your file system:

$ $ORACLE_HOME/*hostname_sid*

$ $ORACLE_HOME/oc4j/j2ee/OC4J_DBConsole_*hostname_sid*

On Windows, you also need to delete the Database Control service, as follows:

1. Run `regedit`.
2. Navigate to `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services`.
3. Locate the `OracleDBConsolesid` entry and delete it.

Alternatively, on Windows XP and Windows Server 2003 you can run the following from the command line, where *service_name* is the Database Control service name (typically `OracleDBConsole`*sid*):

```
sc delete service_name
```

Also available from Microsoft is The `delsrv.exe` executable is also available as a free download from Microsoft.

Option 5: Deleting the Database Control Repository Using RepManager

You can delete the Database Control repository using RepManager. This option is not as complete as the other options. You may find that dropping the repository using the command line options is a better solution. Also note that RepManager cannot be used to create a Database Control Repository.

Run the following command:

```
$ $ORACLE_HOME/sysman/admin/emdrep/bin/RepManager hostname listener_port sid
-action drop
```

> **Note:** This command places the database in Quiesce Mode.

For more information about Quiesce Mode, see My Oracle Support Note 375946.1, "Running EMCA Results in Database Quiesce" and "No New Connections or Operations Can Be Performed During the DB Control Repository Creation".

Creating Database Control Objects

This section discusses the three options that you can use to create Database Control objects.

- Option 1: Creating the DB Control Configuration Files
- Option 2: Creating the Database Control Repository Objects and Configuration Files
- Option 3: Creating the Database Control Repository Objects and Configuration Files For a Cloned Database Home

Option 1: Creating the DB Control Configuration Files

The first option that you can use to create Database Control objects is to create the Database Control configuration files. Use the following commands to create the files without going through the repository creation process.

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -cluster
```

Option 2: Creating the Database Control Repository Objects and Configuration Files

The second option you can use to create Database Control objects is to create the Database Control repository objects and configuration files.

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos create
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos create -cluster
```

Option 3: Creating the Database Control Repository Objects and Configuration Files For a Cloned Database Home

The third option you can use to create Database Control objects is to create the Database Control repository objects and configuration files for a cloned database home, as follows:

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate -cluster
```

Re-Creating Or Reconfiguring Database Control

This section discusses the two options you can use to re-create or reconfigure Database Control.

- Option 1: Re-Creating the Database Control Configuration Files Only (Leaving Repository Intact)
- Option 2: Re-Creating the Database Control Configuration Files and Repository

Option 1: Re-Creating the Database Control Configuration Files Only (Leaving Repository Intact)

The first option you can use to re-create Database Control is to re-create only the Database Control configuration files, as follows:

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -cluster
```

Option 2: Re-Creating the Database Control Configuration Files and Repository

The second option you can use to re-create Database Control is to re-create the Database Control configuration files and the repository.

Run the following command for a single-instance database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate
```

Run the following command for an Oracle RAC database:

```
$ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate -cluster
```

Note: This command places the database in Quiesce Mode.

EMCA Troubleshooting Tips

The following sections describe troubleshooting tips to consider when using EMCA to configure Database Control.

- Using EMCA After Changing the Database Listener Port
- Upgrading 11g Release 2 Grid Control Agents
- Using EMCA When the Database Host Name or IP Address Changes
- Using EMCA When the TNS Configuration Is Changed
- Using EMCA for Desktop Class Installations
- Handling EMCA Error Messages

Using EMCA After Changing the Database Listener Port

If you change the listener port of the database after you have configured Database Control, the database status will appear as `down`. To reconfigure Database Control so it uses the new listener port, run the EMCA command using the `-config dbcontrol db [-cluster]` command-line arguments.

Upgrading 11*g* Release 2 Grid Control Agents

When upgrading an Oracle Database 11*g* Release 2 database or an Oracle ASM instance that was configured for Oracle Enterprise Manager (either Database Control or a Grid Control central agent) to an Oracle Database 11*g* Release 2 (11.2) database, all Enterprise Manager targets on the relevant hosts referring to the upgraded instances are updated automatically. This occurs because the upgrade involves altering the instance Oracle home, port, or other target-associated properties. However, some of these targets on the hosts are not updated successfully during the upgrade if they are managed by an Oracle Database 11*g* Release 2 (11.2) Grid Control Agent.

To update these targets, in the Home page for the upgraded database (or Oracle ASM) target, click **Monitoring Configuration**. On the Monitoring Configuration page, you can update properties such as Oracle home, listener port, and so on to the correct values.

Using EMCA When the Database Host Name or IP Address Changes

When the database host name (including the domain name) or the IP address changes, deconfigure and then reconfigure Database Control with the `-repos recreate` flag for the `emca` command. For example, to re-create Database Control with a new host name, complete the following steps:

1. Set `ORACLE_HOSTNAME=`*`old_hostname`* in the environment.
2. De-configure the old `dbconsole` by running the following command:

   For a single-instance database:

   ```
   $ $ORACLE_HOME/bin/emca -deconfig dbcontrol db
   ```

   For Oracle RAC:

   ```
   $ $ORACLE_HOME/bin/emca -deconfig dbcontrol db -cluster
   ```

3. Un-set `ORACLE_HOSTNAME` in the environment.
4. Configure a new `dbconsole` by running one of the following commands:

   For a single-instance database, enter:

   ```
   $ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate
   ```

   For Oracle RAC, enter:

   ```
   $ $ORACLE_HOME/bin/emca -config dbcontrol db -repos recreate -cluster
   ```

Using EMCA When the TNS Configuration Is Changed

When the TNS configuration is changed, set the environment variable, then run the following command:

```
$ emca -config dbcontrol db
```

Using EMCA for Desktop Class Installations

For Desktop class installation where hostname is `localhost`, set `ORACLE_HOSTNAME=localhost` in the environment before you perform any EMCA or `emctl` operations.

Handling EMCA Error Messages

The following are some EMCA error messages and instructions for handling them.

Failed To Allocate Port(s) In the Specified Range(s)

Check the following:

- Free the ports if they are in use or use EMCA to provide free ports in command line options.
- Open up the required ports in the firewall and re-run the `emca -config dbcontrol db [-cluster]` command by specifying free ports in the command line.
- Check the `/etc/hosts` file. It should contain an entry for the physical host. Enter the following command to determine the physical host name:

  ```
  $ CMD>hostname
  ```

  Set the environment variable:

  ```
  $ set ORACLE_HOSTNAME=output of above command
  ```

  Try to run the following command:

  ```
  $ $ORACLE_HOME/bin/emca -config dbcontrol db
  ```

  By completing these steps you are fixing the environment variable and then running the `-config dbcontrol db` command again to re-attempt to allocate the ports.

Listener Is Not Up or Database Service Is Not Registered With It

Check the following:

- If the listener is not up on the specified port, then start it and rerun the EMCA script.
- Check whether the database service is registered with the listener. If not, then register it and rerun the EMCA script.
- If the listener is configured with a different host name than the physical host name, configure Database Control as follows:

  ```
  $ set ORACLE_HOSTNAME=listener host
  ```

  ```
  $ $ORACLE_HOME/bin/emca -config dbcontrol db
  ```

  If the listener port changes, then reconfigure Database Control by providing the new listener port:

  ```
  $ $ORACLE_HOME/bin/emca -config dbcontrol db
  ```

Part II

Oracle Database Structure and Storage

Part II describes database structure in terms of storage components and explains how to create and manage those components. It contains the following chapters:

- Chapter 11, "Managing Control Files"
- Chapter 12, "Managing the Redo Log"
- Chapter 13, "Managing Archived Redo Logs"
- Chapter 14, "Managing Tablespaces"
- Chapter 15, "Managing Datafiles and Tempfiles"
- Chapter 16, "Managing Undo"
- Chapter 17, "Using Oracle Managed Files"

11

Managing Control Files

In this chapter:

- 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
- Control Files Data Dictionary Views

See Also:

- *Oracle Database Concepts* for an overview of control files
- Chapter 17, "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**, which is a small binary file that records the physical structure of the database. The control file includes:

- The database name
- Names and locations of associated datafiles and 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 can also 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
- 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 initialization parameter file (see "Creating Initial Control Files" on page 11-3). The instance recognizes and opens all the listed file during startup, and 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:

- If you are not using Oracle Managed Files, then the database creates a control file and uses a default filename. The default name is operating system specific.
- If you are using Oracle Managed Files, then the initialization parameters you set to enable that feature determine the name and location of the control files, as described in Chapter 17, "Using Oracle Managed Files".
- If you are using Oracle Automatic Storage Management (Oracle ASM), you can place incomplete Oracle ASM filenames in the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` initialization parameters. Oracle ASM then automatically creates control files in the appropriate places. See the sections "About Oracle ASM Filenames" and "Creating a Database That Uses Oracle ASM" in *Oracle Automatic Storage Management Administrator's Guide* for more information.

Multiplex Control Files on Different Disks

Every Oracle Database should have at least two control files, each stored on a different physical 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 behavior of multiplexed control files is this:

- The database writes to all filenames listed for the initialization parameter `CONTROL_FILES` in the database initialization parameter file.
- The database reads only the first file listed in the `CONTROL_FILES` parameter 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 physical disks.

One way to multiplex control files is to store a control file copy on every disk drive that stores members of redo log groups, if the redo log is multiplexed. By storing control files in these locations, you minimize the risk that all control files and all groups of the 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 every time 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 11-8.

Manage the Size of Control Files

The main determinants of the size of a control file 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.
- *Oracle Database SQL Language 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.ctl,
                 /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` clause.

The size of the control file changes between some releases of Oracle Database, 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 for multiplexing by copying an existing control file to a new location and adding the file 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 name in the control file list. In both cases, to guarantee that control files do not change during the procedure, shut down the database before copying the control file.

To add a multiplexed copy of the current control file or to rename a control file:

1. Shut down the database.
2. Copy an existing control file to a new location, using operating system commands.
3. Edit the `CONTROL_FILES` parameter in the database initialization parameter file to add the new control file 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 the database name.

  For example, you would change a database name if it conflicted with another database name in a distributed environment.

> **Note:** You can change the database name and DBID (internal database identifier) using the DBNEWID utility. See *Oracle Database Utilities* for information about using this utility.

- The compatibility level is set to a value that is earlier than 10.2.0, and you must make a change to an area of database configuration that relates to any of the following parameters from the CREATE DATABASE or CREATE CONTROLFILE commands: MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, and MAXINSTANCES. If compatibility is 10.2.0 or later, you do not have to create new control files when you make such a change; the control files automatically expand, if necessary, to accommodate the new configuration information.

  For example, assume that when you created the database or re-created the control files, you set MAXLOGFILES to 3. Suppose that now you want to add a fourth redo log file group to the database with the ALTER DATABASE command. If compatibility is set to 10.2.0 or later, you can do so and the control files automatically expand to accommodate the new log file information. However, with compatibility set earlier than 10.2.0, your ALTER DATABASE command would generate an error, and you would have to first create new control files.

  For information on compatibility level, see "About The COMPATIBLE Initialization Parameter" on page 2-31.

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 (a database that formerly 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')
   RESETLOGS
   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 redo log files. Omitting a filename can cause loss of the data in that file, or loss of access to the entire database. Use caution when issuing 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 2-23.

See Also: *Oracle Database SQL Language 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 redo log files of the database.

   If you follow recommendations for control file backups as discussed in "Backing Up Control Files" on page 11-8, you will already have a list of datafiles and 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 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 comprise 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` clauses only as a last resort.

3. Back up all datafiles and 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, specify the `RESETLOGS` clause if you have lost any 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 specify the `RESETLOGS` clause if you have renamed the database. Otherwise, select the `NORESETLOGS` clause.

6. Store a backup of the new control file on an offline storage device. See "Backing Up Control Files" on page 11-8 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 in your instance parameter file 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` clause (step 5), you can recover the database with complete, closed database recovery.

   If the new control file was created using the `RESETLOGS` clause, 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.

See Also: *Oracle Database Backup and Recovery User's Guide* for information about recovering your database and methods of recovering a lost control file

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.

Troubleshooting After Creating Control Files

After issuing the `CREATE CONTROLFILE` statement, you may encounter some errors. This section describes the most common control file errors:

- 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 log to see if the database has detected inconsistencies between the data dictionary and the control file, such as a datafile in the data dictionary includes that the control file does not list.

If a datafile exists in the data dictionary but not in the new control file, the database creates a placeholder entry in the control file under the name `MISSING`*nnnn*, where *nnnn* is the file number in decimal. `MISSING`*nnnn* is flagged in the control file as being offline and requiring media recovery.

If the actual datafile corresponding to `MISSING`*nnnn* is read-only or offline normal, then you can make the datafile accessible by renaming `MISSING`*nnnn* to the name of the actual datafile. If `MISSING`*nnnn* corresponds to a datafile that was not read-only or offline normal, then you cannot use the rename operation 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.

Conversely, if a datafile listed in the control file is not present in the data dictionary, then the database removes references to it from the new control file. In both cases, the database includes an explanatory message in the alert log to let you know what was found.

Handling Errors During CREATE CONTROLFILE

If Oracle Database 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 11-6 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:

- 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';
  ```

- 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 a trace file where it can be captured and edited to reproduce the control file. View the alert log to determine the name and location of the trace file.

  See Also:

  - *Oracle Database Backup and Recovery User's Guide* for more information on backing up your control files
  - "Viewing the Alert Log" on page 9-20

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, that the control file directory is still accessible, and that 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 /u03/oracle/prod/control03.ctl  /u02/oracle/prod/control02.ctl
   ```

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 that 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.ctl  /u04/oracle/prod/control03.ctl
   ```

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.ctl,
                    /u02/oracle/prod/control02.ctl,
                    /u04/oracle/prod/control03.ctl)
   ```

3. Start SQL*Plus and open the database:

```
SQL> STARTUP
```

If you have multiplexed control files, you can get the database started up quickly by editing the CONTROL_FILES initialization parameter. Remove the bad control file from CONTROL_FILES setting and you can restart 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 want to drop control files from the database, for example, if the location of a control file is no longer appropriate. Remember that the database should have at least two control files at all times.

1. Shut down the database.
2. Edit the CONTROL_FILES parameter in the database initialization parameter file to delete the old control file 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.

Control Files Data Dictionary Views

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	Displays the names of control files as specified in the CONTROL_FILES 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
```

12

Managing the Redo Log

In this chapter:

- What Is the Redo Log?
- Planning the Redo Log
- Creating Redo Log Groups and Members
- Relocating and Renaming Redo Log Members
- Dropping Redo Log Groups and Members
- Forcing Log Switches
- Verifying Blocks in Redo Log Files
- Clearing a Redo Log File
- Redo Log Data Dictionary Views

> **See Also:** Chapter 17, "Using Oracle Managed Files" for information about redo log files that are both created and managed by the Oracle Database server

What Is the Redo Log?

The most crucial structure for recovery operations is the **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 redo log to protect the database in case of an instance failure.

Redo Threads

When speaking in the context of multiple database instances, the redo log for each database instance is also referred to as a *redo thread*. In typical configurations, only one database instance accesses an Oracle Database, so only one thread is present. In an Oracle Real Application Clusters environment, however, two or more instances concurrently access a single database and each instance has its own thread of redo. A separate redo thread for each instance avoids contention for a single set of redo log files, thereby eliminating a potential performance bottleneck.

This chapter describes how to configure and manage the redo log on a standard single-instance Oracle Database. The thread number can be assumed to be 1 in all discussions and examples of statements. For information about redo log groups in an Oracle Real Application Clusters environment, see *Oracle Real Application Clusters Administration and Deployment Guide*.

Redo Log Contents

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 undo segment data block, and the transaction table of the undo segments.

Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. Therefore, the redo log also protects rollback data. When you recover the database using redo data, the database 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 Database Writes to the Redo Log" on page 12-2) and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a **system change number** (SCN) 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 logs is the user process notified that the transaction has been committed.

Redo records can also be written to a 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 a redo log file, even though some redo records may not be committed. If necessary, the database can roll back these changes.

How Oracle Database Writes to the Redo Log

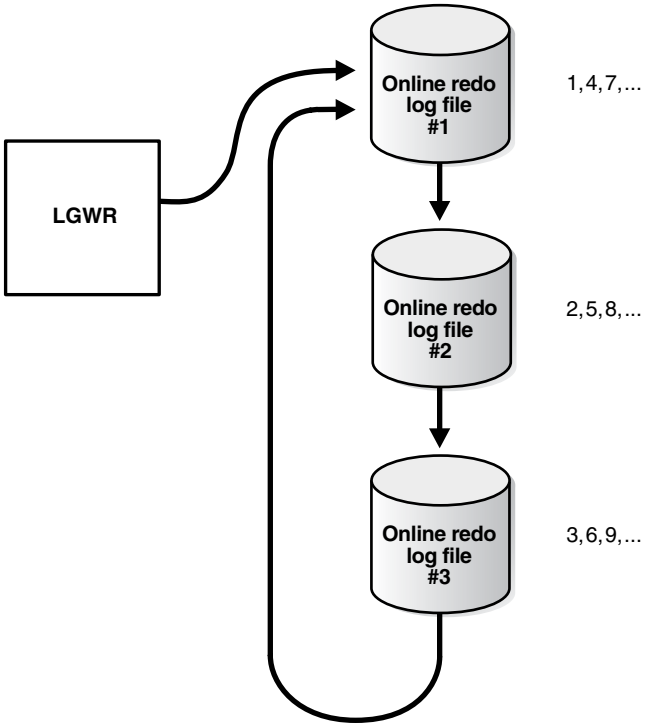
The redo log of a database consists of two or more redo log files. The database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in `ARCHIVELOG` mode). See "Managing Archived Redo Logs" on page 13-1 for more information.

LGWR writes to redo log files in a circular fashion. When the current redo log file fills, LGWR begins writing to the next available redo log file. When the last available redo log file is filled, LGWR returns to the first redo log file and writes to it, starting the cycle again. Figure 12–1 illustrates the circular writing of the redo log file. The numbers next to each line indicate the sequence in which LGWR writes to each redo log file.

Filled redo log files are available to LGWR for reuse depending on whether archiving is enabled.

- If archiving is disabled (the database is in `NOARCHIVELOG` mode), a filled redo log file is available after the changes recorded in it have been written to the datafiles.
- If archiving is enabled (the database is in `ARCHIVELOG` mode), a filled redo log file is available to LGWR after the changes recorded in it have been written to the datafiles *and* the file has been archived.

Figure 12–1 Reuse of Redo Log Files by LGWR



Active (Current) and Inactive Redo Log Files

Oracle Database uses only one redo log files at a time to store redo records written from the redo log buffer. The redo log file that LGWR is actively writing to is called the **current** redo log file.

Redo log files that are required for instance recovery are called **active** redo log files. Redo log files that are no longer required for instance recovery are called **inactive** redo log files.

If you have enabled archiving (the database is in `ARCHIVELOG` mode), then the database cannot reuse or overwrite an active online log file until one of the archiver background processes (ARC*n*) has archived its contents. If archiving is disabled (the database is in `NOARCHIVELOG` mode), then when the last redo log file is full, LGWR continues by overwriting the first available active file.

Log Switches and Log Sequence Numbers

A **log switch** is the point at which the database stops writing to one redo log file and begins writing to another. Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file. However, you can configure log switches to occur at regular intervals, regardless of whether the current redo log file is completely filled. You can also force log switches manually.

Oracle Database assigns each redo log file a new **log sequence number** every time a log switch occurs and LGWR begins writing to it. When the database archives redo log files, the archived log retains its log sequence number. A 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, the database properly applies redo log files in ascending order by using the log sequence number of the necessary archived and redo log files.

Planning the Redo Log

This section provides guidelines you should consider when configuring a database instance redo log and contains the following topics:

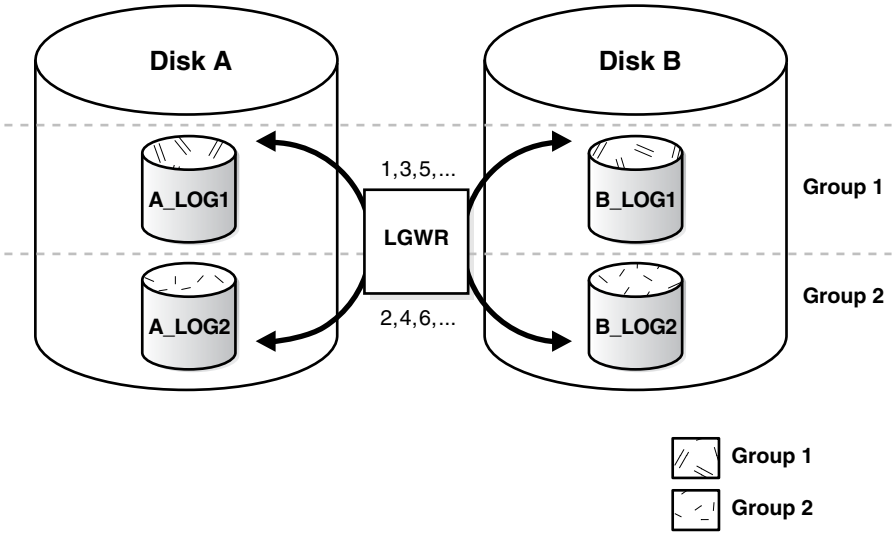
- Multiplexing Redo Log Files
- Placing Redo Log Members on Different Disks
- Planning the Size of Redo Log Files
- Planning the Block Size of Redo Log Files
- Choosing the Number of Redo Log Files
- Controlling Archive Lag

Multiplexing Redo Log Files

To protect against a failure involving the redo log itself, Oracle Database allows a **multiplexed** redo log, meaning that two or more identical copies of the redo log can be automatically maintained in separate locations. For the most benefit, these locations should be on separate disks. Even if all copies of the redo log are on the same disk, however, the redundancy can help protect against I/O errors, file corruption, and so on. When redo log files are multiplexed, LGWR concurrently writes the same redo log information to multiple identical redo log files, thereby eliminating a single point of redo log failure.

Multiplexing is implemented by creating *groups* of redo log files. A **group** consists of a redo log file and its multiplexed copies. Each identical copy is said to be a **member** of the group. Each redo log group is defined by a number, such as group 1, group 2, and so on.

Figure 12–2 Multiplexed Redo Log Files



In Figure 12–2, `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 the same size.

Each member of a log file group is concurrently active—that is, concurrently written to by LGWR—as indicated by the identical log sequence numbers assigned by LGWR. In Figure 12–2, first LGWR writes concurrently to both `A_LOG1` and `B_LOG1`. Then it writes

concurrently to both A_LOG2 and B_LOG2, and so on. LGWR never writes concurrently to members of different groups (for example, to A_LOG1 and B_LOG2).

> **Note:** Oracle recommends that you multiplex your redo log files. The loss of the log file data can be catastrophic if recovery is required. Note that when you multiplex the redo log, the database must increase the amount of I/O that it performs. Depending on your configuration, this may impact overall database performance.

Responding to Redo Log Failure

Whenever LGWR cannot write to a member of a group, the database marks that member as INVALID and writes an error message to the LGWR trace file and to the database alert log to indicate the problem with the inaccessible files. The specific reaction of LGWR when a redo log member is unavailable depends on the reason for the lack of availability, as summarized in the table that follows.

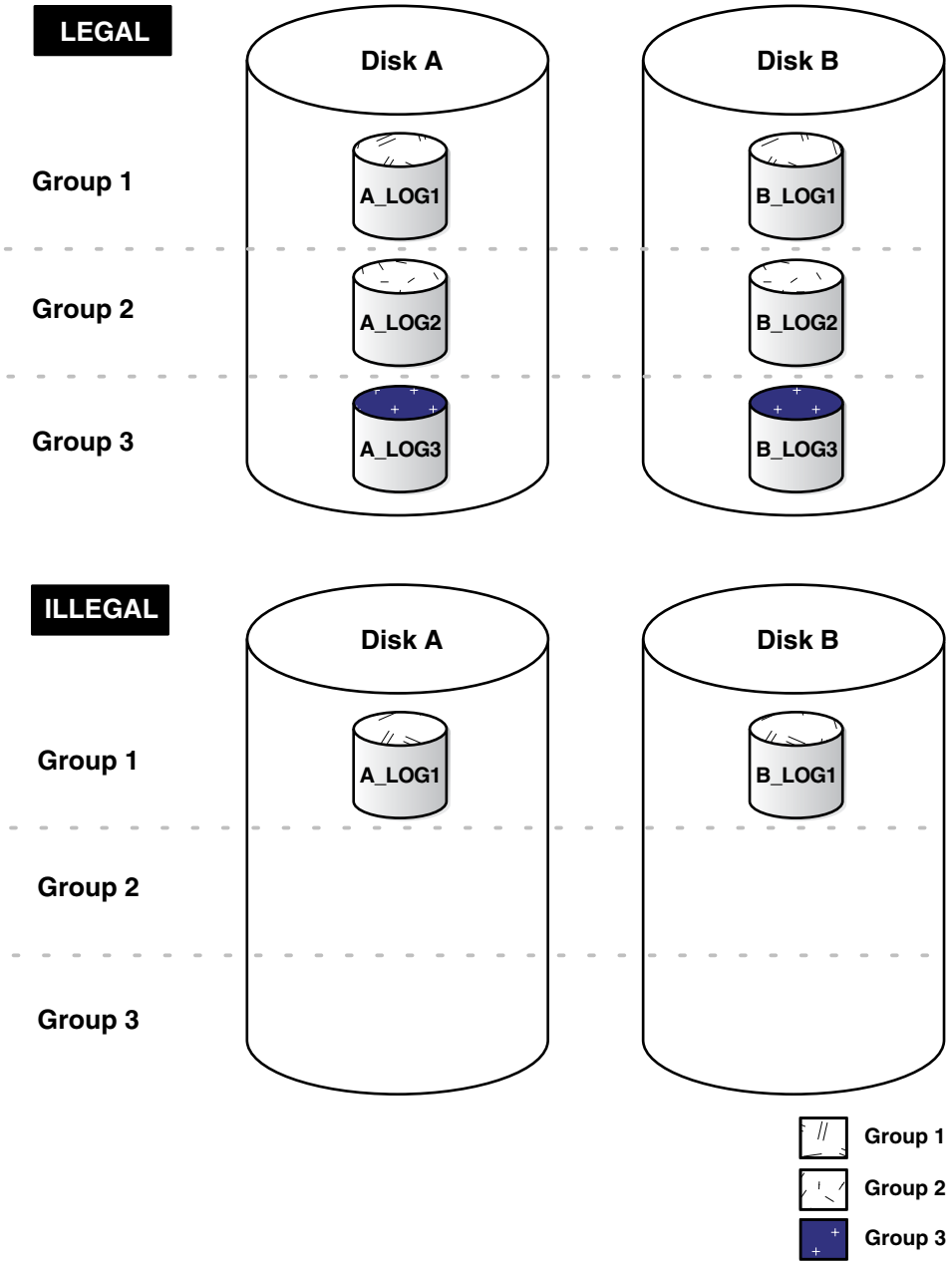
Condition	LGWR Action
LGWR can successfully write to at least one member in a group	Writing proceeds as normal. LGWR writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group must be archived	Database operation 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	Oracle Database 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 a redo log file. If the database checkpoint has moved beyond the lost redo log, media recovery is not necessary, because the database has saved the data recorded in the redo log to the datafiles. You need only drop the inaccessible redo log group. If the database did not archive the bad log, use ALTER DATABASE CLEAR UNARCHIVED LOG to disable archiving before the log can be dropped.
All members of a group suddenly become inaccessible to LGWR while it is writing to them	Oracle Database 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 need only turn the drive back on and let the database perform automatic instance recovery.

Legal and Illegal Configurations

In most cases, a multiplexed redo log should be symmetrical: all groups of the redo log should have the same number of members. However, the database does not require that a multiplexed redo log be symmetrical. For example, one group can have only one member, and other groups can have two members. This configuration protects against disk failures that temporarily affect some redo log members but leave others intact.

The only requirement for an instance redo log is that it have at least two groups. Figure 12–3 shows legal and illegal multiplexed redo log configurations. The second configuration is illegal because it has only one group.

Figure 12–3 Legal and Illegal Multiplexed Redo Log Configuration



Placing Redo Log Members on Different Disks

When setting up a multiplexed redo log, place members of a group on different physical 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 redo log members across disks to eliminate contention between the LGWR and ARC*n* background processes. For example, if you have two groups of multiplexed redo log members (a *duplexed* redo log), place each member on a different disk and set your archiving destination to a fifth disk. Doing so will avoid contention between LGWR (writing to the members) and ARC*n* (reading the members).

Datafiles should also be placed on different disks from redo log files to reduce contention in writing data blocks and redo records.

Planning the Size of Redo Log Files

When setting the size of redo log files, consider whether you will be archiving the redo log. 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 redo log group can fit on a tape and 49% of the tape storage capacity remains unused. In this case, it is better to decrease the size of the redo log files slightly, so that two log groups could be archived on each tape.

All members of the same multiplexed redo log 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.

The minimum size permitted for a redo log file is 4 MB.

> **See Also:** Your operating system–specific Oracle documentation. The default size of redo log files is operating system dependent.

Planning the Block Size of Redo Log Files

Unlike the database block size, which can be between 2K and 32K, redo log files always default to a block size that is equal to the physical sector size of the disk. Historically, this has typically been 512 bytes (512B).

Some newer high-capacity disk drives offer 4K byte (4K) sector sizes for both increased ECC capability and improved format efficiency. Most Oracle Database platforms are able to detect this larger sector size. The database then automatically creates redo log files with a 4K block size on those disks.

However, with a block size of 4K, there is increased redo wastage. In fact, the amount of redo wastage in 4K blocks versus 512B blocks is significant. You can determine the amount of redo wastage by viewing the statistics stored in the `V$SESSTAT` and `V$SYSSTAT` views.

```
SQL> SELECT name, value FROM v$sysstat WHERE name = 'redo wastage';

NAME                                 VALUE
------------------------------- ----------
redo wastage                      17941684
```

To avoid the additional redo wastage, if you are using emulation-mode disks—4K sector size disk drives that emulate a 512B sector size at the disk interface—you can override the default 4K block size for redo logs by specifying a 512B block size or, for some platforms, a 1K block size. However, you will incur a significant performance degradation when a redo log write is not aligned with the beginning of the 4K physical sector. Because seven out of eight 512B slots in a 4K physical sector are not aligned, performance degradation typically does occur. Thus, you must evaluate the trade-off between performance and disk wastage when planning the redo log block size on 4K sector size emulation-mode disks.

Beginning with Oracle Database 11*g* Release 2, you can specify the block size of online redo log files with the `BLOCKSIZE` keyword in the `CREATE DATABASE`, `ALTER DATABASE`, and `CREATE CONTROLFILE` statements. The permissible block sizes are 512, 1024, and 4096.

The following statement adds a redo log file group with a block size of 512B. The BLOCKSIZE 512 clause is valid but not required for 512B sector size disks. For 4K sector size emulation-mode disks, the BLOCKSIZE 512 clause overrides the default 4K size.

```
ALTER DATABASE orcl ADD LOGFILE
  GROUP 4 ('/u01/logs/orcl/redo04a.log','/u01/logs/orcl/redo04b.log')
  SIZE 100M BLOCKSIZE 512 REUSE;
```

To ascertain the redo log file block size, run the following query:

```
SQL> SELECT BLOCKSIZE FROM V$LOG;

BLOCKSIZE
---------
      512
```

See Also:

- *Oracle Database SQL Language Reference* for information about the ALTER DATABASE command.
- *Oracle Database Reference* for information about the V$SESSTAT and V$SYSSTAT views

Choosing the Number of Redo Log Files

The best way to determine the appropriate number of redo log files for a database instance is to test different configurations. The optimum configuration has the fewest groups possible without hampering LGWR from 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 whether the current redo log configuration is satisfactory is to examine the contents of the LGWR trace file and the database 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 redo log files before setting up or altering the configuration of an instance redo log. The following parameters limit the number of 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 redo log files for each database. Group values can range from 1 to MAXLOGFILES. When the compatibility level is set earlier than 10.2.0, the only way to override this upper limit is to re-create the database or its control file. Therefore, it is important to consider this limit before creating a database. When compatibility is set to 10.2.0 or later, you can exceed the MAXLOGFILES limit, and the control files expand as needed. If MAXLOGFILES is not specified for the CREATE DATABASE statement, then the database 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. Therefore, it is important to consider this limit before creating a database. If no MAXLOGMEMBERS parameter is specified for the CREATE DATABASE statement, then the database 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
- *Oracle Database Performance Tuning Guide*

Controlling Archive Lag

You can force all enabled redo log threads to switch their current logs at regular time intervals. In a primary/standby database configuration, changes are made available to the standby database by archiving redo logs at the primary site and then shipping them to the standby database. The changes that are being applied by the standby database can lag behind the changes that are occurring on the primary database, because the standby database must wait for the changes in the primary database redo log to be archived (into the archived redo log) and then shipped to it. To limit this lag, you can set the ARCHIVE_LAG_TARGET initialization parameter. Setting this parameter lets you specify in seconds how long that lag can be.

Setting the ARCHIVE_LAG_TARGET Initialization Parameter

When you set the ARCHIVE_LAG_TARGET initialization parameter, you cause the database to examine the current redo log of the instance periodically. If the following conditions are met, then 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 causes other threads to switch and archive their 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).

The ARCHIVE_LAG_TARGET initialization parameter specifies the target of how many seconds of redo the standby could lose in the event of a primary shutdown or failure if the Oracle Data Guard environment is not configured in a no-data-loss mode. It also provides an upper limit of how long (in 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 unpredictable behavior.

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 Redo Log Groups and Members

Plan the redo log of a database and create all required groups and members of 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 a redo log can correct redo log group availability problems.

To create new redo log groups and members, you must have the ALTER DATABASE system privilege. A database can have up to MAXLOGFILES groups.

> **See Also:** *Oracle Database SQL Language Reference* for a complete description of the ALTER DATABASE statement

Creating Redo Log Groups

To create a new group of 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 100M;
```

> **Note:** Provide full path names of new log members to specify their location. Otherwise, the files are 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 clause:

```
ALTER DATABASE
  ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo')
      SIZE 100M BLOCKSIZE 512;
```

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 unnecessary space in the control files of the database.

In the preceding statement, the BLOCKSIZE clause is optional. See "Planning the Block Size of Redo Log Files" on page 12-7 for more information.

Creating Redo Log Members

In some cases, it might not be necessary to create a complete group of 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 redo log members for an existing group, use the SQL statement ALTER DATABASE with the ADD LOGFILE MEMBER clause. 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 clause, 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 Redo Log Members

You can use operating system commands to relocate 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 redo log files is going to be removed, or if datafiles and several redo log files are stored on the same disk and should be separated to reduce contention.

To rename 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 redo log files, immediately back up the database 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 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 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 Redo Log Members

1. Shut down the database.

   ```
   SHUTDOWN
   ```

2. Copy the redo log files to the new location.

   Operating system files, such as 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 an exclamation point (!) in UNIX.

   The following example uses operating system commands (UNIX) to move the 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 redo log members.

   Use the `ALTER DATABASE` statement with the `RENAME FILE` clause to rename the database 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 redo log alterations take effect when the database is opened.

   ```
   ALTER DATABASE OPEN;
   ```

Dropping Redo Log Groups and Members

In some cases, you may want to drop an entire group of redo log members. For example, you want to reduce the number of groups in an instance redo log. In a different case, you may want to drop one or more specific redo log members. For example, if a disk failure occurs, you may need to drop all the redo log files on the

failed disk so that the database does not try to write to the inaccessible files. In other situations, particular redo log files become unnecessary. For example, a file might be stored in an inappropriate location.

Dropping Log Groups

To drop a redo log group, you must have the `ALTER DATABASE` system privilege. Before dropping a redo log group, consider the following restrictions and precautions:

- An instance requires at least two groups of redo log files, regardless of the number of members in the groups. (A group comprises one or more members.)
- You can drop a redo log group only if it is inactive. If you must drop the current group, then first force a log switch to occur.
- Make sure a 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 a 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 a 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 a redo log group, ensure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped redo log files.

When using Oracle Managed Files, the cleanup of operating systems files is done automatically for you.

Dropping Redo Log Members

To drop a redo log member, you must have the `ALTER DATABASE` system privilege. Consider the following restrictions and precautions before dropping individual redo log members:

- It is permissible to drop redo log files so that a multiplexed redo log becomes temporarily asymmetric. For example, if you use duplexed groups of 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 redo log.
- An instance always requires at least two valid groups of redo log files, regardless of the number of members in the groups. (A group comprises 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 status, use the V$LOGFILE view. A redo log file becomes INVALID if the database cannot access it. It becomes STALE if the database 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 a redo log member only if it is *not* part of an active or current group. To drop a member of an active group, first force a log switch to occur.
- Make sure the group to which a 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 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 a 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 a redo log file, ensure that the drop completed successfully, and then use the appropriate operating system command to delete the dropped 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 redo log group and starts writing to another. By default, a log switch occurs automatically when the current redo log file group fills.

You can force a log switch to make the currently active group inactive and available for 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 want to force a log switch if the currently active group must be archived at a specific time before the members of the group are completely filled. This option is useful in configurations with large 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 the database to use checksums to verify blocks in the redo log files. If you set the initialization parameter DB_BLOCK_CHECKSUM to TYPICAL (the default), the database computes a checksum for each database block when it is written to disk, including each redo log block as it is being written to the current log. The checksum is stored the header of the block.

Oracle Database uses the checksum to detect corruption in a redo log block. The database verifies the redo log block when the block is read from an archived log during recovery and when it writes the block to an archive log file. An error is raised and written to the alert log if corruption is detected.

If corruption is detected 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 of the redo log group, then archiving cannot proceed.

The value of the DB_BLOCK_CHECKSUM parameter can be changed dynamically using the ALTER SYSTEM statement.

> **Note:** There is a slight 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:** *Oracle Database Reference* for a description of the DB_BLOCK_CHECKSUM initialization parameter

Clearing a Redo Log File

A 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. The database 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.

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.

Redo Log Data Dictionary Views

The following views provide information on redo logs.

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 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:** *Oracle Database Reference* for detailed information about these views

13

Managing Archived Redo Logs

In this chapter:

- What Is the Archived Redo Log?
- Choosing Between NOARCHIVELOG and ARCHIVELOG Mode
- Controlling Archiving
- Specifying Archive Destinations
- About Log Transmission Modes
- Managing Archive Destination Failure
- Controlling Trace Output Generated by the Archivelog Process
- Viewing Information About the Archived Redo Log

See Also:

- Chapter 17, "Using Oracle Managed Files" for information about creating an archived redo log that is both created and managed by the Oracle Database server
- *Oracle Real Application Clusters Administration and Deployment Guide* for information specific to archiving in the Oracle Real Application Clusters environment

What Is the Archived Redo Log?

Oracle Database lets you save filled groups of redo log files to one or more offline destinations, known collectively as the **archived redo log**. The process of turning 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 filled members of a redo log group. It includes the redo entries and the unique log sequence number of the identical member of the redo log group. For example, if you are multiplexing your redo log, and if group 1 contains identical member files `a_log1` and `b_log1`, then the archiver process (ARC*n*) will archive one of these member files. Should `a_log1` become corrupted, then ARC*n* can still archive the identical `b_log1`. The archived redo log contains a copy of every group created since you enabled archiving.

When the database is running in `ARCHIVELOG` mode, the log writer process (LGWR) cannot reuse and hence overwrite a redo log group until it has been archived. The background process ARC*n* automates archiving operations when automatic archiving

is enabled. The database starts multiple archiver processes as needed to ensure that the archiving of filled redo logs does not fall behind.

You can use archived redo logs to:

- Recover a database
- Update a standby database
- Get information about the history of a database using the LogMiner utility

> **See Also:** The following sources document the uses for archived redo logs:
>
> - *Oracle Database Backup and Recovery User's Guide*
> - *Oracle Data Guard Concepts and Administration* discusses setting up and maintaining a standby database
> - *Oracle Database Utilities* contains instructions for using the LogMiner PL/SQL package

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

The choice of whether to enable the archiving of filled groups of 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 redo log files can require you to perform extra administrative operations.

Running a Database in NOARCHIVELOG Mode

When you run your database in `NOARCHIVELOG` mode, you disable the archiving of the redo log. The database 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.

`NOARCHIVELOG` mode protects a database from instance failure but not from media failure. Only the most recent changes made to the database, which are stored in the online redo log groups, are available for instance recovery. If a media failure occurs while the database is in `NOARCHIVELOG` mode, you can only restore the database to the point of the most recent full database backup. You cannot recover transactions subsequent to that backup.

In `NOARCHIVELOG` mode you cannot perform online tablespace backups, nor can you use online tablespace backups taken earlier while the database was in `ARCHIVELOG` mode. To restore a database operating in `NOARCHIVELOG` mode, you can use only whole database backups taken while the database is closed. 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 enable the archiving of the redo log. The database control file indicates that a group of filled redo log files cannot be reused by LGWR until the group is archived. A filled group becomes available for archiving immediately after a redo log switch occurs.

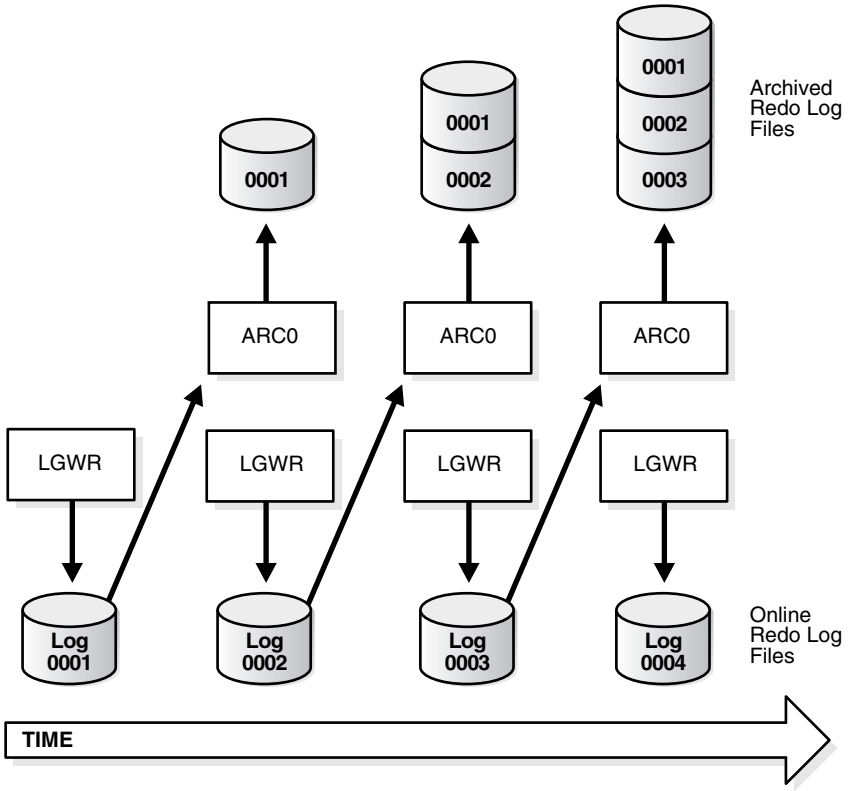
The archiving of filled groups has these advantages:

- 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.
- If you keep an archived log, you can use a backup taken while the database is open and in normal system use.
- You can keep a standby database current with its original database by continuously applying the original archived redo logs to the standby.

You can configure an instance to archive filled redo log files automatically, or you can archive manually. For convenience and efficiency, automatic archiving is usually best. Figure 13–1 illustrates how the archiver process (ARC0 in this illustration) writes filled redo log files to the database archived redo log.

If all databases in a distributed database operate in ARCHIVELOG mode, you can perform coordinated distributed database recovery. However, if any database in a distributed database is in NOARCHIVELOG mode, 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 13–1 Redo Log File Use in ARCHIVELOG Mode



Tip: It is good practice to move archived redo log files and corresponding database backups from the local disk to permanent offline storage media such as tape. A primary value of archived logs is database recovery, so you want to ensure that these logs are safe should disaster strike your primary database.

Controlling Archiving

This section describes how to set 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
- Performing Manual Archiving
- Adjusting the Number of Archiver Processes

See Also: your Oracle operating system specific documentation for additional information on controlling archiving modes

Setting the Initial Database Archiving Mode

You set the 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 by that process. After creating the database, decide whether to change the initial archiving mode.

If you specify `ARCHIVELOG` mode, you must have initialization parameters set that specify the destinations for the archived redo log files (see "Setting Initialization Parameters for Archive Destinations" on page 13-6).

Changing the Database Archiving Mode

To change the archiving mode of the database, use the `ALTER DATABASE` statement with the `ARCHIVELOG` or `NOARCHIVELOG` clause. To change the archiving mode, you must be connected to the database with administrator privileges (`AS SYSDBA`).

The following steps switch the database 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 archiving mode. You cannot change the mode from `ARCHIVELOG` to `NOARCHIVELOG` 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 during the change to `ARCHIVELOG` mode. See *Oracle Database Backup and Recovery User's Guide* for information about taking database backups.

3. Edit the initialization parameter file to include the initialization parameters that specify the destinations for the archived redo log files (see "Setting Initialization Parameters for Archive Destinations" on page 13-6).

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. Change the database 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: *Oracle Real Application Clusters Administration and Deployment Guide* for more information about switching the archiving mode when using Real Application Clusters

Performing Manual Archiving

As mentioned in "Running a Database in ARCHIVELOG Mode" on page 13-3, for convenience and efficiency, automatic archiving is usually best. However, you can configure your database for manual archiving only. To operate your database in manual archiving mode, follow the procedure described in "Changing the Database Archiving Mode" on page 13-4, but replace the `ALTER DATABASE` statement in step 5 with the following statement:

```
ALTER DATABASE ARCHIVELOG MANUAL;
```

When you operate your database in manual `ARCHIVELOG` mode, you must archive inactive groups of filled redo log files or your database operation can be temporarily suspended. To archive a filled redo log group manually, connect with administrator privileges. Ensure that the database is either mounted or open. Use the `ALTER SYSTEM` statement with the `ARCHIVE LOG` clause to manually archive filled redo log files. The following statement archives all unarchived log files:

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

When you use manual archiving mode, you cannot specify any standby databases in the archiving destinations.

Even when automatic archiving is enabled, you can use manual archiving for such actions as rearchiving an inactive group of filled redo log members to another location. In this case, it is possible for the instance to reuse the redo log group before you have finished manually archiving, and thereby overwrite the files. If this happens, the database writes an error message to the alert log.

Adjusting the Number of Archiver Processes

The `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter specifies the number of ARC*n* processes that the database initially starts. The default is four processes. There is usually no need specify this initialization parameter or to change its default value,

because the database 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 run-time overhead of starting additional ARC*n* processes, you can set the `LOG_ARCHIVE_MAX_PROCESSES` initialization parameter to specify that up to 30 ARC*n* processes be started at instance startup. The `LOG_ARCHIVE_MAX_PROCESSES` parameter is dynamic, so you can change it using the `ALTER SYSTEM` statement.

The following statement configures the database to start six ARC*n* processes upon startup:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=6;
```

The statement also has an immediate effect on the currently running instance. It increases or decreases the current number of running ARC*n* processes to six.

Specifying Archive Destinations

Before you can archive redo logs, you must determine the destination to which you will archive, and familiarize yourself with the various destination states. The dynamic performance (V$) views, listed in "Viewing Information About the Archived Redo Log" on page 13-14, provide all needed archive information.

This section contains:

- Setting Initialization Parameters for Archive Destinations
- Understanding Archive Destination Status
- Specifying Alternate Destinations

Setting Initialization Parameters for Archive Destinations

You can choose to archive redo logs to a single destination or to multiple destinations. Destinations can be local—within the local file system or an Oracle Automatic Storage Management (Oracle ASM) disk group—or remote (on a standby database). When you archive to multiple destinations, a copy of each filled redo log file is written to each destination. These redundant copies help ensure that archived logs are always available in the event of a failure at one of the destinations.

To archive to only a single destination, specify that destination using the `LOG_ARCHIVE_DEST` initialization parameter. To archive to multiple destinations, you can choose to archive to two or more locations using the `LOG_ARCHIVE_DEST_n` initialization parameters, or to archive only to a primary and secondary destination using the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` initialization parameters.

For local destinations, in addition to the local file system or an Oracle ASM disk group, you can archive to the Fast Recovery Area. The database uses the Fast Recovery Area to store and automatically manage disk space for a variety of files related to backup and recovery. See *Oracle Database Backup and Recovery User's Guide* for details about the Fast Recovery Area.

Typically, you determine archive log destinations during database planning, and you set the initialization parameters for archive destinations during database installation. However, you can use the `ALTER SYSTEM` command to dynamically add or change archive destinations after your database is running. Any destination changes that you make take effect at the next log switch (automatic or manual).

The following table summarizes the archive destination alternatives, which are further described in the sections that follow.

Method	Initialization Parameter	Host	Example
1	LOG_ARCHIVE_DEST_*n* where: *n* is an integer from 1 to 31. Archive destinations 1 to 10 are available for local or remote locations. Archive destinations 11 to 31 are available for remote locations only.	Local or remote	LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc' LOG_ARCHIVE_DEST_2 = 'LOCATION=/disk2/arc' LOG_ARCHIVE_DEST_3 = 'SERVICE=standby1'
2	LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST	Local only	LOG_ARCHIVE_DEST = '/disk1/arc' LOG_ARCHIVE_DUPLEX_DEST = '/disk2/arc'

Method 1: Using the LOG_ARCHIVE_DEST_*n* Parameter

Use the LOG_ARCHIVE_DEST_*n* parameter (where *n* is an integer from 1 to 31) to specify from one to 31 different destinations for archived logs. Each numerically suffixed parameter uniquely identifies an individual destination.

You specify the location for LOG_ARCHIVE_DEST_*n* using the keywords explained in the following table:

Keyword	Indicates	Example
LOCATION	A local file system location or Oracle ASM disk group	LOG_ARCHIVE_DEST_*n* = 'LOCATION=/disk1/arc' LOG_ARCHIVE_DEST_*n* = 'LOCATION=+DGROUP1'
LOCATION	The Fast Recovery Area	LOG_ARCHIVE_DEST_*n* = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
SERVICE	Remote archival through Oracle Net service name.	LOG_ARCHIVE_DEST_*n* = 'SERVICE=standby1'

If you use the LOCATION keyword, specify one of the following:

- A valid path name in your operating system's local file system
- An Oracle ASM disk group
- The keyword USE_DB_RECOVERY_FILE_DEST to indicate the Fast Recovery Area

If you specify SERVICE, supply a net service name that Oracle Net can resolve to a connect descriptor for a standby database. The connect descriptor contains the information necessary for connecting to the remote database.

Perform the following steps to set the destination for archived redo logs using the LOG_ARCHIVE_DEST_*n* initialization parameter:

1. Set the LOG_ARCHIVE_DEST_*n* initialization parameter to specify from one to 31 archiving locations. For example, enter:

   ```
   LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
   LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
   LOG_ARCHIVE_DEST_3 = 'LOCATION = +RECOVERY'
   ```

   If you are archiving to a standby database, then use the SERVICE keyword to specify a valid net service name. For example, enter:

   ```
   LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
   ```

2. Optionally, set the LOG_ARCHIVE_FORMAT initialization parameter, using %t to include the thread number as part of the file name, %s to include the log sequence number, and %r to include the resetlogs ID (a timestamp value represented in ub4). Use capital letters (%T, %S, and %R) to pad the file name to the left with zeroes.

> **Note:** If the COMPATIBLE initialization parameter is set to 10.0.0 or higher, the database requires the specification of resetlogs ID (%r) when you include the LOG_ARCHIVE_FORMAT parameter. The default for this parameter is operating system dependent. For example, this is the default format for UNIX:
>
> ```
> LOG_ARCHIVE_FORMAT=%t_%s_%r.dbf
> ```
>
> The incarnation of a database changes when you open it with the RESETLOGS option. Specifying %r causes the database to capture the resetlogs ID in the archived redo log file name. See *Oracle Database Backup and Recovery User's Guide* for more information about this method of recovery.

The following example shows a setting of LOG_ARCHIVE_FORMAT:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s_%r.arc
```

This setting will generate archived logs as follows for thread 1; log sequence numbers 100, 101, and 102; resetlogs ID 509210197. The identical resetlogs ID indicates that the files are all from the same database incarnation:

```
/disk1/archive/arch_1_100_509210197.arc,
/disk1/archive/arch_1_101_509210197.arc,
/disk1/archive/arch_1_102_509210197.arc

/disk2/archive/arch_1_100_509210197.arc,
/disk2/archive/arch_1_101_509210197.arc,
/disk2/archive/arch_1_102_509210197.arc

/disk3/archive/arch_1_100_509210197.arc,
/disk3/archive/arch_1_101_509210197.arc,
/disk3/archive/arch_1_102_509210197.arc
```

Method 2: Using LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST

To specify a maximum of two locations, 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. All locations must be local. Whenever the database archives a redo log, it archives it to every destination specified by either set of parameters.

Perform the following steps the use method 2:

1. 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'
   ```

2. Set the LOG_ARCHIVE_FORMAT initialization parameter as described in step 2 for method 1.

Note: If you configure a Fast Recovery Area (by setting the DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE parameters) and do not specify any local archive destinations, the database automatically selects the Fast Recovery Area as a local archive destination and sets LOG_ARCHIVE_DEST_1 to USE_DB_RECOVERY_FILE_DEST.

WARNING: You must ensure that there is sufficient disk space at all times for archive log destinations. If the database encounters a disk full error as it attempts to archive a log file, a fatal error occurs and the database stops responding. You can check the alert log for a disk full message.

See Also:

- *Oracle Database Reference* for additional information about the initialization parameters used to control the archiving of redo logs
- *Oracle Data Guard Concepts and Administration* for information about using the LOG_ARCHIVE_DEST_*n* initialization parameter for specifying a standby destination. There are additional keywords that can be specified with this initialization parameter that are not discussed in this book.
- *Oracle Database Net Services Administrator's Guide* for a discussion of net service names and connect descriptors.
- *Oracle Database Backup and Recovery User's Guide* for information about the Fast Recovery Area

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 the database 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 LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 31) initialization parameter lets you control the availability state of the specified destination (*n*).

- ENABLE indicates that the database can use the destination.
- DEFER indicates that the location is temporarily disabled.
- ALTERNATE indicates that the destination is an alternate. The availability state of an alternate destination is DEFER. If its parent destination fails, the availability state of

the alternate becomes ENABLE. ALTERNATE cannot be specified for destinations LOG_ARCHIVE_DEST_11 to LOG_ARCHIVE_DEST_31.

Specifying Alternate Destinations

To specify that a location be an archive destination only in the event of a failure of another destination, you can make it an alternate destination. Both local and remote destinations can be alternates. The following example makes LOG_ARCHIVE_DEST_4 an alternate for LOG_ARCHIVE_DEST_3:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_4 = 'LOCATION=/disk4/arch';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_3 = 'LOCATION=/disk3/arch
   ALTERNATE=LOG_ARCHIVE_DEST_4';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ALTERNATE;

SQL> SELECT dest_name, status, destination FROM v$archive_dest;

DEST_NAME               STATUS    DESTINATION
----------------------- --------- ---------------------------------------------
LOG_ARCHIVE_DEST_1      VALID     /disk1/arch
LOG_ARCHIVE_DEST_2      VALID     +RECOVERY
LOG_ARCHIVE_DEST_3      VALID     /disk3/arch
LOG_ARCHIVE_DEST_4      ALTERNATE /disk4/arch
```

About Log Transmission Modes

The two modes of transmitting archived logs to their destination are **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.

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.

See Also:

- *Oracle Data Guard Concepts and Administration*
- *Oracle Database 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. Oracle Database provides procedures to help you minimize the problems associated with destination failure. These procedures are discussed in the sections that follow:

- Specifying the Minimum Number of Successful Destinations
- Rearchiving to a Failed Destination

Specifying the Minimum Number of Successful Destinations

The optional initialization parameter `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` determines the minimum number of destinations to which the database must successfully archive a redo log group before it can reuse online log files. The default value is 1. Valid values for *n* are 1 to 2 if you are using duplexing, or 1 to 31 if you are multiplexing.

Specifying Mandatory and Optional Destinations

The `LOG_ARCHIVE_DEST_n` parameter lets you specify whether a destination is `OPTIONAL` (the default) or `MANDATORY`. The `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` parameter uses all `MANDATORY` destinations plus some number of non-standby `OPTIONAL` destinations to determine whether LGWR can overwrite the online log. The following rules apply:

- Omitting the `MANDATORY` attribute for a destination is the same as specifying `OPTIONAL`.
- You must have at least one local destination, which you can declare `OPTIONAL` or `MANDATORY`.
- The `MANDATORY` attribute can only be specified for destinations `LOG_ARCHIVE_DEST_1` through `LOG_ARCHIVE_DEST_10`.
- When you specify a value for `LOG_ARCHIVE_MIN_SUCCEED_DEST=n`, Oracle Database will treat at least one local destination as `MANDATORY`, because the minimum value for `LOG_ARCHIVE_MIN_SUCCEED_DEST` is 1.
- The `LOG_ARCHIVE_MIN_SUCCEED_DEST` value cannot be greater than the number of destinations, nor can it be greater than the number of `MANDATORY` destinations plus the number of `OPTIONAL` local destinations.
- If you `DEFER` a `MANDATORY` destination, and the database overwrites the online log without transferring the archived log to the standby site, then you must transfer the log to the standby manually.

If you are duplexing the archived logs, you can establish which destinations are mandatory or optional by using the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` parameters. The following rules apply:

- 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`.

Specifying the Number of Successful Destinations: Scenarios

You can see the relationship between the `LOG_ARCHIVE_DEST_n` and `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameters most easily through sample scenarios.

Scenario for Archiving to Optional Local Destinations In this scenario, you archive to three local destinations, each of which you declare as OPTIONAL. Table 13–1 illustrates the possible values for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* in this case.

Table 13–1 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 1

Value	Meaning
1	The database can reuse log files only if at least one of the OPTIONAL destinations succeeds.
2	The database can reuse log files only if at least two of the OPTIONAL destinations succeed.
3	The database 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, the database must successfully archive to one or more of these locations when LOG_ARCHIVE_MIN_SUCCEED_DEST is set to 1, 2, or 3.

Scenario for Archiving to Both Mandatory and Optional Destinations Consider a case in which:

- You specify two MANDATORY destinations.
- You specify two OPTIONAL destinations.
- No destination is a standby database.

Table 13–2 shows the possible values for LOG_ARCHIVE_MIN_SUCCEED_DEST=*n*.

Table 13–2 LOG_ARCHIVE_MIN_SUCCEED_DEST Values for Scenario 2

Value	Meaning
1	The database ignores the value and uses the number of MANDATORY destinations (in this example, 2).
2	The database can reuse log files even if no OPTIONAL destination succeeds.
3	The database can reuse logs only if at least one OPTIONAL destination succeeds.
4	The database can reuse logs only if both OPTIONAL destinations succeed.
5 or greater	ERROR: The value is greater than the number of destinations.

This case shows that the database 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.

Rearchiving to a Failed Destination

Use the REOPEN attribute of the LOG_ARCHIVE_DEST_*n* parameter to specify whether and when ARC*n* should attempt 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 ARC*n* 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 attribute; ARC*n* will not attempt to archive after a failure. If you

do not specify the REOPEN keyword, ARC*n* will never reopen a destination following an error.

You cannot use REOPEN to specify the number of attempts ARC*n* should make to reconnect and transfer archived logs. The REOPEN attempt either succeeds or fails.

When you specify REOPEN for an OPTIONAL destination, the database can overwrite online logs if there is an error. If you specify REOPEN for a MANDATORY destination, the database 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:

- ARC*n* reopens a destination only when *starting* an archive operation from the beginning of the log file, never *during* a current operation. ARC*n* always retries the log copy from the beginning.
- If you specified REOPEN, either with a specified time the default, ARC*n* checks to see whether the time of the recorded error plus the REOPEN interval is less than the current time. If it is, ARC*n* retries the log copy.
- The REOPEN clause successfully affects the ACTIVE=TRUE destination state. The VALID and ENABLED states are not changed.

Controlling Trace Output Generated by the Archivelog Process

Background processes always write to a trace file when appropriate. (See the discussion of this topic in "Monitoring Errors with Trace Files and the Alert Log" on page 8-1.) In the case of the archivelog process, you can control the output that is generated to the trace file. You do this by setting the LOG_ARCHIVE_TRACE initialization parameter to specify a **trace level**. The following values can be specified:

Trace Level	Meaning
0	Disable archivelog tracing. This is the default.
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*n* 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*n*/RFS heartbeat tracking.

Trace Level	Meaning
4096	Track real-time apply
8192	Track redo apply activity (media recovery or physical standby)

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. At this level, the archivelog process generates appropriate alert and trace entries for error conditions.

You can change the value of this parameter dynamically using the `ALTER SYSTEM` statement. The database must be mounted but not open. 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:** *Oracle 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 log using dynamic performance views or the `ARCHIVE LOG LIST` command.

This section contains the following topics:

- Archived Redo Logs Views
- The ARCHIVE LOG LIST Command

Archived Redo Logs Views

Several dynamic performance views contain useful information about archived redo logs, as summarized in the following table.

Dynamic Performance View	Description
`V$DATABASE`	Shows if the database is in `ARCHIVELOG` or `NOARCHIVELOG` mode and if `MANUAL` (archiving mode) has been specified.
`V$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 redo log groups for the database and indicates which need to be archived.

Dynamic Performance View	Description
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 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: *Oracle Database Reference* for detailed descriptions of dynamic performance views

The ARCHIVE LOG LIST Command

The SQL*Plus command ARCHIVE LOG LIST displays archiving information for the connected instance. For example:

```
SQL> ARCHIVE LOG LIST

Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            D:\oracle\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 destination is D:\oracle\oradata\IDDB2\archive.
- The oldest filled redo log group has a sequence number of 11160.
- The next filled redo log group to archive has a sequence number of 11163.
- The current 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

14

Managing Tablespaces

In this chapter:

- Guidelines for Managing Tablespaces
- Creating Tablespaces
- Specifying Nonstandard Block Sizes for Tablespaces
- Controlling the Writing of Redo Records
- Altering Tablespace Availability
- Using Read-Only Tablespaces
- Altering and Maintaining Tablespaces
- Renaming Tablespaces
- Dropping Tablespaces
- Managing the SYSAUX Tablespace
- Diagnosing and Repairing Locally Managed Tablespace Problems
- Migrating the SYSTEM Tablespace to a Locally Managed Tablespace
- Transporting Tablespaces Between Databases
- Tablespace Data Dictionary Views

See Also:

- *Oracle Database Concepts* for a complete discussion of database structure, space management, tablespaces, and datafiles
- Chapter 17, "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:

- Using Multiple Tablespaces
- Assigning Tablespace Quotas to Users

Using Multiple Tablespaces

Using multiple tablespaces allows you more flexibility in performing database operations. When a database has multiple tablespaces, you can:

- Separate user data from data dictionary data to reduce I/O contention.
- Separate data of one application from the data of another to prevent multiple applications from being affected if a tablespace must be taken offline.
- Store the datafiles of different tablespaces on different disk drives to reduce I/O contention.
- Take individual tablespaces offline while others remain online, providing better overall availability.
- Optimizing tablespace use by reserving a tablespace for a particular type of database use, such as high update activity, read-only activity, or temporary segment storage.
- Back up individual tablespaces.

Some operating systems set a limit on the number of files that can be open simultaneously. Such limits can affect the number of tablespaces that can be simultaneously online. To avoid exceeding your operating system limit, plan your tablespaces efficiently. Create only enough tablespaces to fulfill your needs, and create these tablespaces with as few files as possible. If you must increase the size of a tablespace, then add one or two large datafiles, or create datafiles with autoextension enabled, rather than creating many small datafiles.

Review your data in light of these factors and decide how many tablespaces you need for your database design.

Assigning 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 segment.

> **See Also:** *Oracle Database Security Guide* for information about creating users and assigning tablespace quotas.

Creating Tablespaces

Before you can create a tablespace, you must create a database to contain it. The primary tablespace in any database is the `SYSTEM` tablespace, which contains information basic to the functioning of the database server, such as the data dictionary and the system rollback segment. The `SYSTEM` tablespace is the first tablespace created at database creation. It is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the `SYSTEM` tablespace or take it offline.

The `SYSAUX` tablespace, which acts as an auxiliary tablespace to the `SYSTEM` tablespace, is also always created when you create a database. It contains the schemas used by various Oracle products and features, so that those products do not require their own tablespaces. As for the `SYSTEM` tablespace, management of the `SYSAUX` tablespace requires a higher level of security and you cannot rename or drop it. The management of the `SYSAUX` tablespace is discussed separately in "Managing the SYSAUX Tablespace" on page 14-25.

The steps for creating tablespaces vary by operating system, but the first step is always to use your operating system to create a directory structure in which your datafiles will be allocated. On most operating systems, you specify the size and fully specified filenames of datafiles when you create a new tablespace or alter an existing tablespace by adding datafiles. Whether you are creating a new tablespace or modifying an existing one, the database 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.

You can also use the `CREATE UNDO TABLESPACE` statement to create a special type of tablespace called an **undo tablespace**, which is specifically designed to contain undo records. These are records generated by the database 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 16, "Managing Undo".

The creation and maintenance of permanent and temporary tablespaces are discussed in the following sections:

- Locally Managed Tablespaces
- Bigfile Tablespaces
- Compressed Tablespaces
- Encrypted Tablespaces
- Temporary Tablespaces
- Multiple Temporary Tablespaces: Using Tablespace Groups

See Also:

- Chapter 2, "Creating and Configuring an Oracle Database" and your Oracle Database installation documentation for your operating system for information about tablespaces that are created at database creation
- *Oracle Database SQL Language Reference* for more information about the syntax and semantics of the `CREATE TABLESPACE`, `CREATE TEMPORARY TABLESPACE`, `ALTER TABLESPACE`, and `ALTER DATABASE` statements.
- "Specifying Database Block Sizes" on page 2-28 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 by using bitmaps, resulting in the following benefits:

- Fast, concurrent space operations. Space allocations and deallocations modify locally managed resources (bitmaps stored in header files).
- Enhanced performance

- Readable standby databases are allowed, because locally managed temporary tablespaces do not generate any undo or redo.
- Space allocation is simplified, because when the AUTOALLOCATE clause is specified, the database automatically selects the appropriate extent size.
- User reliance on the data dictionary is reduced, because the necessary information is stored in file headers and bitmap blocks.
- Coalescing free extents is unnecessary for locally managed tablespaces.

All tablespaces, including the SYSTEM tablespace, can be locally managed.

The DBMS_SPACE_ADMIN package provides maintenance procedures for locally managed tablespaces.

See Also:

- "Creating a Locally Managed SYSTEM Tablespace" on page 2-17, "Migrating the SYSTEM Tablespace to a Locally Managed Tablespace" on page 14-29, and "Diagnosing and Repairing Locally Managed Tablespace Problems" on page 14-26
- "Bigfile Tablespaces" on page 14-6 for information about creating another type of locally managed tablespace that contains only a single datafile or tempfile.
- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_SPACE_ADMIN package

Creating a Locally Managed Tablespace

Create a locally managed tablespace by specifying LOCAL in the EXTENT MANAGEMENT clause of the CREATE TABLESPACE statement. This is the default for new permanent tablespaces, but you must specify the EXTENT MANAGEMENT LOCAL clause to specify either the AUTOALLOCATE clause or the UNIFORM clause. You can have the database manage extents for you automatically with the AUTOALLOCATE clause (the default), or you can specify that the tablespace is managed with uniform extents of a specific size (UNIFORM).

If you expect the tablespace to contain objects of varying sizes requiring many extents with different extent sizes, then AUTOALLOCATE is the best choice. AUTOALLOCATE is also a good choice if it is not important for you to have a lot of control over space allocation and deallocation, because it simplifies tablespace management. Some space may be wasted with this setting, but the benefit of having Oracle Database manage your space most likely outweighs this drawback.

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 setting ensures that you will never have unusable space in your tablespace.

When you do not explicitly specify the type of extent management, Oracle Database determines extent management as follows:

- If the CREATE TABLESPACE statement omits the DEFAULT storage clause, then the database creates a locally managed autoallocated tablespace.
- If the CREATE TABLESPACE statement includes a DEFAULT storage clause, then the database considers the following:

- If you specified the MINIMUM EXTENT clause, the database evaluates whether the values of MINIMUM EXTENT, INITIAL, and NEXT are equal and the value of PCTINCREASE is 0. If so, the database 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, the database ignores any extent storage parameters you may specify and creates a locally managed, autoallocated tablespace.
- If you did not specify MINIMUM EXTENT clause, the database 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 a minimum extent size of 64K.

The alternative to AUTOALLOCATE is UNIFORM. which specifies that the tablespace is managed with extents of uniform size. You can specify that size in the SIZE clause of UNIFORM. If you omit SIZE, then the default size is 1M.

The following example creates a tablespace with uniform 128K extents. (In a database with 2K blocks, each extent would be equivalent to 64 database blocks). Each 128K extent 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. 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 specify the UNIFORM clause in the extent management clause but you omit the SIZE parameter, then the default extent size is 1MB. In that 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

In a locally managed tablespace, there are two methods that Oracle Database can use to manage segment space: automatic and manual. Manual segment space management uses linked lists called "freelists" to manage free space in the segment, while automatic segment space management uses bitmaps. Automatic segment space management is the more efficient method, and is the default for all new permanent, locally managed tablespaces.

Automatic segment space management delivers better space utilization than manual segment space management. It is also self-tuning, in that it scales with increasing number of users or instances. In an Oracle Real Application Clusters environment, automatic segment space management allows for a dynamic affinity of space to

instances. In addition, for many standard workloads, application performance with automatic segment space management is better than the performance of a well-tuned application using manual segment space management.

Although automatic segment space management is the default for all new permanent, locally managed tablespaces, you can explicitly enable it with the `SEGMENT SPACE MANAGEMENT AUTO` clause.

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;
```

The `SEGMENT SPACE MANAGEMENT MANUAL` clause disables automatic segment space management.

The segment space management that you specify at tablespace creation time applies to all segments subsequently created in the tablespace. You cannot change the segment space management mode of a tablespace.

Notes:

- If you set extent management to `LOCAL UNIFORM`, then you must ensure that each extent contains at least 5 database blocks.
- If you set extent management to `LOCAL AUTOALLOCATE`, and if the database block size is 16K or greater, then Oracle manages segment space by creating extents with a minimum size of 5 blocks rounded up to 64K.

Locally managed tablespaces using automatic segment space management can be created as single-file or bigfile tablespaces, as described in "Bigfile Tablespaces" on page 14-6.

Bigfile Tablespaces

A **bigfile tablespace** is a tablespace with a single, but very large (up to 4G blocks) datafile. Traditional smallfile tablespaces, in contrast, can contain multiple datafiles, but the files cannot be as large. The benefits of bigfile tablespaces are the following:

- A bigfile tablespace with 8K blocks can contain a 32 terabyte datafile. A bigfile tablespace with 32K blocks can contain a 128 terabyte datafile. The maximum number of datafiles in an Oracle Database is limited (usually to 64K files). Therefore, bigfile tablespaces can significantly enhance the storage capacity of an Oracle Database.
- Bigfile tablespaces can reduce the number of datafiles needed for a database. An additional benefit is that the `DB_FILES` initialization parameter and `MAXDATAFILES` parameter of the `CREATE DATABASE` and `CREATE CONTROLFILE` statements can be adjusted to reduce the amount of SGA space required for datafile information and the size of the control file.
- Bigfile tablespaces simplify database management by providing datafile transparency. SQL syntax for the `ALTER TABLESPACE` statement lets you perform operations on tablespaces, rather than the underlying individual datafiles.

Bigfile tablespaces are supported only for locally managed tablespaces with automatic segment space management, with three exceptions: locally managed undo tablespaces, temporary tablespaces, and the SYSTEM tablespace.

Notes:

- Bigfile tablespaces are intended to be used with Automatic Storage Management (Oracle ASM) or other logical volume managers that supports striping or RAID, and dynamically extensible logical volumes.
- Avoid creating bigfile tablespaces on a system that does not support striping because of negative implications for parallel query execution and RMAN backup parallelization.
- Using bigfile tablespaces on platforms that do not support large file sizes is not recommended and can limit tablespace capacity. See your operating system specific documentation for information about maximum supported file sizes.

Creating a Bigfile Tablespace

To create a bigfile tablespace, specify the BIGFILE keyword of the CREATE TABLESPACE statement (CREATE BIGFILE TABLESPACE ...). Oracle Database automatically creates a locally managed tablespace with automatic segment space management. You can, but need not, specify EXTENT MANAGEMENT LOCAL and SEGMENT SPACE MANAGEMENT AUTO in this statement. However, the database returns an error if you specify EXTENT MANAGEMENT DICTIONARY or SEGMENT SPACE MANAGEMENT MANUAL. The remaining syntax of the statement is the same as for the CREATE TABLESPACE statement, but you can only specify one datafile. For example:

```
CREATE BIGFILE TABLESPACE bigtbs
    DATAFILE '/u02/oracle/data/bigtbs01.dbf' SIZE 50G
...
```

You can specify SIZE in kilobytes (K), megabytes (M), gigabytes (G), or terabytes (T).

If the default tablespace type was set to BIGFILE at database creation, you need not specify the keyword BIGFILE in the CREATE TABLESPACE statement. A bigfile tablespace is created by default.

If the default tablespace type was set to BIGFILE at database creation, but you want to create a traditional (smallfile) tablespace, then specify a CREATE SMALLFILE TABLESPACE statement to override the default tablespace type for the tablespace that you are creating.

See Also: "Supporting Bigfile Tablespaces During Database Creation" on page 2-20

Identifying a Bigfile Tablespace

The following views contain a BIGFILE column that identifies a tablespace as a bigfile tablespace:

- DBA_TABLESPACES
- USER_TABLESPACES
- V$TABLESPACE

You can also identify a bigfile tablespace by the relative file number of its single datafile. That number is 1024 on most platforms, but 4096 on OS/390.

Compressed Tablespaces

You can specify that all tables created in a tablespace are compressed by default. You specify the type of table compression using the `DEFAULT` keyword, followed by one of the compression type clauses used when creating a table.

The following statement indicates that all tables created in the tablespace are to use OLTP compression, unless otherwise specified:

```
CREATE TABLESPACE ... DEFAULT COMPRESS FOR OLTP ... ;
```

You can override the default tablespace compression specification when you create a table in that tablespace.

See Also:

- "Consider Using Table Compression" on page 20-5 for information about the various types of table compression
- *Oracle Database SQL Language Reference* for the exact syntax to use when creating a tablespace with a default compression type

Encrypted Tablespaces

You can encrypt any permanent tablespace to protect sensitive data. Tablespace encryption is completely transparent to your applications, so no application modification is necessary. Encrypted tablespaces primarily protect your data from unauthorized access by means other than through the database. For example, when encrypted tablespaces are written to backup media for travel from one Oracle database to another or for travel to an off-site facility for storage, they remain encrypted. Also, encrypted tablespaces protect data from users who try to circumvent the security features of the database and access database files directly through the operating system file system.

Tablespace encryption does not address all security issues. It does not, for example, provide access control from within the database. Any user who is granted privileges on objects stored in an encrypted tablespace can access those objects without providing any kind of additional password or key.

When you encrypt a tablespace, all tablespace blocks are encrypted. All segment types are supported for encryption, including tables, clusters, indexes, LOBs (`BASICFILE` and `SECUREFILE`), table and index partitions, and so on.

Note: There is no need to use LOB encryption on `SECUREFILE` LOBs stored in an encrypted tablespace.

To maximize security, data from an encrypted tablespace is automatically encrypted when written to the undo tablespace, to the redo logs, and to any temporary tablespace. There is no need to explicitly create encrypted undo or temporary tablespaces, and in fact, you cannot specify encryption for those tablespace types.

For partitioned tables and indexes that have different partitions in different tablespaces, it is permitted to use both encrypted and non-encrypted tablespaces in the same table or index.

Tablespace encryption uses the transparent data encryption feature of Oracle Database, which requires that you create an *Oracle wallet* to store the master encryption key for the database. The wallet must be open before you can create the encrypted tablespace and before you can store or retrieve encrypted data. When you open the wallet, it is available to all session, and it remains open until you explicitly close it or until the database is shut down.

To encrypt a tablespace, you must open the database with the `COMPATIBLE` initialization parameter set to 11.1.0 or higher. The default setting for `COMPATIBLE` for a new Oracle Database 11*g* Release 2 installation is 11.2.0. Any user who can create a tablespace can create an encrypted tablespace.

Transparent data encryption supports industry-standard encryption algorithms, including the following Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- AES256
- AES192
- AES128
- 3DES168

The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys. You specify the algorithm to use when you create the tablespace, and different tablespaces can use different algorithms. Although longer key lengths theoretically provide greater security, there is a trade-off in CPU overhead. If you do not specify the algorithm in your `CREATE TABLESPACE` statement, AES128 is the default. There is no disk space overhead for encrypting a tablespace.

Examples

The following statement creates an encrypted tablespace with the default encryption algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION
DEFAULT STORAGE(ENCRYPT);
```

The following statement creates the same tablespace with the AES256 algorithm:

```
CREATE TABLESPACE securespace
DATAFILE '/u01/app/oracle/oradata/orcl/secure01.dbf' SIZE 100M
ENCRYPTION USING 'AES256'
DEFAULT STORAGE(ENCRYPT);
```

Restrictions

The following are restrictions for encrypted tablespaces:

- You cannot encrypt an existing tablespace with an `ALTER TABLESPACE` statement. However, you can use Data Pump or SQL statements such as `CREATE TABLE AS SELECT` or `ALTER TABLE MOVE` to move existing table data into an encrypted tablespace.
- Encrypted tablespaces are subject to restrictions when transporting to another database. See "Limitations on Transportable Tablespace Use" on page 14-32.
- When recovering a database with encrypted tablespaces (for example after a `SHUTDOWN ABORT` or a catastrophic error that brings down the database instance),

you must open the Oracle wallet after database mount and before database open, so the recovery process can decrypt data blocks and redo.

In addition, see *Oracle Database Advanced Security Administrator's Guide* for general restrictions for transparent data encryption.

Querying Tablespace Encryption Information

The `DBA_TABLESPACES` and `USER_TABLESPACES` data dictionary views include a column named `ENCRYPTED`. This column contains `YES` for encrypted tablespaces.

The view `V$ENCRYPTED_TABLESPACES` lists all currently encrypted tablespaces. The following query displays the name and encryption algorithm of encrypted tablespaces:

```
SELECT t.name, e.encryptionalg algorithm
FROM  v$tablespace t, v$encrypted_tablespaces e
WHERE t.ts# = e.ts#;

NAME                           ALGORITHM
------------------------------ ---------
SECURESPACE                    AES256
```

See Also:

- *Oracle Database 2 Day + Security Guide* for more information about transparent data encryption and for instructions for creating and opening wallets
- "Consider Encrypting Columns That Contain Sensitive Data" on page 20-8 for an alternative to encrypting an entire tablespace
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on using an Oracle wallet in an Oracle Real Application Clusters environment
- *Oracle Database SQL Language Reference* for information about the `CREATE TABLESPACE` statement

Temporary Tablespaces

A **temporary tablespace** contains transient data that persists only for the duration of the session. Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory and can improve the efficiency of space management operations during sorts.

Temporary tablespaces are used to store the following:

- Intermediate sort results
- Temporary tables and temporary indexes
- Temporary LOBs
- Temporary B-trees

Within a temporary tablespace, all sort operations for a particular instance share a single *sort segment*, and sort segments exist for every instance that performs sort operations that require temporary space. A sort segment is created by the first statement after startup that uses the temporary tablespace for sorting, and is released only at shutdown.

By default, a single temporary tablespace named `TEMP` is created for each new Oracle Database installation. You can create additional temporary tablespaces with the `CREATE TABLESPACE` statement. You can assign a temporary tablespace to each database

user with the CREATE USER or ALTER USER statement. A single temporary tablespace can be shared by multiple users.

You cannot explicitly create objects in a temporary tablespace.

> **Note:** The exception to the preceding statement is a temporary table. When you create a temporary table, its rows are stored in your default temporary tablespace, unless you create the table in a new temporary tablespace. See "Creating a Temporary Table" on page 20-14 for more information.

Default Temporary Tablespace

Users who are not explicitly assigned a temporary tablespace use the database default temporary tablespace, which for new installations is TEMP. You can change the default temporary tablespace for the database with the following command:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace_name;
```

To determine the current default temporary tablespace for the database, run the following query:

```
SELECT PROPERTY_NAME, PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
   PROPERTY_NAME='DEFAULT_TEMP_TABLESPACE';

PROPERTY_NAME              PROPERTY_VALUE
-------------------------- ------------------------------
DEFAULT_TEMP_TABLESPACE    TEMP
```

Space Allocation in a Temporary Tablespace

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.

When a sort operation that uses temporary space completes, allocated extents in the sort segment are not deallocated; they are just marked as free and available for reuse. The DBA_TEMP_FREE_SPACE view displays the total allocated and free space in each temporary tablespace. See "Viewing Space Usage for Temporary Tablespaces" on page 14-12 for more information. You can manually shrink a locally managed temporary tablespace that has a large amount of unused space. See "Shrinking a Locally Managed Temporary Tablespace" on page 14-23 for details.

See Also:

- *Oracle Database Security Guide* for information about creating users and assigning temporary tablespaces
- *Oracle Database Concepts* for more information about the default temporary tablespace
- *Oracle Database Reference* for more information about the V$SORT_SEGMENT, V$TEMPSEG_USAGE, and DBA_TEMP_FREE_SPACE views
- *Oracle Database Performance Tuning Guide* 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. Because of this, they enable you to perform on-disk sorting operations in a read-only or standby database.

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 default for `SIZE` is 1M. To specify another value for `SIZE`, you can do so as shown in the preceding statement.

> **Note:** On some operating systems, the database 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. See your operating system documentation to determine whether the database allocates tempfile space in this way on your system.

Creating a Bigfile Temporary Tablespace

Just as for regular tablespaces, you can create single-file (bigfile) temporary tablespaces. Use the `CREATE BIGFILE TEMPORARY TABLESPACE` statement to create a single-tempfile tablespace. See the sections "Creating a Bigfile Tablespace" on page 14-7 and "Altering a Bigfile Tablespace" on page 14-21 for information about bigfile tablespaces, but consider that you are creating temporary tablespaces that use tempfiles instead of datafiles.

Viewing Space Usage for Temporary Tablespaces

The `DBA_TEMP_FREE_SPACE` dictionary view contains information about space usage for each temporary tablespace. The information includes the space allocated and the free space. You can query this view for these statistics using the following command.

```
SELECT * from DBA_TEMP_FREE_SPACE;

TABLESPACE_NAME                    TABLESPACE_SIZE ALLOCATED_SPACE FREE_SPACE
---------------------------------- --------------- --------------- ----------
TEMP                               250609664       250609664       249561088
```

Multiple Temporary Tablespaces: Using Tablespace Groups

A **tablespace group** enables a user to consume temporary space from multiple tablespaces. Using a tablespace group, rather than a single temporary tablespace, can alleviate problems caused where one tablespace is inadequate to hold the results of a sort, particularly on a table that has many partitions. A tablespace group enables parallel execution servers in a single parallel operation to use multiple temporary tablespaces.

A tablespace group has the following characteristics:

- It contains at least one tablespace. There is no explicit limit on the maximum number of tablespaces that are contained in a group.
- It shares the namespace of tablespaces, so its name cannot be the same as any tablespace.
- You can specify a tablespace group name wherever a tablespace name would appear when you assign a default temporary tablespace for the database or a temporary tablespace for a user.

You do not explicitly create a tablespace group. Rather, it is created implicitly when you assign the first temporary tablespace to the group. The group is deleted when the last temporary tablespace it contains is removed from it.

The view `DBA_TABLESPACE_GROUPS` lists tablespace groups and their member tablespaces.

> **See Also:** *Oracle Database Security Guide* for more information about assigning a temporary tablespace or tablespace group to a user

Creating a Tablespace Group

You create a tablespace group implicitly when you include the `TABLESPACE GROUP` clause in the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement and the specified tablespace group does not currently exist.

For example, if neither `group1` nor `group2` exists, then the following statements create those groups, each of which has only the specified tablespace as a member:

```
CREATE TEMPORARY TABLESPACE lmtemp2 TEMPFILE '/u02/oracle/data/lmtemp201.dbf'
     SIZE 50M
     TABLESPACE GROUP group1;

ALTER TABLESPACE lmtemp TABLESPACE GROUP group2;
```

Changing Members of a Tablespace Group

You can add a tablespace to an existing tablespace group by specifying the existing group name in the `TABLESPACE GROUP` clause of the `CREATE TEMPORARY TABLESPACE` or `ALTER TABLESPACE` statement.

The following statement adds a tablespace to an existing group. It creates and adds tablespace `lmtemp3` to `group1`, so that `group1` contains tablespaces `lmtemp2` and `lmtemp3`.

```
CREATE TEMPORARY TABLESPACE lmtemp3 TEMPFILE '/u02/oracle/data/lmtemp301.dbf'
     SIZE 25M
     TABLESPACE GROUP group1;
```

The following statement also adds a tablespace to an existing group, but in this case because tablespace lmtemp2 already belongs to group1, it is in effect moved from group1 to group2:

```
ALTER TABLESPACE lmtemp2 TABLESPACE GROUP group2;
```

Now group2 contains both lmtemp and lmtemp2, while group1 consists of only tmtemp3.

You can remove a tablespace from a group as shown in the following statement:

```
ALTER TABLESPACE lmtemp3 TABLESPACE GROUP '';
```

Tablespace lmtemp3 no longer belongs to any group. Further, since there are no longer any members of group1, this results in the implicit deletion of group1.

Assigning a Tablespace Group as the Default Temporary Tablespace

Use the ALTER DATABASE...DEFAULT TEMPORARY TABLESPACE statement to assign a tablespace group as the default temporary tablespace for the database. For example:

```
ALTER DATABASE sample DEFAULT TEMPORARY TABLESPACE group2;
```

Any user who has not explicitly been assigned a temporary tablespace will now use tablespaces lmtemp and lmtemp2.

If a tablespace group is specified as the default temporary tablespace, you cannot drop any of its member tablespaces. You must first remove the tablespace from the tablespace group. Likewise, you cannot drop a single temporary tablespace as long as it is the default temporary tablespace.

Specifying Nonstandard Block Sizes for Tablespaces

You can create tablespaces with block sizes different from the standard database block size, which is specified by the DB_BLOCK_SIZE initialization parameter. This feature lets you transport tablespaces with unlike block sizes between databases.

Use the BLOCKSIZE clause of the CREATE TABLESPACE statement to create a tablespace with a block size different from the database standard block size. In order for the BLOCKSIZE clause to succeed, you must have already set the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE initialization parameter. Further, and the integer you specify in the BLOCKSIZE 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.

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;
```

See Also:

- "Specifying Database Block Sizes" on page 2-28
- "Setting the Buffer Cache Initialization Parameters" on page 6-15 for information about the DB_CACHE_SIZE and DB_nK_CACHE_SIZE parameter settings
- "Transporting Tablespaces Between Databases" on page 14-30

Controlling the Writing of Redo Records

For some database operations, you can control whether the database generates redo records. Without redo, no media recovery is possible. However, suppressing redo generation can improve performance, and may be appropriate for easily recoverable operations. An example of such an operation is a `CREATE TABLE...AS SELECT` statement, which can be repeated in case of database or instance failure.

Specify the `NOLOGGING` clause in the `CREATE TABLESPACE` statement to suppress redo when these operations are performed for objects within the tablespace. If you do not include this clause, or if you specify `LOGGING` instead, then the database generates redo 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.

If you have a standby database, `NOLOGGING` mode 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:

- *Oracle Database SQL Language Reference* for information about operations that can be done in `NOLOGGING` mode
- "Specifying FORCE LOGGING Mode" on page 2-23 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 it is temporarily unavailable for general use. The rest of the database remains 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.

To alter the availability of a tablespace, use the `ALTER TABLESPACE` statement. You must have the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

See Also: "Altering Datafile Availability" on page 15-6 for information about altering the availability of individual datafiles within a tablespace

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
- To rename or relocate tablespace datafiles

  See "Renaming and Relocating Datafiles" on page 15-8 for details.

When a tablespace is taken offline, the database takes all the associated files offline.

You cannot take the following tablespaces offline:

- SYSTEM
- The undo tablespace
- Temporary tablespaces

Before taking a tablespace offline, consider altering the tablespace allocation of any users who have been assigned the tablespace as a default tablespace. Doing so is advisable because those users will not be able to access objects in the tablespace while it is offline.

You can specify any of the following parameters as part of the ALTER TABLESPACE...OFFLINE statement:

Clause	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 you specify OFFLINE NORMAL, the database takes a checkpoint for all datafiles of the tablespace as it takes them offline. NORMAL 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 you specify OFFLINE TEMPORARY, the database 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 clause, 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 the database taking a checkpoint on any of the datafiles. When you specify OFFLINE IMMEDIATE, 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 NOARCHIVELOG mode.

Caution: If you must take a tablespace offline, use the NORMAL clause (the default) if possible. This setting guarantees that the tablespace will not require recovery to come back online, 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 settings.

The following example takes the users tablespace offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

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.

If a tablespace to be brought online was not taken offline "cleanly" (that is, using the NORMAL clause of the ALTER TABLESPACE OFFLINE statement), you must first perform media recovery on the tablespace before bringing it online. Otherwise, the database returns an error and the tablespace remains offline.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for information about performing media recovery

The following statement brings the users tablespace online:

```
ALTER TABLESPACE users ONLINE;
```

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. Read-only tablespaces also provide a way to protecting historical data so that users cannot modify it. 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, as described in "Transporting Tablespaces Between Databases" on page 14-30.

You can drop items, such as tables or indexes, from a read-only tablespace, but you cannot create or alter objects in a read-only 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 use 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: "Transporting Tablespaces Between Databases" on page 14-30

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 must be applied to the tablespace.
- The tablespace cannot be the active undo tablespace or `SYSTEM` tablespace.
- The tablespace must not currently be involved in an online backup, because 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 the database 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 can issue the `ALTER TABLESPACE...READ ONLY` statement while the database is processing transactions. After the statement is issued, the tablespace is put into a transitional read-only state. No transactions are allowed to make further changes (using DML statements) to the tablespace. If a transaction attempts further changes, it is terminated and rolled back. However, transactions that already made changes and that attempt no further changes are allowed to commit or roll back.

The `ALTER TABLESPACE...READ ONLY` statement waits for the following transactions to either commit or roll back before returning: transactions that have pending or uncommitted changes to the tablespace and that were started before you issued the statement. If a transaction started before the statement remains active, but rolls back to a savepoint, rolling back its changes to the tablespace, then the statement no longer waits for this active transaction.

If you find it is taking a long time for the `ALTER TABLESPACE` statement to complete, you can identify the transactions that are preventing the read-only state from taking effect. You can then notify the owners of those transactions and decide whether to terminate the transactions, if necessary.

The following example identifies the transaction entry for the `ALTER TABLESPACE...READ ONLY` statement and displays 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$TRANSACTION view. Displaying this view sorted by ascending start SCN lists the transactions in execution order. From the preceding example, you already know the session address of the transaction entry for the read-only statement, and you can now locate it in the V$TRANSACTION view. All transactions with smaller start SCN, which indicates an earlier execution, 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
```

You can now find the owners of the blocking transactions.

```
SELECT T.SES_ADDR, S.USERNAME, S.MACHINE
   FROM V$SESSION S, V$TRANSACTION T
   WHERE T.SES_ADDR = S.SADDR
   ORDER BY T.SES_ADDR

SES_ADDR USERNAME             MACHINE
-------- -------------------- --------------------
800352A0 DAVIDB               DAVIDBLAP             --> Contact this user
80035A50 MIKEL                LAB61                 --> Contact this user
80034AF0 DBA01                STEVEFLAP
80037910 NICKD                NICKDLAP
```

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, because no changes can be made to it.

See Also: *Oracle Database Backup and Recovery 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` do 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. To avoid accessing read-only files for these operations, take those files 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.

Altering and Maintaining Tablespaces

This section covers various subjects that relate to altering and maintaining tablespaces.

This section covers the following topics:

- Increasing the Size of a Tablespace
- Altering a Locally Managed Tablespace
- Altering a Bigfile Tablespace
- Altering a Locally Managed Temporary Tablespace
- Shrinking a Locally Managed Temporary Tablespace

Increasing the Size of a Tablespace

You can increase the size of a tablespace by either increasing the size of a datafile in the tablespace or adding one. See "Creating Datafiles and Adding Datafiles to a Tablespace" on page 15-4 for more information.

Additionally, you can enable automatic file extension (`AUTOEXTEND`) to datafiles and bigfile tablespaces. See "Enabling and Disabling Automatic Extension for a Datafile" on page 15-5.

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. Coalescing free extents is unnecessary for locally managed tablespaces. However, you can use the `ALTER TABLESPACE` statement on locally managed tablespaces for some operations, including the following:

- Adding a datafile. For example:

```
ALTER TABLESPACE lmtbsb
   ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- Altering tablespace availability (`ONLINE`/`OFFLINE`). See "Altering Tablespace Availability" on page 14-15.
- Making a tablespace read-only or read/write. See "Using Read-Only Tablespaces" on page 14-17.
- Renaming a datafile, or enabling or disabling the autoextension of the size of a datafile in the tablespace. See Chapter 15, "Managing Datafiles and Tempfiles".

Altering a Bigfile Tablespace

Two clauses of the `ALTER TABLESPACE` statement support datafile transparency when you are using bigfile tablespaces:

- `RESIZE`: The `RESIZE` clause lets you resize the single datafile in a bigfile tablespace to an absolute size, without referring to the datafile. For example:

```
ALTER TABLESPACE bigtbs RESIZE 80G;
```

- `AUTOEXTEND` (used outside of the `ADD DATAFILE` clause):

  With a bigfile tablespace, you can use the `AUTOEXTEND` clause outside of the `ADD DATAFILE` clause. For example:

```
ALTER TABLESPACE bigtbs AUTOEXTEND ON NEXT 20G;
```

An error is raised if you specify an ADD DATAFILE clause for a bigfile tablespace.

Altering a Locally Managed Temporary Tablespace

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.

You can use ALTER TABLESPACE to add a tempfile, take a tempfile offline, or bring a tempfile online, as illustrated in the following examples:

```
ALTER TABLESPACE lmtemp
   ADD TEMPFILE '/u02/oracle/data/lmtemp02.dbf' SIZE 18M REUSE;

ALTER TABLESPACE lmtemp TEMPFILE OFFLINE;
ALTER TABLESPACE lmtemp TEMPFILE ONLINE;
```

Note: You cannot take a temporary tablespace offline. Instead, you take its tempfile offline. The view V$TEMPFILE displays online status for a tempfile.

The ALTER DATABASE statement can be used to alter tempfiles.

The following statements take offline and bring online tempfiles. They behave identically to the last two ALTER TABLESPACE statements in the previous example.

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' ONLINE;
```

The following statement resizes a tempfile:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' RESIZE 18M;
```

The following statement drops a tempfile and deletes its operating system file:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
    INCLUDING DATAFILES;
```

The tablespace to which this tempfile belonged remains. A message is written to the alert log for the tempfile 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 log.

It is also possible to use the ALTER DATABASE statement to enable or disable the automatic extension of an existing tempfile, and to rename a tempfile. See *Oracle Database SQL Language Reference* for the required syntax.

Note: To rename a tempfile, you take the tempfile offline, use operating system commands to rename or relocate the tempfile, and then use the ALTER DATABASE RENAME FILE command to update the database controlfiles.

Shrinking a Locally Managed Temporary Tablespace

Large sort operations performed by the database may result in a temporary tablespace growing and occupying a considerable amount of disk space. After the sort operation completes, the extra space is not released; it is just marked as free and available for reuse. Therefore, a single large sort operation might result in a large amount of allocated temporary space that remains unused after the sort operation is complete. For this reason, the database enables you to shrink locally managed temporary tablespaces and release unused space.

You use the `SHRINK SPACE` clause of the `ALTER TABLESPACE` statement to shrink a temporary tablespace, or the `SHRINK TEMPFILE` clause of the `ALTER TABLESPACE` statement to shrink a specific tempfile of a temporary tablespace. Shrinking frees as much space as possible while maintaining the other attributes of the tablespace or tempfile. The optional `KEEP` clause defines a minimum size for the tablespace or tempfile.

Shrinking is an online operation, which means that user sessions can continue to allocate sort extents if needed, and already-running queries are not affected.

The following example shrinks the locally managed temporary tablespace `lmtmp1` to a size of 20M.

```
ALTER TABLESPACE lmtemp1 SHRINK SPACE KEEP 20M;
```

The following example shrinks the tempfile `lmtemp02.dbf` of the locally managed temporary tablespace `lmtmp2`. Because the `KEEP` clause is omitted, the database attempts to shrink the tempfile to the minimum possible size.

```
ALTER TABLESPACE lmtemp2 SHRINK TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

Renaming Tablespaces

Using the `RENAME TO` clause of the `ALTER TABLESPACE`, you can rename a permanent or temporary tablespace. For example, the following statement renames the `users` tablespace:

```
ALTER TABLESPACE users RENAME TO usersts;
```

When you rename a tablespace the database updates all references to the tablespace name in the data dictionary, control file, and (online) datafile headers. The database does not change the tablespace ID so if this tablespace were, for example, the default tablespace for a user, then the renamed tablespace would show as the default tablespace for the user in the `DBA_USERS` view.

The following affect the operation of this statement:

- The `COMPATIBLE` parameter must be set to 10.0.0 or higher.
- If the tablespace being renamed is the `SYSTEM` tablespace or the `SYSAUX` tablespace, then it will not be renamed and an error is raised.
- If any datafile in the tablespace is offline, or if the tablespace is offline, then the tablespace is not renamed and an error is raised.
- If the tablespace is read only, then datafile headers are not updated. This should not be regarded as corruption; instead, it causes a message to be written to the alert log indicating that datafile headers have not been renamed. The data dictionary and control file are updated.

- If the tablespace is the default temporary tablespace, then the corresponding entry in the database properties table is updated and the DATABASE_PROPERTIES view shows the new name.
- If the tablespace is an undo tablespace and if the following conditions are met, then the tablespace name is changed to the new tablespace name in the server parameter file (SPFILE).
  - The server parameter file was used to start up the database.
  - The tablespace name is specified as the UNDO_TABLESPACE for any instance.

  If a traditional initialization parameter file (PFILE) is being used then a message is written to the alert log stating that the initialization parameter file must be manually changed.

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. You must have the DROP TABLESPACE system privilege to drop a tablespace.

> **Caution:** Once a tablespace has been dropped, the data in the tablespace is not recoverable. Therefore, ensure 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 Database to delete the operating system files (datafiles) that constituted the dropped tablespace. If you do not direct the database 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 undo data needed to roll back uncommitted transactions, 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 clause. Use the CASCADE CONSTRAINTS clause 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 users tablespace and its associated datafiles:

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

A message is written to the alert log 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 log.

> **See Also:** "Dropping Datafiles" on page 15-11

Managing the SYSAUX Tablespace

The `SYSAUX` tablespace was installed as an auxiliary tablespace to the `SYSTEM` tablespace when you created your database. Some database components that formerly created and used separate tablespaces now occupy the `SYSAUX` tablespace.

If the `SYSAUX` tablespace becomes unavailable, core database functionality will remain operational. The database features that use the `SYSAUX` tablespace could fail, or function with limited capability.

Monitoring Occupants of the SYSAUX Tablespace

The list of registered occupants of the `SYSAUX` tablespace are discussed in "About the SYSAUX Tablespace" on page 2-17. These components can use the `SYSAUX` tablespace, and their installation provides the means of establishing their occupancy of the `SYSAUX` tablespace.

You can monitor the occupants of the `SYSAUX` tablespace using the `V$SYSAUX_OCCUPANTS` view. This view lists the following information about the occupants of the `SYSAUX` tablespace:

- Name of the occupant
- Occupant description
- Schema name
- Move procedure
- Current space usage

View information is maintained by the occupants.

> **See Also:** *Oracle Database Reference* for a detailed description of the `V$SYSAUX_OCCUPANTS` view

Moving Occupants Out Of or Into the SYSAUX Tablespace

You will have an option at component install time to specify that you do not want the component to reside in `SYSAUX`. Also, if you later decide that the component should be relocated to a designated tablespace, you can use the move procedure for that component, as specified in the `V$SYSAUX_OCCUPANTS` view, to perform the move.

The move procedure also lets you move a component from another tablespace into the `SYSAUX` tablespace.

Controlling the Size of the SYSAUX Tablespace

The `SYSAUX` tablespace is occupied by several database components, and its total size is governed by the space consumed by those components. The space consumed by the components, in turn, depends on which features or functionality are being used and on the nature of the database workload.

The largest portion of the SYSAUX tablespace is occupied by the Automatic Workload Repository (AWR). The space consumed by the AWR is determined by several factors, including the number of active sessions in the system at any given time, the snapshot interval, and the historical data retention period. A typical system with an average of 10 concurrent active sessions may require approximately 200 to 300 MB of space for its AWR data.

The following table provides guidelines on sizing the SYSAUX tablespace based on the system configuration and expected load.

Parameter/Recommendation	Small	Medium	Large
Number of CPUs	2	8	32
Number of concurrently active sessions	10	20	100
Number of user objects: tables and indexes	500	5,000	50,000
Estimated SYSAUX size at steady state with default configuration	500 MB	2 GB	5 GB

You can control the size of the AWR by changing the snapshot interval and historical data retention period. For more information on managing the AWR snapshot interval and retention period, see *Oracle Database Performance Tuning Guide*.

Another major occupant of the SYSAUX tablespace is the embedded Oracle Enterprise Manager repository. This repository is used by Oracle Enterprise Manager Database Control to store its metadata. The size of this repository depends on database activity and on configuration-related information stored in the repository.

Other database components in the SYSAUX tablespace will grow in size only if their associated features (for example, Oracle Text and Oracle Streams) are in use. If the features are not used, then these components do not have any significant effect on the size of the SYSAUX tablespace.

Diagnosing and Repairing Locally Managed Tablespace Problems

Oracle Database includes the DBMS_SPACE_ADMIN package, which is a collection of aids for diagnosing and repairing problems in locally managed tablespaces.

DBMS_SPACE_ADMIN Package Procedures

The following table lists the DBMS_SPACE_ADMIN package procedures. See *Oracle Database PL/SQL Packages and Types Reference* for details on each procedure.

Procedure	Description
ASSM_SEGMENT_VERIFY	Verifies the integrity of segments created in tablespaces that have automatic segment space management enabled. Outputs a dump file named *sid*_ora_*process_id*.trc to the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Use SEGMENT_VERIFY for tablespaces with manual segment space management.

Procedure	Description
ASSM_TABLESPACE_VERIFY	Verifies the integrity of tablespaces that have automatic segment space management enabled. Outputs a dump file named *sid*_ora_*process_id*.trc to the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Use TABLESPACE_VERIFY for tablespaces with manual segment space management.
DROP_EMPTY_SEGMENTS	Drops segments from empty tables or table partitions and dependent objects
MATERIALIZE_DEFERRED_SEGMENTS	Materializes segments for tables and table partitions with deferred segment creation and their dependent objects.
SEGMENT_CORRUPT	Marks the segment corrupt or valid so that appropriate error recovery can be done
SEGMENT_DROP_CORRUPT	Drops a segment currently marked corrupt (without reclaiming space)
SEGMENT_DUMP	Dumps the segment header and bitmap blocks of a specific segment to a dump file named *sid*_ora_*process_id*.trc in the location that corresponds to the Diag Trace entry in the V$DIAG_INFO view. Provides an option to select a slightly abbreviated dump, which includes segment header and includes bitmap block summaries, without percent-free states of each block.
SEGMENT_VERIFY	Verifies the consistency of the extent map of the segment
TABLESPACE_FIX_BITMAPS	Marks the appropriate DBA range (extent) as free or used in bitmap
TABLESPACE_FIX_SEGMENT_STATES	Fixes the state of the segments in a tablespace in which migration was stopped
TABLESPACE_MIGRATE_FROM_LOCAL	Migrates a locally managed tablespace to dictionary-managed tablespace
TABLESPACE_MIGRATE_TO_LOCAL	Migrates a dictionary-managed tablespace to a locally managed tablespace
TABLESPACE_REBUILD_BITMAPS	Rebuilds the appropriate bitmaps
TABLESPACE_REBUILD_QUOTAS	Rebuilds quotas for a specific tablespace
TABLESPACE_RELOCATE_BITMAPS	Relocates the bitmaps to the specified destination
TABLESPACE_VERIFY	Verifies that the bitmaps and extent maps for the segments in the tablespace are synchronized

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:

- *Oracle Database PL/SQL Packages and Types Reference* for details about the DBMS_SPACE_ADMIN package
- "Viewing ADR Locations with the V$DIAG_INFO View" on page 9-9

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 rebuild 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 rebuild 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 that 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

Use the TABLESPACE_MIGRATE_TO_LOCAL procedure to migrate a dictionary-managed tablespace to a locally managed tablespace. This operation is done online, but space management operations are blocked until the migration has been completed. Therefore, you can read or modify data while the migration is in progress, but if you are loading a large amount of data that requires the allocation of additional extents, then the operation may be blocked.

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). Allow the system to choose the bitmap allocation unit. 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 an allocation unit size, it must be a factor of the unit size calculated by the system.

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 the dictionary-managed tablespace.
- 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.
- The SYSAUX tablespace is offline.
- The system is in restricted mode.
- There is a cold backup of the database.

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 use the dictionary-managed tablespaces in read/write mode, then 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
- About Transporting Tablespaces Across Platforms
- Limitations on Transportable Tablespace Use
- Compatibility Considerations for Transportable Tablespaces
- Transporting Tablespaces Between Databases: A Procedure and Example
- Using Transportable Tablespaces: Scenarios

Note: You must be using the Enterprise Edition of Oracle Database Release 8*i* or later to generate a transportable tablespace set. However, you can use any edition of Oracle Database 8*i* or later to import a transportable tablespace set into an Oracle database on the same platform. To import a transportable tablespace set into an Oracle database on a different platform, both databases must have compatibility set to at least 10.0.0. See "Compatibility Considerations for Transportable Tablespaces" on page 14-34 for a discussion of database compatibility for transporting tablespaces across release levels.

Introduction to Transportable Tablespaces

You can use the Transportable Tablespaces feature to copy a set of tablespaces from one Oracle Database to another.

Note: This method for transporting tablespaces requires that you place the tablespaces to be transported in read-only mode until you complete the transporting process. If this is undesirable, you can use the Transportable Tablespaces from Backup feature, described in *Oracle Database Backup and Recovery User's Guide*.

The tablespaces being transported can be either dictionary managed or locally managed. Starting with Oracle9*i*, the transported tablespaces are not required to be of the same block size as the destination database standard block size.

Moving data using transportable tablespaces is much faster than performing either an export/import or unload/load of the same data. This is because the datafiles containing all of the actual data are just copied to the destination location, and you use Data Pump to transfer only the metadata of the tablespace objects to the new database.

Note: Beginning with Oracle Database 11g Release 1, you must use Data Pump for transportable tablespaces. The only circumstance under which you can use the original import and export utilities, IMP and EXP, is for a backward migration of XMLType data to a database version 10*g* Release 2 or earlier. See *Oracle Database Utilities* for more information on these utilities and to *Oracle XML DB Developer's Guide* for more information on XMLTypes.

The transportable tablespace feature is useful in several scenarios, including:

- Exporting and importing partitions in data warehousing tables
- Publishing structured data on CDs
- Copying multiple read-only versions of a tablespace on multiple databases
- Archiving historical data
- Performing tablespace point-in-time-recovery (TSPITR)

These scenarios are discussed in "Using Transportable Tablespaces: Scenarios" on page 14-44.

There are two ways to transport a tablespace:

- Manually, following the steps described in this section. This involves issuing commands to SQL*Plus, RMAN, and Data Pump.
- Using the Transport Tablespaces Wizard in Enterprise Manager

  To run the Transport Tablespaces Wizard:

  1. Log in to Enterprise Manager with a user that has the `EXP_FULL_DATABASE` role.
  2. At the top of the Database Home page, click **Data Movement** to view the Data Movement page.
  3. Under Move Database Files, click **Transport Tablespaces**.

  See Also: *Oracle Database Data Warehousing Guide* for information about using transportable tablespaces in a data warehousing environment

About Transporting Tablespaces Across Platforms

Starting with Oracle Database Release 10*g*, you can transport tablespaces across platforms. This functionality can be used to:

- Allow a database to be migrated from one platform to another
- Provide an easier and more efficient means for content providers to publish structured data and distribute it to customers running Oracle Database on different platforms
- Simplify the distribution of data from a data warehouse environment to data marts, which are often running on smaller platforms
- Enable the sharing of read-only tablespaces between Oracle Database installations on different operating systems or platforms, assuming that your storage system is accessible from those platforms and the platforms all have the same endianness, as described in the sections that follow.

Many, but not all, platforms are supported for cross-platform tablespace transport. You can query the `V$TRANSPORTABLE_PLATFORM` view to see the platforms that are supported, and to determine each platform's endian format (byte ordering). The following query displays the platforms that support cross-platform tablespace transport:

```
SQL> COLUMN PLATFORM_NAME FORMAT A36
SQL> SELECT * FROM V$TRANSPORTABLE_PLATFORM ORDER BY PLATFORM_NAME;

PLATFORM_ID PLATFORM_NAME                        ENDIAN_FORMAT
----------- ------------------------------------ --------------
          6 AIX-Based Systems (64-bit)           Big
```

```
 16 Apple Mac OS                        Big
 19 HP IA Open VMS                      Little
 15 HP Open VMS                         Little
  5 HP Tru64 UNIX                       Little
  3 HP-UX (64-bit)                      Big
  4 HP-UX IA (64-bit)                   Big
 18 IBM Power Based Linux               Big
  9 IBM zSeries Based Linux             Big
 10 Linux IA (32-bit)                   Little
 11 Linux IA (64-bit)                   Little
 13 Linux x86 64-bit                    Little
  7 Microsoft Windows IA (32-bit)       Little
  8 Microsoft Windows IA (64-bit)       Little
 12 Microsoft Windows x86 64-bit        Little
 17 Solaris Operating System (x86)      Little
 20 Solaris Operating System (x86-64)   Little
  1 Solaris[tm] OE (32-bit)             Big
  2 Solaris[tm] OE (64-bit)             Big

19 rows selected.
```

If the source platform and the destination platform are of different endianness, then an additional step must be done on either the source or destination platform to convert the tablespace being transported to the destination format. If they are of the same endianness, then no conversion is necessary and tablespaces can be transported as if they were on the same platform.

Before a tablespace can be transported to a different platform, the datafile header must identify the platform to which it belongs. In an Oracle Database with compatibility set to 10.0.0 or later, you can accomplish this by making the datafile read/write at least once.

Limitations on Transportable Tablespace Use

Be aware of the following limitations as you plan to transport tablespaces:

- The source and the destination databases must use compatible database character sets. That is, one of the following must be true:
  - The database character sets of the source and the target databases are the same.
  - The source database character set is a strict (binary) subset of the target database character set, and the following three conditions are true:
    - The source database is in version 10.1.0.3 or higher.
    - The tablespaces to be transported contain no table columns with character length semantics or the maximum character width is the same in both the source and target database character sets.
    - The tablespaces to be transported contain no columns with the `CLOB` data type, or the source and the target database character sets are both single-byte or both multibyte.
  - The source database character set is a strict (binary) subset of the target database character set, and the following two conditions are true:
    - The source database is in a version lower than 10.1.0.3.
    - The maximum character width is the same in the source and target database character sets.

- The source and the target databases must use compatible national character sets. That is:
  - The national character sets of the source and target databases are the same
  - The source database is in version 10.1.0.3 or higher and the tablespaces to be transported contain no columns with `NCHAR`, `NVARCHAR2`, or `NCLOB` data type.
- You cannot transport a tablespace to a destination database that contains a tablespace of the same name. However, before the transport operation, you can rename either the tablespace to be transported or the destination tablespace.

  Note: The subset-superset relationship between character sets recognized by Oracle Database is documented in the *Oracle Database Globalization Support Guide*.

- Objects with underlying objects (such as materialized views) or contained objects (such as partitioned tables) are not transportable unless all of the underlying or contained objects are in the tablespace set.
- Encrypted tablespaces have the following the limitations:
  - Before transporting an encrypted tablespace, you must copy the Oracle wallet manually to the destination database, unless the master encryption key is stored in a Hardware Security Module (HSM) device instead of an Oracle wallet. When copying the wallet, the wallet password remains the same in the destination database. However, it is recommended that you change the password on the destination database so that each database has its own wallet password. See *Oracle Database Advanced Security Administrator's Guide* for information about HSM devices, about determining the location of the Oracle wallet, and about changing the wallet password with Oracle Wallet Manager.
  - You cannot transport an encrypted tablespace to a database that already has an Oracle wallet for transparent data encryption. In this case, you must use Oracle Data Pump to export the tablespace's schema objects and then import them to the destination database. You can optionally take advantage of Oracle Data Pump features that enable you to maintain encryption for the data while it is being exported and imported. See *Oracle Database Utilities* for more information.
  - You cannot transport an encrypted tablespace to a platform with different endianness.
- Tablespaces that do not use block encryption but that contain tables with encrypted columns cannot be transported. You must use Oracle Data Pump to export and import the tablespace's schema objects. You can take advantage of Oracle Data Pump features that enable you to maintain encryption for the data while it is being exported and imported. See *Oracle Database Utilities* for more information.
- Beginning with Oracle Database 10*g* Release 2, you can transport tablespaces that contain XMLTypes. Beginning with Oracle Database 11*g* Release 1, you must use only Data Pump to export and import the tablespace metadata for tablespaces that contain XMLTypes.

  The following query returns a list of tablespaces that contain XMLTypes:

```
select distinct p.tablespace_name from dba_tablespaces p,
  dba_xml_tables x, dba_users u, all_all_tables t where
  t.table_name=x.table_name and t.tablespace_name=p.tablespace_name
```

```
and x.owner=u.username
```

See *Oracle XML DB Developer's Guide* for information on XMLTypes.

Transporting tablespaces with XMLTypes has the following limitations:

- The destination database must have XML DB installed.
- Schemas referenced by XMLType tables cannot be the XML DB standard schemas.
- Schemas referenced by XMLType tables cannot have cyclic dependencies.
- XMLType tables with row level security are not supported, because they cannot be exported or imported.
- If the schema for a transported XMLType table is not present in the destination database, it is imported and registered. If the schema already exists in the destination database, an error is returned unless the `ignore=y` option is set.
- If an XMLType table uses a schema that is dependent on another schema, the schema that is depended on is not exported. The import succeeds only if that schema is already in the destination database.

Additional limitations include the following:

SYSTEM Tablespace Objects You cannot transport the SYSTEM tablespace or objects owned by the user SYS. Some examples of such objects are PL/SQL, Java classes, callouts, views, synonyms, users, privileges, dimensions, directories, and sequences.

Opaque Types Types whose interpretation is application-specific and opaque to the database (such as `RAW`, `BFILE`, and the AnyTypes) can be transported, but they are not converted as part of the cross-platform transport operation. Their actual structure is known only to the application, so the application must address any endianness issues after these types are moved to the new platform. Types and objects that use these opaque types, either directly or indirectly, are also subject to this limitation.

Floating-Point Numbers `BINARY_FLOAT` and `BINARY_DOUBLE` types are transportable using Data Pump.

Compatibility Considerations for Transportable Tablespaces

When you create a transportable tablespace set, Oracle Database computes the lowest compatibility level at which the destination database must run. This is referred to as the compatibility level of the transportable set. Beginning with Oracle Database 11*g*, a tablespace can always be transported to a database with the same or higher compatibility setting, whether the destination database is on the same or a different platform. The database signals an error if the compatibility level of the transportable set is higher than the compatibility level of the destination database.

The following table shows the minimum compatibility requirements of the source and destination tablespace in various scenarios. The source and destination database need not have the same compatibility setting.

Table 14–1 Minimum Compatibility Requirements

	Minimum Compatibility Setting	
Transport Scenario	**Source Database**	**Destination Database**
Databases on the same platform	8.0	8.0
Tablespace with different database block size than the destination database	9.0	9.0
Databases on different platforms	10.0	10.0

Transporting Tablespaces Between Databases: A Procedure and Example

The following list of tasks summarizes the process of transporting a tablespace. Details for each task are provided in the subsequent example.

> **Note:** This method of generating a transportable tablespace requires that you temporarily make the tablespace read-only. If this is undesirable, you can use the alternate method known as transportable tablespace from backup. See *Oracle Database Backup and Recovery User's Guide* for details.

1. For cross-platform transport, check the endian format of both platforms by querying the `V$TRANSPORTABLE_PLATFORM` view.

   Ignore this task if you are transporting your tablespace set to the same platform.

2. Pick a self-contained set of tablespaces.

3. At the source database, place the set of tablespaces in read-only mode and generate a transportable tablespace set.

   A **transportable tablespace set** (or **transportable set**) consists of datafiles for the set of tablespaces being transported and an export file containing structural information (metadata) for the set of tablespaces. You use Data Pump to perform the export.

   If you are transporting the tablespace set to a platform with different endianness from the source platform, you must convert the tablespace set to the endianness of the destination platform. You can perform a source-side conversion at this step in the procedure, or you can perform a destination-side conversion as part of Task 4.

4. Transport the tablespace set.

   Copy the datafiles and the export file to a place that is accessible to the destination database.

   If you transported the tablespace set to a platform with different endianness from the source platform, and you have not performed a source-side conversion to the endianness of the destination platform, perform a destination-side conversion now.

5. (Optional) Restore tablespaces to read/write mode.

6. At the destination database, import the tablespace set.

   Invoke the Data Pump utility to import the metadata for the tablespace set.

Example

These tasks for transporting a tablespace are illustrated more fully in the example that follows, where it is assumed the following datafiles and tablespaces exist:

Tablespace	Datafile
sales_1	/u01/app/oracle/oradata/salesdb/sales_101.dbf
sales_2	/u01/app/oracle/oradata/salesdb/sales_201.dbf

Task 1: Determine if Platforms are Supported and Determine Endianness

This task is only necessary if you are transporting the tablespace set to a platform different from the source platform.

If you are transporting the tablespace set to a platform different from the source platform, then determine if cross-platform tablespace transport is supported for both the source and destination platforms, and determine the endianness of each platform. If both platforms have the same endianness, no conversion is necessary. Otherwise you must do a conversion of the tablespace set either at the source or destination database.

If you are transporting sales_1 and sales_2 to a different platform, you can execute the following query on each platform. If the query returns a row, the platform supports cross-platform tablespace transport.

```
SELECT d.PLATFORM_NAME, ENDIAN_FORMAT
     FROM V$TRANSPORTABLE_PLATFORM tp, V$DATABASE d
     WHERE tp.PLATFORM_NAME = d.PLATFORM_NAME;
```

The following is the query result from the source platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Solaris[tm] OE (32-bit)            Big
```

The following is the result from the destination platform:

```
PLATFORM_NAME                      ENDIAN_FORMAT
---------------------------------- --------------
Microsoft Windows IA (32-bit)      Little
```

You can see that the endian formats are different and thus a conversion is necessary for transporting the tablespace set.

Task 2: 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. To transport a subset of a partition table, you must exchange the partitions into tables.

See *Oracle Database VLDB and Partitioning Guide* for information about exchanging partitions.

- 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 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 `LOB`s outside the set of tablespaces.
- An XML DB schema (*.xsd) that was registered by user A imports a global schema that was registered by user B, and the following is true: the default tablespace for user A is tablespace A, the default tablespace for user B is tablespace B, and only tablespace A is included in 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 *Oracle Database Backup and Recovery User's Guide*.

Note: The default for transportable tablespaces is to check for self containment rather than full containment.

The following statement can be used to 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 example illustrates 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.

```
SQL> 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 task, one choice for bypassing the integrity constraint violation is to not export the integrity constraints.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TTS` package
- *Oracle Database Backup and Recovery User's Guide* for information specific to using the `DBMS_TTS` package for TSPITR

Task 3: 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 completing the following steps:

1. Start SQL*Plus and connect to the database as an administrator or as a user who has either the `ALTER TABLESPACE` or `MANAGE TABLESPACE` system privilege.

   See "Connecting to the Database with SQL*Plus" on page 1-7 for instructions.

2. Make all tablespaces in the set read-only.

   ```
   SQL> ALTER TABLESPACE sales_1 READ ONLY;

   Tablespace altered.

   SQL> ALTER TABLESPACE sales_2 READ ONLY;

   Tablespace altered.
   ```

3. Invoke the Data Pump export utility as user `system` and specify the tablespaces in the transportable set.

   ```
   SQL> HOST

   $ expdp system dumpfile=expdat.dmp directory=data_pump_dir
           transport_tablespaces=sales_1,sales_2 logfile=tts_export.log

   Password: password
   ```

   You must always specify `TRANSPORT_TABLESPACES`, which determines the mode of the export operation. In this example:

   - The `DUMPFILE` parameter specifies the name of the structural information export file to be created, `expdat.dmp`.
   - The `DIRECTORY` parameter specifies the directory object that points to the operating system or Oracle Automatic Storage Management location of the dump file. You must create the `DIRECTORY` object before invoking Data Pump, and you must grant the `READ` and `WRITE` object privileges on the directory to `PUBLIC`. See *Oracle Database SQL Language Reference* for information on the `CREATE DIRECTORY` command.

> **Note:** The directory object DATA_PUMP_DIR is automatically created when you install Oracle Database. Read and write access to this directory is automatically granted to the DBA role, and thus to users SYS and SYSTEM. If the DIRECTORY parameter is omitted, DATA_PUMP_DIR is used as the default directory.

- The LOGFILE parameter specifies the file name of the log file to be written by the export utility. The log file is written to the same directory as the dump file.
- EXPDP prompts for the password for the system account if you do not specify it on the command line.
- Triggers and indexes are included in the export operation by default.

To perform a transport tablespace operation with a strict containment check, use the TRANSPORT_FULL_CHECK parameter, as shown in the following example:

```
expdp system dumpfile=expdat.dmp directory=data_pump_dir
      transport_tablespaces=sales_1,sales_2 transport_full_check=y
      logfile=tts_export.log
```

In this case, the Data Pump export utility verifies that there are no dependencies between the objects inside the transportable set and objects outside the transportable set. If the tablespace set being transported is not self-contained, then the export fails and indicates that the transportable set is not self-contained. You must then return to Task 2 to resolve all violations.

> **Notes:** The Data Pump utility is used to export only data dictionary structural information (metadata) for the tablespaces. No actual data is unloaded, so this operation goes relatively quickly even for large tablespace sets.

4. Check the log file for errors, and take note of the dump file and datafiles that you must transport to the destination database. EXPDP outputs the names and paths of these files in messages like these:

```
******************************************************************************
Dump file set for SYSTEM.SYS_EXPORT_TRANSPORTABLE_01 is:
  /u01/app/oracle/admin/salesdb/dpdump/expdat.dmp
******************************************************************************
Datafiles required for transportable tablespace SALES_1:
  /u01/app/oracle/oradata/salesdb/sales_101.dbf
Datafiles required for transportable tablespace SALES_2:
  /u01/app/oracle/oradata/salesdb/sales_201.dbf
```

5. When finished, exit back to SQL*Plus:

```
$ EXIT
```

> **See Also:** *Oracle Database Utilities* for information about using the Data Pump utility

If sales_1 and sales_2 are being transported to a different platform, and the endianness of the platforms is different, and if you want to convert before transporting the tablespace set, then convert the datafiles composing the sales_1 and sales_2 tablespaces:

1. From SQL*Plus, return to the host system:

```
SQL> HOST
```

2. Start RMAN and connect to the source database:

```
$ RMAN TARGET /

Recovery Manager: Release 11.2.0.0.1

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

connected to target database: salesdb (DBID=3295731590)
```

3. Use the RMAN `CONVERT TABLESPACE` command to convert the datafiles into a temporary location on the source platform.

   In this example, assume that the temporary location, directory `/tmp`, has already been created. The converted datafiles are assigned names by the system.

```
RMAN> CONVERT TABLESPACE sales_1,sales_2
2> TO PLATFORM 'Microsoft Windows IA (32-bit)'
3> FORMAT '/tmp/%U';

Starting conversion at source at 30-SEP-08
using channel ORA_DISK_1
channel ORA_DISK_1: starting datafile conversion
input datafile file number=00007 name=/u01/app/oracle/oradata/salesdb/sales_
101.dbf
converted datafile=/tmp/data_D-SALESDB_I-1192614013_TS-SALES_1_FNO-7_03jru08s
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:45
channel ORA_DISK_1: starting datafile conversion
input datafile file number=00008 name=/u01/app/oracle/oradata/salesdb/sales_
201.dbf
converted datafile=/tmp/data_D-SALESDB_I-1192614013_TS-SALES_2_FNO-8_04jru0aa
channel ORA_DISK_1: datafile conversion complete, elapsed time: 00:00:25
Finished conversion at source at 30-SEP-08
```

   See Also: *Oracle Database Backup and Recovery Reference* for a description of the RMAN `CONVERT` command

4. Exit Recovery Manager:

```
RMAN> exit
Recovery Manager complete.
```

Task 4: Transport the Tablespace Set

Complete the following steps:

1. Transport *both the datafiles and the export (dump) file* of the tablespaces to a place that is accessible to the destination database. To accomplish this, do one of the following:

   - If no endianness conversion of the tablespace set is needed, or if you already converted the tablespace set:

     a. Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object, or to any other directory of your choosing.

        Run the following query to determine the location of `DATA_PUMP_DIR`:

```
SELECT * FROM DBA_DIRECTORIES WHERE DIRECTORY_NAME = 'DATA_PUMP_DIR';
```

```
OWNER      DIRECTORY_NAME   DIRECTORY_PATH
---------- ---------------- ------------------------------------
SYS        DATA_PUMP_DIR    C:\app\orauser\admin\orawin\dpdump\
```

b. Transport the datafiles to the location of the existing datafiles of the destination database.

On the UNIX and Linux platforms, this location is typically /u01/app/oracle/oradata/*SID*/ or +*DISKGROUP*/*SID*/datafile/.

Note: If you converted the datafiles, obtain the new names and locations of the datafiles from the `CONVERT TABLESPACE` command output, as shown in Step 3 of Task 3: Generate a Transportable Tablespace Set.

- If you intend to perform endianness conversion after transporting to the destination host:

  a. Transport the dump file to the directory pointed to by the `DATA_PUMP_DIR` directory object, or to any other directory of your choosing.

  b. Transport the datafiles to a temporary location on the destination host (for example, /tmp or C:\TEMP). During conversion, you can move the converted datafiles to the location of the existing datafiles of the destination database.

  Tip: If both the source and destination are file systems, you can transport using:

  - Any facility for copying flat files (for example, an operating system copy utility or ftp)
  - The `DBMS_FILE_TRANSFER` package
  - RMAN
  - Any facility for publishing on CDs

  If either the source or destination is an Oracle Automatic Storage Management (Oracle ASM) disk group, you can use:

  - ftp to or from the `/sys/asm` virtual folder in the XML DB repository

    See *Oracle Automatic Storage Management Administrator's Guide* for more information.
  - The `DBMS_FILE_TRANSFER` package
  - RMAN

Caution: Exercise caution when using the UNIX `dd` utility to copy raw-device files between databases. The `dd` utility can be used to copy an entire source raw-device file, or it can be invoked with options that instruct it to copy only a specific range of blocks from the source raw-device file.

It is difficult to ascertain actual datafile size for a raw-device file because of hidden control information that is stored as part of the datafile. Thus, it is advisable when using the `dd` utility to specify copying the entire source raw-device file contents.

2. If you are transporting the tablespace set to a platform with endianness that is different from the source platform, and you have not yet converted the tablespace set, do so now with RMAN.

   The following example places the converted datafiles into C:\app\orauser\oradata\orawin\, which is the location of the existing datafiles for the destination database:

```
C:\>RMAN TARGET /

Recovery Manager: Release 11.2.0.0.1

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

connected to target database: ORAWIN (DBID=3462152886)

RMAN> CONVERT DATAFILE
2>'C:\Temp\sales_101.dbf',
3>'C:\Temp\sales_201.dbf'
4>TO PLATFORM="Microsoft Windows IA (32-bit)"
5>FROM PLATFORM="Solaris[tm] OE (32-bit)"
6>DB_FILE_NAME_CONVERT=
7>'C:\Temp\', 'C:\app\orauser\oradata\orawin\'
8> PARALLELISM=4;
```

   You identify the datafiles by filename, not by tablespace name. Until the tablespace metadata is imported, the destination instance has no way of knowing the desired tablespace names. The source and destination platforms are optional. RMAN determines the source platform by examining the datafile, and the destination platform defaults to the platform of the host running the conversion.

   See Also: "Copying Files Using the Database Server" on page 15-12 for information about using the `DBMS_FILE_TRANSFER` package to copy the files that are being transported and their metadata

Task 5: (Optional) Restore Tablespaces to Read/Write Mode

Make the transported tablespaces read/write again at the source database, as follows:

```
ALTER TABLESPACE sales_1 READ WRITE;
ALTER TABLESPACE sales_2 READ WRITE;
```

You can postpone this task to first ensure that the import process succeeds.

Task 6: Import 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_*n*K_CACHE_SIZE initialization parameter entry in the receiving database 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 *Oracle Database Reference* for information about specifying values for the DB_*n*K_CACHE_SIZE initialization parameter.

Any privileged user can perform this task. To import a tablespace set, complete the following steps:

1. Import the tablespace metadata using the Data Pump Import utility, impdp:

```
impdp system dumpfile=expdat.dmp directory=data_pump_dir
   transport_datafiles=
   c:\app\orauser\oradata\orawin\sales_101.dbf,
   c:\app\orauser\oradata\orawin\sales_201.dbf
   remap_schema=sales1:crm1  remap_schema=sales2:crm2
   logfile=tts_import.log

Password: password
```

   In this example we specify the following:

   - The DUMPFILE parameter specifies the exported file containing the metadata for the tablespaces to be imported.
   - The DIRECTORY parameter specifies the directory object that identifies the location of the dump file.
   - The TRANSPORT_DATAFILES parameter identifies all of the datafiles containing the tablespaces to be imported.
   - The REMAP_SCHEMA parameter changes the ownership of database objects. If you do not specify REMAP_SCHEMA, all database objects (such as tables and indexes) are created in the same user schema as in the source database, and those users must already exist in the destination database. If they do not exist, then the import utility returns an error. In this example, objects in the tablespace set owned by sales1 in the source database will be owned by crm1 in the destination database after the tablespace set is imported. Similarly, objects owned by sales2 in the source database will be owned by crm2 in the destination database. In this case, the destination database is not required to have users sales1 and sales2, but must have users crm1 and crm2.
   - The LOGFILE parameter specifies the file name of the log file to be written by the import utility. The log file is written to the directory from which the dump file is read.

   After this statement executes successfully, all tablespaces in the set being copied remain in read-only mode. Check the import log file to ensure that 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 Data Pump import utility as follows:

```
impdp system parfile='par.f'
```

where the parameter file, `par.f` contains the following:

```
DUMPFILE=expdat.dmp
DIRECTORY=data_pump_dir
TRANSPORT_DATAFILES=
C:\app\orauser\oradata\orawin\sales_101.dbf,
C:\app\orauser\oradata\orawin\sales_201.dbf
REMAP_SCHEMA=sales1:crm1  REMAP_SCHEMA=sales2:crm2
LOGFILE=tts_import.log
```

See Also: *Oracle Database Utilities* for information about using the import utility

2. If required, put the tablespaces into read/write mode on the destination database.

Using Transportable Tablespaces: Scenarios

The following sections describe some uses for transportable tablespaces:

- Transporting and Attaching Partitions for Data Warehousing
- Publishing Structured Data on CDs
- Mounting the Same Tablespace Read-Only on Multiple Databases
- Archiving Historical Data Using Transportable Tablespaces
- Using Transportable Tablespaces to Perform TSPITR

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. 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 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 non-prefixed 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 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 non-prefixed 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` clause. 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, the exchange operation might fail. For example, if the `jan98` partition of `sales` did not come from the same staging database, the preceding 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 add the CD contents to an existing database without having to copy the datafiles from the CD to disk storage. For example, suppose on a Windows NT system D: drive is the CD drive. You can import a transportable set with datafile `catalog.f` and export file `expdat.dmp` as follows:

```
IMPDP system/password DUMPFILE=expdat.dmp DIRECTORY=dpump_dir
   TRANSPORT_DATAFILES='D:\catalog.f'
```

You can remove the CD while the database is still up. Subsequent queries to the tablespace return an error indicating that the database 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, the database 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`, the database reads the file only when someone queries the transported tablespace. Thus, when transporting a tablespace from 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.

The following are two scenarios for mounting the same tablespace read-only on multiple databases:

- The tablespace originates in a database that is separate from the databases that will share the tablespace.

  You generate a transportable set in the source database, put the transportable set onto a disk that is accessible to all databases, and then import the metadata into each database on which you want to mount the tablespace.

- The tablespace already belongs to one of the databases that will share the tablespace.

  It is assumed that the datafiles are already on a shared disk. In the database where the tablespace already exists, you make the tablespace read-only, generate the transportable set, and then import the tablespace into the other databases, leaving the datafiles in the same location on the shared disk.

You can make a disk accessible by multiple computers in several ways. You can use either a cluster file system or raw disk. You can also use network file system (NFS), but be aware 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.

Archiving Historical Data Using Transportable Tablespaces

Since a transportable tablespace set is a self-contained set of files that can be imported 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:** *Oracle Database 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:** *Oracle Database Backup and Recovery User's Guide* for information about how to perform TSPITR using transportable tablespaces

Moving Databases Across Platforms Using Transportable Tablespaces

You can use the transportable tablespace feature to migrate a database to a different platform by creating a new database on the destination platform and performing a transport of all the user tablespaces. See *Oracle Database Backup and Recovery User's Guide* for more information.

You cannot transport the `SYSTEM` tablespace. Therefore, objects such as sequences, PL/SQL packages, and other objects that depend on the `SYSTEM` tablespace are not transported. You must either create these objects manually on the destination database, or use Data Pump to transport the objects that are not moved by transportable tablespace.

Tablespace Data Dictionary Views

The following data dictionary and dynamic performance views provide useful information about the tablespaces of a database.

View	Description
`V$TABLESPACE`	Name and number of all tablespaces from the control file.
`V$ENCRYPTED_TABLESPACES`	Name and encryption algorithm of all encrypted tablespaces.
`DBA_TABLESPACES`, `USER_TABLESPACES`	Descriptions of all (or user accessible) tablespaces.
`DBA_TABLESPACE_GROUPS`	Displays the tablespace groups and the tablespaces that belong to them.
`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.
`DBA_TEMP_FREE_SPACE`	Displays the total allocated and free space in each temporary tablespace.
`V$DATAFILE`	Information about all datafiles, including tablespace number of owning tablespace.
`V$TEMPFILE`	Information about all tempfiles, including tablespace number of owning tablespace.

View	Description
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 TEMPORARY type.
V$TEMPSEG_USAGE	Describes temporary (sort) segment usage by user for temporary or permanent tablespaces.

The following are just a few examples of using some of these views.

> **See Also:** *Oracle Database Reference* for complete description of these views

Example 1: Listing Tablespaces and Default Storage Parameters

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
```

Example 2: Listing the Datafiles and Associated Tablespaces of a Database

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/DBF/RBS01.DBF                   1536  RBS
/U02/ORACLE/IDDB3/DBF/SYSTEM01.DBF                6586  SYSTEM
/U02/ORACLE/IDDB3/DBF/TEMP01.DBF                  6400  TEMP
```

```
/U02/ORACLE/IDDB3/DBF/TESTTBS01.DBF              6400  TESTTBS
/U02/ORACLE/IDDB3/DBF/USERS01.DBF                 384  USERS
```

Example 3: Displaying Statistics for Free Space (Extents) of Each Tablespace

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 ensure that there is enough space in the containing tablespace.

15

Managing Datafiles and Tempfiles

In this chapter:

- Guidelines for Managing Datafiles
- Creating Datafiles and Adding Datafiles to a Tablespace
- Changing Datafile Size
- Altering Datafile Availability
- Renaming and Relocating Datafiles
- Dropping Datafiles
- Verifying Data Blocks in Datafiles
- Copying Files Using the Database Server
- Mapping Files to Physical Devices
- Datafiles Data Dictionary Views

See Also:

- Chapter 17, "Using Oracle Managed Files" for information about creating datafiles and tempfiles that are both created and managed by the Oracle Database server
- *Oracle Database Concepts*

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.

Note: Tempfiles are a special class of datafiles that are associated only with temporary tablespaces. Information in this chapter applies to both datafiles and tempfiles except where differences are noted. Tempfiles are further described in "Creating a Locally Managed Temporary Tablespace" on page 14-12

Oracle Database 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*. This file number can be used in many SQL statements that reference datafiles in place of using the file name. The absolute file number can be found in the `FILE#` column of the `V$DATAFILE` or `V$TEMPFILE` view, or in the `FILE_ID` column of the `DBA_DATA_FILES` or `DBA_TEMP_FILES` view.
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. In a bigfile tablespace, the relative file number is always 1024 (4096 on OS/390 platform).

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` and `SYSAUX` tablespaces of a database. Your database should contain several other tablespaces with their associated datafiles or tempfiles. The number of datafiles that you anticipate creating for your database can affect the settings of initialization parameters and the specification of `CREATE DATABASE` statement clauses.

Be aware that your operating system might impose limits on the number of datafiles contained in your Oracle Database. Also consider that the number of datafiles, and how and where they are allocated can affect the performance of your database.

> **Note:** One means of controlling the number of datafiles in your database and simplifying their management is to use bigfile tablespaces. Bigfile tablespaces comprise a single, very large datafile and are especially useful in ultra large databases and where a logical volume manager is used for managing operating system files. Bigfile tablespaces are discussed in "Bigfile Tablespaces" on page 14-6.

Consider the following guidelines when determining the number of datafiles for your database.

Determine a Value for the DB_FILES Initialization Parameter

When starting an Oracle Database 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.

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.

Consider Possible Limitations When Adding Datafiles to a Tablespace

You can add datafiles to traditional smallfile 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.
- The database 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 contained in a tablespace, and ultimately the database, can have an impact upon performance.

Oracle Database allows more datafiles in the database than the operating system defined limit. The database DBW*n* processes can open all online datafiles. Oracle Database 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
- *Oracle Database SQL Language Reference* for more information about the MAXDATAFILES parameter of the CREATE DATABASE or CREATE CONTROLFILE statement

Determine the Size of Datafiles

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 to ensure that data is spread evenly across all devices.

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.

> **See Also:** *Oracle Database Performance Tuning Guide* for information about I/O and the placement of datafiles

Store Datafiles Separate from Redo Log Files

Datafiles should not be stored on the same disk drive that stores the database 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

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 specifically described:

SQL Statement	Description	Additional Information
`CREATE TABLESPACE`	Creates a tablespace and the datafiles that comprise it	"Creating Tablespaces" on page 14-2
`CREATE TEMPORARY TABLESPACE`	Creates a locally-managed temporary tablespace and the *tempfiles* (tempfiles are a special kind of datafile) that comprise it	"Creating a Locally Managed Temporary Tablespace" on page 14-12
`ALTER TABLESPACE ... ADD DATAFILE`	Creates and adds a datafile to a tablespace	"Altering a Locally Managed Temporary Tablespace" on page 14-22
`ALTER TABLESPACE ... ADD TEMPFILE`	Creates and adds a tempfile to a temporary tablespace	"Creating a Locally Managed Temporary Tablespace" on page 14-12
`CREATE DATABASE`	Creates a database and associated datafiles	"Creating a Database with the CREATE DATABASE Statement" on page 2-6
`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.	See *Oracle Database Backup and Recovery User's Guide*.

If you add new datafiles to a tablespace and do not fully specify the filenames, the database 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, the database 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 Datafile 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 file size increases 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 or be suspended 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`
- `ALTER 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. For a bigfile tablespace, you are able to perform these operations using the `ALTER TABLESPACE` 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:** *Oracle Database SQL Language 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. Therefore, 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.

For a bigfile tablespace you can use the ALTER TABLESPACE statement to resize a datafile. You are not allowed to add a datafile to a bigfile tablespace.

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. It could be that the file contains data beyond the specified decreased size, in which case the database will return an error.

Altering Datafile Availability

You can alter the availability of individual datafiles or tempfiles by taking them offline or bringing them online. Offline datafiles are unavailable to the database and cannot be accessed until they are brought back online.

Reasons for altering datafile availability include the following:

- You want to perform an offline backup of a datafile.
- You want to rename or relocate a datafile. You must first take it offline or take the tablespace offline.
- The database 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.
- A datafile becomes missing or corrupted. You must take it offline before you can open the database.

The datafiles of a read-only tablespace can be taken offline or brought online, but bringing a file online does not affect the read-only status of the tablespace. You cannot write to the datafile until the tablespace is returned to the read/write state.

> **Note:** You can make all datafiles of a tablespace temporarily unavailable by taking the tablespace itself offline. You *must* leave these files in the tablespace to bring the tablespace back online, although you can relocate or rename them following procedures similar to those shown in "Renaming and Relocating Datafiles" on page 15-8.
>
> For more information, see "Taking Tablespaces Offline" on page 14-15.

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

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 FOR DROP` clauses.

- The `OFFLINE` keyword causes the database to mark the datafile `OFFLINE`, whether or not it is corrupted, so that you can open the database.
- The `FOR DROP` keywords mark the datafile for subsequent dropping. Such a datafile can no longer be brought back online.

Note: This operation does not actually drop the datafile. It remains in the data dictionary, and you must drop it yourself using one of the following methods:

- An `ALTER TABLESPACE ... DROP DATAFILE` statement.

  After an `OFFLINE FOR DROP`, this method works for dictionary managed tablespaces only.
- A `DROP TABLESPACE ... INCLUDING CONTENTS AND DATAFILES` statement
- If the preceding methods fail, an operating system command to delete the datafile. This is the least desirable method, as it leaves references to the datafile in the data dictionary and control files.

The following statement takes the specified datafile offline and marks it to be dropped:

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE FOR 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 preceding `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 tablespace 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 possible procedures for doing this are described in the following sections:

- Procedures for Renaming and Relocating Datafiles in a Single Tablespace
- Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces

When you rename and relocate datafiles with these procedures, only the pointers to the datafiles, as recorded in the database 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.

Procedures for Renaming and Relocating Datafiles in a Single Tablespace

The section suggests some procedures for renaming and relocating datafiles that can be used for a single tablespace. You must have `ALTER TABLESPACE` system privileges.

> **See Also:** "Taking Tablespaces Offline" on page 14-15 for more information about taking tablespaces offline in preparation for renaming or relocating datafiles

Procedure for Renaming Datafiles in a Single Tablespace

To rename datafiles in a single tablespace, complete the following steps:

1. Take the tablespace that contains the datafiles offline. The database must be open.

   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';
   ```

   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.

Procedure for Relocating Datafiles in a Single Tablespace

Here is a sample procedure 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. If you do not know the specific file names or sizes, you can obtain this information by issuing the following query of the data dictionary view DBA_DATA_FILES:

```
SQL> SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
  2> 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:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

3. Copy the datafiles to their new locations and rename them using the operating system. You can copy the files using the DBMS_FILE_TRANSFER package discussed in "Copying Files Using the Database Server" on page 15-12.

> **Note:** You can temporarily exit SQL*Plus to execute an operating system command to copy a file by using the SQL*Plus HOST command.

4. Rename the datafiles within the database.

   The datafile pointers for the files that comprise the users tablespace, recorded in the control file of the associated database, must now be changed from the old names to the new names.

   Use the ALTER TABLESPACE...RENAME DATAFILE 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. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Procedure for Renaming and Relocating Datafiles in Multiple Tablespaces

You can rename and relocate datafiles in one or more tablespaces using the ALTER DATABASE RENAME FILE statement. This method is the only choice if you want to rename or relocate datafiles of several tablespaces in one operation. You must have the ALTER DATABASE system privilege.

> **Note:** To rename or relocate datafiles of the SYSTEM tablespace, the default temporary tablespace, or the active undo tablespace you must use this ALTER DATABASE method because you cannot take these tablespaces offline.

To rename datafiles in multiple tablespaces, follow these steps.

1. Ensure that the database is mounted but closed.

> **Note:** Optionally, the database does not have to be closed, but the datafiles (or tempfiles) must be offline.

2. Copy the datafiles to be renamed to their new locations and new names, using the operating system. You can copy the files using the `DBMS_FILE_TRANSFER` package discussed in "Copying Files Using the Database Server" on page 15-12.
3. Use `ALTER DATABASE` to rename the file pointers in the database 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;
   ```

   Always provide complete filenames (including their paths) to properly identify the old and new datafiles. In particular, specify the old datafile names exactly as they appear in the `DBA_DATA_FILES` view.
4. Back up the database. After making any structural changes to a database, always perform an immediate and complete backup.

Dropping Datafiles

You use the `DROP DATAFILE` and `DROP TEMPFILE` clauses of the `ALTER TABLESPACE` command to drop a single datafile or tempfile. The datafile must be empty. (A datafile is considered to be empty when no extents remain allocated from it.) When you drop a datafile or tempfile, references to the datafile or tempfile are removed from the data dictionary and control files, and the physical file is deleted from the file system or Oracle Automatic Storage Management (Oracle ASM) disk group.

The following example drops the datafile identified by the alias `example_df3.f` in the Oracle ASM disk group `DGROUP1`. The datafile belongs to the `example` tablespace.

```
ALTER TABLESPACE example DROP DATAFILE '+DGROUP1/example_df3.f';
```

The next example drops the tempfile `lmtemp02.dbf`, which belongs to the `lmtemp` tablespace.

```
ALTER TABLESPACE lmtemp DROP TEMPFILE '/u02/oracle/data/lmtemp02.dbf';
```

This is equivalent to the following statement:

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtemp02.dbf' DROP
     INCLUDING DATAFILES;
```

See *Oracle Database SQL Language Reference* for `ALTER TABLESPACE` syntax details.

Restrictions for Dropping Datafiles

The following are restrictions for dropping datafiles and tempfiles:

- The database must be open.
- If a datafile is not empty, it cannot be dropped.

If you must remove a datafile that is not empty and that cannot be made empty by dropping schema objects, you must drop the tablespace that contains the datafile.

- You cannot drop the first or only datafile in a tablespace.

  Therefore, `DROP DATAFILE` cannot be used with a bigfile tablespace.
- You cannot drop datafiles in a read-only tablespace that was migrated from dictionary managed to locally managed.
- You cannot drop datafiles in the `SYSTEM` tablespace.
- If a datafile in a locally managed tablespace is offline, it cannot be dropped.

See Also: Dropping Tablespaces on page 14-24

Verifying Data Blocks in Datafiles

To configure the database to use checksums to verify data blocks, set the initialization parameter `DB_BLOCK_CHECKSUM` to `TYPICAL` (the default). This causes the DBW*n* process and the direct loader to calculate a checksum for each block and to store the checksum in the block header when writing the block to disk.

The checksum is verified when the block is read, but only if `DB_BLOCK_CHECKSUM` is `TRUE` and the last write of the block stored a checksum. If corruption is detected, the database returns message `ORA-01578` and writes information about the corruption to the alert log.

The value of the `DB_BLOCK_CHECKSUM` parameter can be changed dynamically using the `ALTER SYSTEM` statement. Regardless of the setting of this parameter, checksums are always used to verify data blocks in the `SYSTEM` tablespace.

See Also: *Oracle Database Reference* for more information about the `DB_BLOCK_CHECKSUM` initialization parameter

Copying Files Using the Database Server

You do not necessarily have to use the operating system to copy a file within a database, or transfer a file between databases as you would do when using the transportable tablespace feature. You can use the `DBMS_FILE_TRANSFER` package, or you can use Streams propagation. Using Streams is not discussed in this book, but an example of using the `DBMS_FILE_TRANSFER` package is shown in "Copying a File on a Local File System" on page 15-13.

The `DBMS_FILE_TRANSFER` package can use a local file system or an Oracle Automatic Storage Management (Oracle ASM) disk group as the source or destination for a file transfer. Only Oracle database files (datafiles, tempfiles, controlfiles, and so on) can be involved in transfers to and from Oracle ASM.

Caution: Do not use the `DBMS_FILE_TRANSFER` package to copy or transfer a file that is being modified by a database because doing so may result in an inconsistent file.

On UNIX systems, the owner of a file created by the `DBMS_FILE_TRANSFER` package is the owner of the shadow process running the instance. Normally, this owner is `ORACLE`. A file created using `DBMS_FILE_TRANSFER` is always writable and readable by all processes in the database, but non privileged users who need to read or write such a file directly may need access from a system administrator.

This section contains the following topics:

- Copying a File on a Local File System
- Third-Party File Transfer
- File Transfer and the DBMS_SCHEDULER Package
- Advanced File Transfer Mechanisms

See Also:

- *Oracle Streams Concepts and Administration*
- "Transporting Tablespaces Between Databases" on page 14-30
- *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_FILE_TRANSFER package.

Copying a File on a Local File System

This section includes an example that uses the COPY_FILE procedure in the DBMS_FILE_TRANSFER package to copy a file on a local file system. The following example copies a binary file named db1.dat from the /usr/admin/source directory to the /usr/admin/destination directory as db1_copy.dat on a local file system:

1. In SQL*Plus, connect as an administrative user who can grant privileges and create directory objects using SQL.
2. Use the SQL command CREATE DIRECTORY to create a directory object for the directory from which you want to copy the file. A directory object is similar to an alias for the directory. For example, to create a directory object called SOURCE_DIR for the /usr/admin/source directory on your computer system, execute the following statement:

   ```
   CREATE DIRECTORY SOURCE_DIR AS '/usr/admin/source';
   ```

3. Use the SQL command CREATE DIRECTORY to create a directory object for the directory into which you want to copy the binary file. For example, to create a directory object called DEST_DIR for the /usr/admin/destination directory on your computer system, execute the following statement:

   ```
   CREATE DIRECTORY DEST_DIR AS '/usr/admin/destination';
   ```

4. Grant the required privileges to the user who will run the COPY_FILE procedure. In this example, the strmadmin user runs the procedure.

   ```
   GRANT EXECUTE ON DBMS_FILE_TRANSFER TO strmadmin;

   GRANT READ ON DIRECTORY source_dir TO strmadmin;

   GRANT WRITE ON DIRECTORY dest_dir TO strmadmin;
   ```

5. Connect as strmadmin user and provide the user password when prompted:

   ```
   CONNECT strmadmin
   ```

6. Run the COPY_FILE procedure to copy the file:

   ```
   BEGIN
     DBMS_FILE_TRANSFER.COPY_FILE(
           source_directory_object       =>  'SOURCE_DIR',
           source_file_name              =>  'db1.dat',
           destination_directory_object  =>  'DEST_DIR',
   ```

```
        destination_file_name         =>  'db1_copy.dat');
END;
/
```

Caution: Do not use the DBMS_FILE_TRANSFER package to copy or transfer a file that is being modified by a database because doing so may result in an inconsistent file.

Third-Party File Transfer

Although the procedures in the DBMS_FILE_TRANSFER package typically are invoked as local procedure calls, they can also be invoked as remote procedure calls. A remote procedure call lets you copy a file within a database even when you are connected to a different database. For example, you can make a copy of a file on database DB, even if you are connected to another database, by executing the following remote procedure call:

```
DBMS_FILE_TRANSFER.COPY_FILE@DB(...)
```

Using remote procedure calls enables you to copy a file between two databases, even if you are not connected to either database. For example, you can connect to database A and then transfer a file from database B to database C. In this example, database A is the third party because it is neither the source of nor the destination for the transferred file.

A third-party file transfer can both push and pull a file. Continuing with the previous example, you can perform a third-party file transfer if you have a database link from A to either B or C, and that database has a database link to the other database. Database A does not need a database link to both B and C.

For example, if you have a database link from A to B, and another database link from B to C, then you can run the following procedure at A to transfer a file from B to C:

```
DBMS_FILE_TRANSFER.PUT_FILE@B(...)
```

This configuration pushes the file.

Alternatively, if you have a database link from A to C, and another database link from C to B, then you can run the following procedure at database A to transfer a file from B to C:

```
DBMS_FILE_TRANSFER.GET_FILE@C(...)
```

This configuration pulls the file.

File Transfer and the DBMS_SCHEDULER Package

You can use the DBMS_SCHEDULER package to transfer files automatically within a single database and between databases. Third-party file transfers are also supported by the DBMS_SCHEDULER package. You can monitor a long-running file transfer done by the Scheduler using the V$SESSION_LONGOPS dynamic performance view at the databases reading or writing the file. Any database links used by a Scheduler job must be fixed user database links.

You can use a restartable Scheduler job to improve the reliability of file transfers automatically, especially if there are intermittent failures. If a file transfer fails before the destination file is closed, then you can restart the file transfer from the beginning once the database has removed any partially written destination file. Hence you should consider using a restartable Scheduler job to transfer a file if the rest of the job

is restartable. See Chapter 29, "Scheduling Jobs with Oracle Scheduler" for more information on Scheduler jobs.

> **Note:** If a single restartable job transfers several files, then you should consider restart scenarios in which some of the files have been transferred already and some have not been transferred yet.

Advanced File Transfer Mechanisms

You can create more sophisticated file transfer mechanisms using both the `DBMS_FILE_TRANSFER` package and the `DBMS_SCHEDULER` package. For example, when several databases have a copy of the file you want to transfer, you can consider factors such as source availability, source load, and communication bandwidth to the destination database when deciding which source database to contact first and which source databases to try if failures occur. In this case, the information about these factors must be available to you, and you must create the mechanism that considers these factors.

As another example, when early completion time is more important than load, you can submit several Scheduler jobs to transfer files in parallel. As a final example, knowing something about file layout on the source and destination databases enables you to minimize disk contention by performing or scheduling simultaneous transfers only if they use different I/O devices.

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 Database 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 Inexpensive 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 the Oracle Database approach to resolving this problem.

The following topics are contained in this section:

- Overview of Oracle Database File Mapping Interface
- How the Oracle Database File Mapping Interface Works
- Using the Oracle Database File Mapping Interface
- File Mapping Examples

> **Note:** This section presents an overview of the Oracle Database 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. It provides an easy to use graphical interface for mapping files to physical devices.

Overview of Oracle Database 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 Database provides a mechanism to show a complete mapping of a file to intermediate layers of logical volumes to actual physical devices. This is accomplished though 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. The database communicates with these libraries through an external non-Oracle Database process that is spawned by a 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.

Note: The file mapping interface is not available on Windows platforms.

How the Oracle Database File Mapping Interface Works

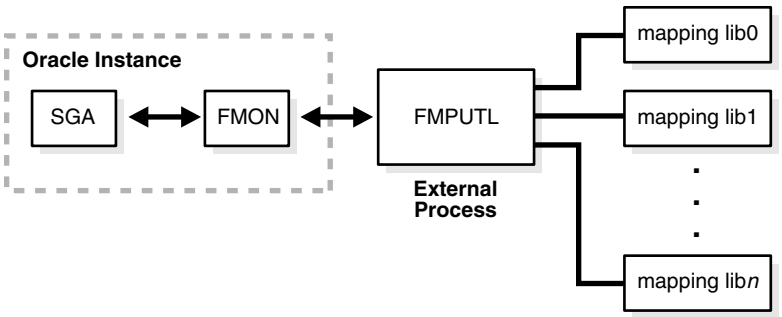
This section describes the components of the Oracle Database file mapping interface and how the interface works. It contains the following topics:

- 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 15–1 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 the database 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 15-18.
- Refreshing mapping information when a change occurs because of:
  - Changes to 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 Database 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 Database 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 the Oracle Database 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 different from Oracle Database extents. File system extents are physical contiguous blocks of data written to a device as managed by the file system. Oracle Database extents are logical structures managed by the database, 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 datafiles 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.

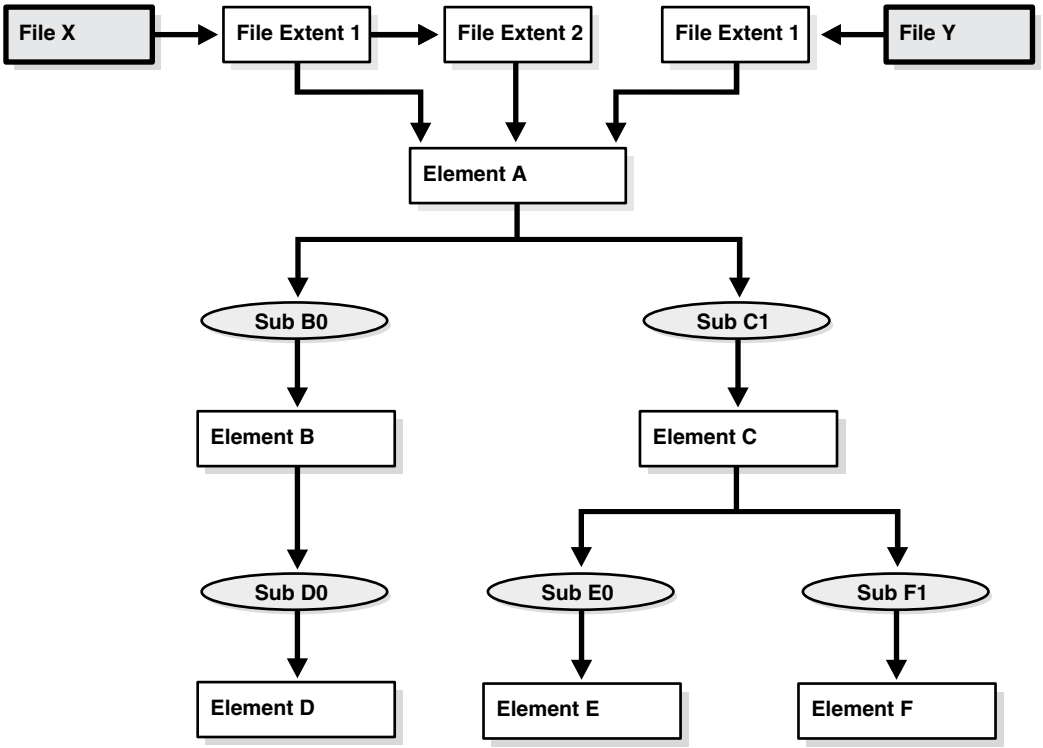
The two extents of File X and the one extent of File Y both map to Element A. Element A is striped to Elements B and C. Element A maps to Elements B and C by way of Subelements B0 and C1, respectively.

Element B is a partition of Element D (a physical disk), and is mapped to Element D by way of subelement D0.

Element C is mirrored over Elements E and F (both physical disks), and is mirrored to those physical disks by way of Subelements E0 and F1, respectively.

All of the mapping structures are illustrated in Figure 15–2.

Figure 15–2 Illustration of Mapping Structures



Note that the mapping structures represented are sufficient to describe the entire mapping information for the Oracle Database 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 the database 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. The database is only capable of refreshing the mapping information while the instance is up.

Using the Oracle Database File Mapping Interface

This section discusses how to use the Oracle Database 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 /opt/ORCLfmap/prot1_32/etc directory for 32-bit platforms, or in the /opt/ORCLfmap/prot1_64/etc directory for 64-bit platforms.

   > **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 the database 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 must be included in filemap.ora 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 must 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`.

   The instance does not have to be shut down to set this parameter. You can set it using the following `ALTER SYSTEM` statement:

   ```
   ALTER SYSTEM SET FILE_MAPPING=TRUE;
   ```

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 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 to control the mapping operations. The various procedures available to you are described in the following table.

Procedure	Use to:
MAP_OBJECT	Build the mapping information for the database object identified by object name, owner, and type
MAP_ELEMENT	Build mapping information for the specified element
MAP_FILE	Build mapping information for the specified filename
MAP_ALL	Build entire mapping information for all types of database 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:

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_STORAGE_MAP package
- "File Mapping Examples" on page 15-22 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
V$MAP_EXT_ELEMENT	Contains supplementary information for all element mapping

View	Description
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: *Oracle Database Reference* for a complete description 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	Data Type	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 FILE_MAP_IDX in V$MAP_FILE)
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 (half KB) 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.
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)	Data type (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 the Oracle Database file mapping feature. This includes:

- The ability to map all the database files that span a particular device

- The ability to map a particular file into its corresponding devices
- The ability to map a particular database object, including its block distribution at all levels within the I/O stack

Consider an Oracle Database 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-vol1`, 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 Database Files that Span a Device

The following query returns all Oracle Database 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-vol1_-1_-1
/oracle/dbs/t_db1.f              _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_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_db1.f                          _sym_hyper_000183600407_00C_1
/oracle/dbs/t_db1.f              _sym_subdisk_/dev/vx/rdsk/ipfdg/ipf-vol1_0_1_0
/oracle/dbs/t_db1.f                  /dev/vx/rdmp/c2t1d1s2
/oracle/dbs/t_db1.f                      _sym_symdev_000183600407_00D
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00D_0
/oracle/dbs/t_db1.f                          _sym_hyper_000183600407_00D_1
```

Example 3: Map a Database 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_db1.f  /dev/vx/dsk/ipfdg/ipf-vol1          0      20
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_plex_/dev/vx/rdsk/ipf          1      20
                                            pdg/if-vol1_-1_-1
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/          2      12
                                            ipfdg/ipf-vol1_0_1_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  _sym_subdisk_/dev/vx/rdsk/ipf       2       8
                                            dg/ipf-vol1_0_2_0
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d1s2               3      12
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.f  /dev/vx/rdmp/c2t1d2s2               3       8
BONUS  SCOTT   TABLE   /oracle/dbs/t_db1.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
```

Datafiles Data Dictionary Views

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 the most recent checkpoint of a datafile.

> **See Also:** *Oracle Database Reference* for complete descriptions of these views

16

Managing Undo

Beginning with Release 11*g*, for a default installation, Oracle Database automatically manages undo. There is typically no need for DBA intervention. However, if your installation uses Oracle Flashback operations, you may need to perform some undo management tasks to ensure the success of these operations.

In this chapter:

- What Is Undo?
- Introduction to Automatic Undo Management
- Setting the Minimum Undo Retention Period
- Sizing a Fixed-Size Undo Tablespace
- Managing Undo Tablespaces
- Migrating to Automatic Undo Management
- Undo Space Data Dictionary Views

> **See Also:** Chapter 17, "Using Oracle Managed Files"for information about creating an undo tablespace whose datafiles are both created and managed by Oracle Database.

What Is Undo?

Oracle Database creates and manages 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. These records are collectively referred to as **undo**.

Undo records are used to:

- Roll back transactions when a `ROLLBACK` statement is issued
- Recover the database
- Provide read consistency
- Analyze data as of an earlier point in time by using Oracle Flashback Query
- Recover from logical corruptions using Oracle Flashback features

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.

Introduction to Automatic Undo Management

This section introduces the concepts of Automatic Undo Management and discusses the following topics:

- Overview of Automatic Undo Management
- About the Undo Retention Period

Overview of Automatic Undo Management

Oracle provides a fully automated mechanism, referred to as automatic undo management, for managing undo information and space. With automatic undo management, the database manages undo segments in an undo tablespace. Beginning with Release 11*g*, automatic undo management is the default mode for a newly installed database. An auto-extending undo tablespace named `UNDOTBS1` is automatically created when you create the database with Database Configuration Assistant (DBCA).

An undo tablespace can also be created explicitly. The methods of creating an undo tablespace are explained in "Creating an Undo Tablespace" on page 16-8.

When the instance starts, the database automatically selects the first available undo tablespace. If no undo tablespace is available, the instance starts without an undo tablespace and stores undo records in the `SYSTEM` tablespace. This is not recommended, and an alert message is written to the alert log file to warn that the system is running without an undo tablespace.

If the database contains multiple undo tablespaces, you can optionally specify at startup that you want to use a specific undo tablespace. This is done by setting the `UNDO_TABLESPACE` initialization parameter, as shown in this example:

```
UNDO_TABLESPACE = undotbs_01
```

If the tablespace specified in the initialization parameter does not exist, the `STARTUP` command fails. The `UNDO_TABLESPACE` parameter can be used to assign a specific undo tablespace to an instance in an Oracle Real Application Clusters environment.

The database can also run in *manual undo management mode*. In this mode, undo space is managed through rollback segments, and no undo tablespace is used.

> **Note:** Space management for rollback segments is complex. Oracle strongly recommends leaving the database in automatic undo management mode.

The following is a summary of the initialization parameters for undo management:

Initialization Parameter	Description
`UNDO_MANAGEMENT`	If `AUTO` or null, enables automatic undo management. If `MANUAL`, sets manual undo management mode. The default is `AUTO`.

Initialization Parameter	Description
UNDO_TABLESPACE	Optional, and valid only in automatic undo management mode. Specifies the name of an undo tablespace. Use only when the database has multiple undo tablespaces and you want to direct the database instance to use a particular undo tablespace.

When automatic undo management is enabled, if the initialization parameter file contains parameters relating to manual undo management, they are ignored.

Note: Earlier releases of Oracle Database default to manual undo management mode. To change to automatic undo management, you must first create an undo tablespace and then change the UNDO_MANAGEMENT initialization parameter to AUTO. If your Oracle Database is release 9*i* or later and you want to change to automatic undo management, see *Oracle Database Upgrade Guide* for instructions.

A null UNDO_MANAGEMENT initialization parameter defaults to automatic undo management mode in Release 11*g* and later, but defaults to manual undo management mode in earlier releases. You must therefore use caution when upgrading a previous release to Release 11g. *Oracle Database Upgrade Guide* describes the correct method of migrating to automatic undo management mode, including information on how to size the undo tablespace.

See Also: *Oracle Database Reference* for complete descriptions of initialization parameters used in undo management

About the Undo Retention Period

After a transaction is committed, undo data is no longer needed for rollback or transaction recovery purposes. However, for consistent read purposes, long-running queries may require this old undo information for producing older images of data blocks. Furthermore, the success of several Oracle Flashback features can also depend upon the availability of older undo information. For these reasons, it is desirable to retain the old undo information for as long as possible.

When automatic undo management is enabled, there is always a current **undo retention period**, which is the minimum amount of time that Oracle Database attempts to retain old undo information before overwriting it. Old (committed) undo information that is older than the current undo retention period is said to be *expired* and its space is available to be overwritten by new transactions. Old undo information with an age that is less than the current undo retention period is said to be *unexpired* and is retained for consistent read and Oracle Flashback operations.

Oracle Database automatically tunes the undo retention period based on undo tablespace size and system activity. You can optionally specify a minimum undo retention period (in seconds) by setting the UNDO_RETENTION initialization parameter. The exact impact this parameter on undo retention is as follows:

- The UNDO_RETENTION parameter is ignored for a fixed size undo tablespace. The database always tunes the undo retention period for the best possible retention, based on system activity and undo tablespace size. See "Automatic Tuning of Undo Retention" on page 16-4 for more information.

- For an undo tablespace with the `AUTOEXTEND` option enabled, the database attempts to honor the minimum retention period specified by `UNDO_RETENTION`. When space is low, instead of overwriting unexpired undo information, the tablespace auto-extends. If the `MAXSIZE` clause is specified for an auto-extending undo tablespace, when the maximum size is reached, the database may begin to overwrite unexpired undo information. The `UNDOTBS1` tablespace that is automatically created by DBCA is auto-extending.

Automatic Tuning of Undo Retention

Oracle Database automatically tunes the undo retention period based on how the undo tablespace is configured.

- If the undo tablespace is configured with the `AUTOEXTEND` option, the database dynamically tunes the undo retention period to be somewhat longer than the longest-running active query on the system. However, this retention period may be insufficient to accommodate Oracle Flashback operations. Oracle Flashback operations resulting in `snapshot too old` errors are the indicator that you must intervene to ensure that sufficient undo data is retained to support these operations. To better accommodate Oracle Flashback features, you can either set the `UNDO_RETENTION` parameter to a value equal to the longest expected Oracle Flashback operation, or you can change the undo tablespace to fixed size.
- If the undo tablespace is fixed size, the database dynamically tunes the undo retention period for the best possible retention for that tablespace size and the current system load. This best possible retention time is typically significantly greater than the duration of the longest-running active query.

  If you decide to change the undo tablespace to fixed-size, you must choose a tablespace size that is sufficiently large. If you choose an undo tablespace size that is too small, the following two errors could occur:

  - DML could fail because there is not enough space to accommodate undo for new transactions.
  - Long-running queries could fail with a `snapshot too old` error, which means that there was insufficient undo data for read consistency.

  See "Sizing a Fixed-Size Undo Tablespace" on page 16-6 for more information.

Note: Automatic tuning of undo retention is not supported for LOBs. This is because undo information for LOBs is stored in the segment itself and not in the undo tablespace. For LOBs, the database attempts to honor the minimum undo retention period specified by `UNDO_RETENTION`. However, if space becomes low, unexpired LOB undo information may be overwritten.

See Also: "Setting the Minimum Undo Retention Period" on page 16-5

Retention Guarantee

To guarantee the success of long-running queries or Oracle Flashback operations, you can enable retention guarantee. If retention guarantee is enabled, the specified minimum undo retention is guaranteed; the database never overwrites unexpired undo data even if it means that transactions fail due to lack of space in the undo tablespace. If retention guarantee is not enabled, the database can overwrite unexpired

undo when space is low, thus lowering the undo retention for the system. This option is disabled by default.

> **WARNING: Enabling retention guarantee can cause multiple DML operations to fail. Use with caution.**

You enable retention guarantee by specifying the `RETENTION GUARANTEE` clause for the undo tablespace when you create it with either the `CREATE DATABASE` or `CREATE UNDO TABLESPACE` statement. Or, you can later specify this clause in an `ALTER TABLESPACE` statement. You disable retention guarantee with the `RETENTION NOGUARANTEE` clause.

You can use the `DBA_TABLESPACES` view to determine the retention guarantee setting for the undo tablespace. A column named `RETENTION` contains a value of `GUARANTEE`, `NOGUARANTEE`, or `NOT APPLY`, where `NOT APPLY` is used for tablespaces other than the undo tablespace.

Undo Retention Tuning and Alert Thresholds

For a fixed-size undo tablespace, the database calculates the best possible retention based on database statistics and on the size of the undo tablespace. For optimal undo management, rather than tuning based on 100% of the tablespace size, the database tunes the undo retention period based on 85% of the tablespace size, or on the warning alert threshold percentage for space used, whichever is lower. (The warning alert threshold defaults to 85%, but can be changed.) Therefore, if you set the warning alert threshold of the undo tablespace below 85%, this may reduce the tuned size of the undo retention period. For more information on tablespace alert thresholds, see "Managing Tablespace Alerts" on page 19-1.

Tracking the Tuned Undo Retention Period

You can determine the current retention period by querying the `TUNED_UNDORETENTION` column of the `V$UNDOSTAT` view. This view contains one row for each 10-minute statistics collection interval over the last 4 days. (Beyond 4 days, the data is available in the `DBA_HIST_UNDOSTAT` view.) `TUNED_UNDORETENTION` is given in seconds.

```
select to_char(begin_time, 'DD-MON-RR HH24:MI') begin_time,
to_char(end_time, 'DD-MON-RR HH24:MI') end_time, tuned_undoretention
from v$undostat order by end_time;

BEGIN_TIME      END_TIME        TUNED_UNDORETENTION
--------------- --------------- -------------------
04-FEB-05 00:01 04-FEB-05 00:11               12100
      ...
07-FEB-05 23:21 07-FEB-05 23:31               86700
07-FEB-05 23:31 07-FEB-05 23:41               86700
07-FEB-05 23:41 07-FEB-05 23:51               86700
07-FEB-05 23:51 07-FEB-05 23:52               86700

576 rows selected.
```

See *Oracle Database Reference* for more information about `V$UNDOSTAT`.

Setting the Minimum Undo Retention Period

You specify the minimum undo retention period (in seconds) by setting the `UNDO_RETENTION` initialization parameter. As described in "About the Undo Retention Period" on page 16-3, the current undo retention period may be automatically tuned to

be greater than UNDO_RETENTION, or, unless retention guarantee is enabled, less than UNDO_RETENTION if space in the undo tablespace is low.

To set the minimum undo retention period:

- Do one of the following:
  - Set UNDO_RETENTION in the initialization parameter file.

    ```
    UNDO_RETENTION = 1800
    ```

  - Change UNDO_RETENTION at any time using the ALTER SYSTEM statement:

    ```
    ALTER SYSTEM SET UNDO_RETENTION = 2400;
    ```

The effect of an UNDO_RETENTION parameter change is immediate, but it can only be honored if the current undo tablespace has enough space.

Sizing a Fixed-Size Undo Tablespace

Automatic tuning of undo retention typically achieves better results with a fixed-size undo tablespace. If you decide to use a fixed-size undo tablespace, the Undo Advisor can help you estimate needed capacity. You can access the Undo Advisor through Enterprise Manager or through the DBMS_ADVISOR PL/SQL package. Enterprise Manager is the preferred method of accessing the advisor. For more information on using the Undo Advisor through Enterprise Manager, see *Oracle Database 2 Day DBA*.

The Undo Advisor relies for its analysis on data collected in the Automatic Workload Repository (AWR). It is therefore important that the AWR have adequate workload statistics available so that the Undo Advisor can make accurate recommendations. For newly created databases, adequate statistics may not be available immediately. In such cases, continue to use the default auto-extending undo tablespace until at least one workload cycle completes.

An adjustment to the collection interval and retention period for AWR statistics can affect the precision and the type of recommendations that the advisor produces. See *Oracle Database Performance Tuning Guide* for more information.

To use the Undo Advisor, you first estimate these two values:

- The length of your expected longest running query

  After the database has completed a workload cycle, you can view the Longest Running Query field on the System Activity subpage of the Automatic Undo Management page.

- The longest interval that you will require for Oracle Flashback operations

  For example, if you expect to run Oracle Flashback queries for up to 48 hours in the past, your Oracle Flashback requirement is 48 hours.

You then take the maximum of these two values and use that value as input to the Undo Advisor.

Running the Undo Advisor does not alter the size of the undo tablespace. The advisor just returns a recommendation. You must use ALTER DATABASE statements to change the tablespace datafiles to fixed sizes.

The following example assumes that the undo tablespace has one auto-extending datafile named undotbs.dbf. The example changes the tablespace to a fixed size of 300MB.

```
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' RESIZE 300M;
```

```
ALTER DATABASE DATAFILE '/oracle/dbs/undotbs.dbf' AUTOEXTEND OFF;
```

Note: To make the undo tablespace fixed-size, Oracle suggests that you first allow enough time after database creation to run a full workload, thus allowing the undo tablespace to grow to its minimum required size to handle the workload. Then, you can use the Undo Advisor to determine, if desired, how much larger to set the size of the undo tablespace to allow for long-running queries and Oracle Flashback operations.

See Also: *Oracle Database 2 Day DBA* for instructions for computing the minimum undo tablespace size with the Undo Advisor

The Undo Advisor PL/SQL Interface

You can activate the Undo Advisor by creating an undo advisor task through the advisor framework. The following example creates an undo advisor task to evaluate the undo tablespace. The name of the advisor is 'Undo Advisor'. The analysis is based on Automatic Workload Repository snapshots, which you must specify by setting parameters `START_SNAPSHOT` and `END_SNAPSHOT`. In the following example, the `START_SNAPSHOT` is "1" and `END_SNAPSHOT` is "2".

```
DECLARE
   tid    NUMBER;
   tname  VARCHAR2(30);
   oid    NUMBER;
BEGIN
   DBMS_ADVISOR.CREATE_TASK('Undo Advisor', tid, tname, 'Undo Advisor Task');
   DBMS_ADVISOR.CREATE_OBJECT(tname, 'UNDO_TBS', null, null, null, 'null', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'TARGET_OBJECTS', oid);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'START_SNAPSHOT', 1);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'END_SNAPSHOT', 2);
   DBMS_ADVISOR.SET_TASK_PARAMETER(tname, 'INSTANCE', 1);
   DBMS_ADVISOR.execute_task(tname);
END;
/
```

After you have created the advisor task, you can view the output and recommendations in the Automatic Database Diagnostic Monitor in Enterprise Manager. This information is also available in the `DBA_ADVISOR_*` data dictionary views (`DBA_ADVISOR_TASKS`, `DBA_ADVISOR_OBJECTS`, `DBA_ADVISOR_FINDINGS`, `DBA_ADVISOR_RECOMMENDATIONS`, and so on).

See Also:

- "Using the Segment Advisor" on page 19-12 for an example of creating an advisor task for a different advisor
- *Oracle Database Reference* for information about the `DBA_ADVISOR_*` data dictionary views

Managing Undo Tablespaces

This section describes the various steps involved in undo tablespace management and contains the following sections:

- Creating an Undo Tablespace

- Altering an Undo Tablespace
- Dropping an Undo Tablespace
- Switching Undo Tablespaces
- Establishing User Quotas for Undo Space
- Undo Space Data Dictionary Views

Creating an Undo Tablespace

Although Database Configuration Assistant (DBCA) automatically creates an undo tablespace for new installations of Oracle Database Release 11*g*, there may be occasions when you want to manually create 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.

Oracle Database enables you to create a single-file undo tablespace. Single-file, or bigfile, tablespaces are discussed in "Bigfile Tablespaces" on page 14-6.

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.

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.

The `CREATE DATABASE` statement also lets you create a single-file undo tablespace at database creation. This is discussed in "Supporting Bigfile Tablespaces During Database Creation" on page 2-20.

> **See Also:** *Oracle Database SQL Language Reference* for the syntax for using the `CREATE DATABASE` statement to create an undo tablespace

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. The database determines most of the attributes of the undo tablespace, but you can specify the `DATAFILE` clause.

This example creates the undotbs_02 undo tablespace with the AUTOEXTEND option:

```
CREATE UNDO TABLESPACE undotbs_02
     DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

You can create multiple undo tablespaces, but only one of them can be active at any one time.

> **See Also:** *Oracle Database SQL Language Reference* for the syntax for using the CREATE UNDO TABLESPACE statement to create an undo tablespace

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
- Enabling and disabling undo retention guarantee

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 Datafile Size" on page 15-5
> - *Oracle Database SQL Language Reference* for ALTER TABLESPACE syntax

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.

See Also: *Oracle Database SQL Language Reference* for DROP TABLESPACE syntax

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 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 (in an Oracle RAC environment only)

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), then the current undo tablespace is switched out and the next available undo tablespace is switched in. Use this statement with care because there may be no undo tablespace available.

The following example unassigns the current undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

Establishing User Quotas for Undo Space

The Oracle 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 undo 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, "Managing Resources with Oracle Database Resource Manager"

Managing Space Threshold Alerts for the Undo Tablespace

Oracle Database also provides proactive help in managing tablespace disk space use by alerting you when tablespaces run low on available space. See "Managing Tablespace Alerts" on page 19-1 for information on how to set alert thresholds for the undo tablespace.

In addition to the proactive undo space alerts, Oracle Database also provides alerts if your system has long-running queries that cause SNAPSHOT TOO OLD errors. To prevent excessive alerts, the long query alert is issued at most once every 24 hours. When the alert is generated, you can check the Undo Advisor Page of Enterprise Manager to get more information about the undo tablespace.

Migrating to Automatic Undo Management

If you are currently using rollback segments to manage undo space, Oracle strongly recommends that you migrate your database to automatic undo management.

For instructions, see *Oracle Database Upgrade Guide*.

Undo Space Data Dictionary Views

This section lists views that are useful for viewing information about undo space in the automatic undo management mode and provides some examples. In addition to views listed here, you can obtain information from the views available for viewing tablespace and datafile information. See "Datafiles Data Dictionary Views" on page 15-24 for information on getting information about those views.

The following dynamic performance views are useful for obtaining space information about the undo tablespace:

View	Description
V$UNDOSTAT	Contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. The database also uses this information to help tune undo usage in the system. This view is meaningful only in automatic undo management mode.
V$ROLLSTAT	For automatic undo management mode, information reflects behavior of the undo segments in the undo tablespace
V$TRANSACTION	Contains undo segment information
DBA_UNDO_EXTENTS	Shows the status and size of each extent in the undo tablespace.
DBA_HIST_UNDOSTAT	Contains statistical snapshots of V$UNDOSTAT information. See *Oracle Database 2 Day DBA* for more information.

> **See Also:** *Oracle Database Reference* for complete descriptions of the views used in automatic undo management mode

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, the tuning of undo retention, and the length and SQL ID of long-running 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 576 rows, spanning a 4 day cycle.

The following example shows the results of a query on the V$UNDOSTAT view.

```
SELECT TO_CHAR(BEGIN_TIME, 'MM/DD/YYYY HH24:MI:SS') BEGIN_TIME,
       TO_CHAR(END_TIME, 'MM/DD/YYYY HH24:MI:SS') END_TIME,
       UNDOTSN, UNDOBLKS, TXNCOUNT, MAXCONCURRENCY AS "MAXCON"
       FROM v$UNDOSTAT WHERE rownum <= 144;

BEGIN_TIME          END_TIME               UNDOTSN   UNDOBLKS   TXNCOUNT     MAXCON
------------------- ------------------- ---------- ---------- ---------- ----------
10/28/2004 14:25:12 10/28/2004 14:32:17          8         74   12071108          3
10/28/2004 14:15:12 10/28/2004 14:25:12          8         49   12070698          2
10/28/2004 14:05:12 10/28/2004 14:15:12          8        125   12070220          1
10/28/2004 13:55:12 10/28/2004 14:05:12          8         99   12066511          3
...
10/27/2004 14:45:12 10/27/2004 14:55:12          8         15   11831676          1
10/27/2004 14:35:12 10/27/2004 14:45:12          8        154   11831165          2

144 rows selected.
```

The preceding example shows how undo space is consumed in the system for the previous 24 hours from the time 14:35:12 on 10/27/2004.

17

Using Oracle Managed Files

In this chapter:

- What Are Oracle Managed Files?
- Enabling the Creation and Use of Oracle Managed Files
- Creating Oracle Managed Files
- Operation 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 that comprise an Oracle Database. With Oracle Managed Files, you specify file system directories in which the database automatically creates, names, and manages files at the database object level. For example, you need only specify that you want to create a tablespace; you do not need to specify the name and path of the tablespace's datafile with the `DATAFILE` clause. This feature works well with a logical volume manager (LVM).

The database internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Redo log files
- Control files
- Archived logs
- Block change tracking files
- Flashback logs
- RMAN backups

Through initialization parameters, you specify the file system directory to be used for a particular type of file. The database 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 logs, and core files.

> **See Also:** *Oracle Automatic Storage Management Administrator's Guide* for information about Oracle Automatic Storage Management (Oracle ASM), the Oracle Database integrated file system and volume manager that extends the power of Oracle Managed Files. With Oracle Managed Files, files are created and managed automatically for you, but with Oracle ASM, you get the additional benefits of features such as striping, software mirroring, and dynamic storage configuration, without the need to purchase a third-party logical volume manager.

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. However, 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 Database. 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 Database 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 table lists the initialization parameters that enable the use of Oracle Managed Files.

Initialization Parameter	Description
DB_CREATE_FILE_DEST	Defines the location of the default file system directory or Oracle ASM disk group where the database creates datafiles or tempfiles when no file specification is given in the create operation. Also used as the default location for redo log and control files if DB_CREATE_ONLINE_LOG_DEST_*n* are not specified.
DB_CREATE_ONLINE_LOG_DEST_*n*	Defines the location of the default file system directory or Oracle ASM disk group for redo log files and control file creation when no file specification is given in the create operation. By changing *n*, you can use this initialization parameter multiple times, where *n* specifies a multiplexed copy of the redo log or control file. You can specify up to five multiplexed copies.
DB_RECOVERY_FILE_DEST	Defines the location of the Fast Recovery Area, which is the default file system directory or Oracle ASM disk group where the database creates RMAN backups when no format option is used, archived logs when no other local destination is configured, and flashback logs. Also used as the default location for redo log and control files or multiplexed copies of redo log and control files if DB_CREATE_ONLINE_LOG_DEST_*n* are not specified.

The file system directories specified by these parameters must already exist; the database does not create them. The directory must also have permissions to allow the database to create the files in it.

The default location is used whenever a location is not explicitly specified for the operation creating the file. The database 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:

- *Oracle Database Reference* for additional information about initialization parameters
- "How Oracle Managed Files Are Named" on page 17-6

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
- Redo log files
- Control files
- Block change tracking 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/app/oracle/oradata` as the default directory to use when creating Oracle Managed Files:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
```

Setting the DB_RECOVERY_FILE_DEST Parameter

Include the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` parameters in your initialization parameter file to identify the default location for the Fast Recovery Area. The Fast Recovery Area contains:

- Redo log files or multiplexed copies of redo log files
- Control files or multiplexed copies of control files
- RMAN backups (datafile copies, control file copies, backup pieces, control file autobackups)
- Archived logs
- Flashback logs

You specify the name of file system directory that becomes the default location for creation of the operating system files for these entities. For example:

```
DB_RECOVERY_FILE_DEST = '/u01/app/oracle/fast_recovery_area'
DB_RECOVERY_FILE_DEST_SIZE = 20G
```

Setting the DB_CREATE_ONLINE_LOG_DEST_n Initialization Parameters

Include the `DB_CREATE_ONLINE_LOG_DEST_`*n* initialization parameters in your initialization parameter file to identify the default locations for the database server to create:

- Redo log files
- Control files

You specify the name of a file system directory or Oracle ASM disk group that becomes the default location for the creation of the files for these entities. You can specify up to five multiplexed locations.

For the creation of redo log files and control files only, this parameter overrides any default location specified in the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` initialization parameters. If you do not specify a `DB_CREATE_FILE_DEST` parameter, but you do specify the `DB_CREATE_ONLINE_LOG_DEST_`*n* parameter, then only 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'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

This allows multiplexing, which provides greater fault-tolerance for the 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 Database creates Oracle Managed Files for you, as appropriate, when no file specification is given in the create operation:

- You have included any of the DB_CREATE_FILE_DEST, DB_REDOVERY_FILE_DEST, or DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters in your initialization parameter file.
- You have issued the ALTER SYSTEM statement to dynamically set any of DB_RECOVERY_FILE_DEST, DB_CREATE_FILE_DEST, or DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameters
- You have issued the ALTER SESSION statement to dynamically set any of the DB_CREATE_FILE_DEST, DB_RECOVERY_FILE_DEST, or 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.

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 Using Oracle Managed Files
- Creating Tempfiles for Temporary Tablespaces Using Oracle Managed Files
- Creating Control Files Using Oracle Managed Files
- Creating Redo Log Files Using Oracle Managed Files
- Creating Archived Logs Using Oracle Managed Files

How Oracle Managed Files Are Named

Note: The naming scheme described in this section applies only to files created in operating system file systems. The naming scheme for files created in Oracle Automatic Storage Management (Oracle ASM) disk groups is described in *Oracle Automatic Storage Management Administrator's Guide*.

The filenames of Oracle Managed Files comply with the Optimal 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.
- Files of one database type are easily distinguishable from other database types.
- Files are clearly associated with important attributes specific to the file type. For example, a datafile name may include the tablespace name to allow for easy association of datafile to tablespace, or an archived log name may include the thread, sequence, and creation date.

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 creation location

- A file name template that is chosen based on the type of the file. The template also depends on the operating system platform and whether or not Oracle Automatic Storage Management is used.
- A unique string created by Oracle Database 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 a Solaris file system:

```
destination_prefix/o1_mf_%t_%u_.dbf
```

where:

- *destination_prefix* is *destination_location/db_unique_name/datafile*

  where:

  – *destination_location* is the location specified in DB_CREATE_FILE_DEST

  – *db_unique_name* is the globally unique name (DB_UNIQUE_NAME initialization parameter) of the target database. If there is no DB_UNIQUE_NAME parameter, then the DB_NAME initialization parameter value is used.
- %t is the tablespace name.
- %u is an eight-character string that guarantees uniqueness

For example, assume the following parameter settings:

```
DB_CREATE_FILE_DEST   = /u01/app/oracle/oradata
DB_UNIQUE_NAME = PAYROLL
```

Then an example datafile name would be:

```
/u01/app/oracle/oradata/PAYROLL/datafile/o1_mf_tbs1_2ixh90q_.dbf
```

Names for other file types are similar. Names on other platforms are also similar, subject to the constraints of the naming rules of the platform.

The examples on the following pages use Oracle managed file names as they might appear with a Solaris file system as an OMF destination.

Caution: Do not rename an Oracle managed file. The database identifies an Oracle managed file based on its name. If you rename the file, the database 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 actions of the CREATE DATABASE statement when using Oracle Managed Files are discussed in this section.

Note: The rules and defaults in this section also apply to creating a database with Database Configuration Assistant (DBCA). With DBCA, you use a graphical interface to enable Oracle Managed Files and to specify file locations that correspond to the initialization parameters described in this section.

> **See Also:** *Oracle Database SQL Language Reference* for a description of the CREATE DATABASE statement

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:

- One or more control files as specified in the DB_CREATE_ONLINE_LOG_DEST_*n* initialization parameter. The file in the first directory is the primary control file. When DB_CREATE_ONLINE_LOG_DEST_*n* is specified, the database does not create a control file in DB_CREATE_FILE_DEST or in DB_RECOVERY_FILE_DEST (the Fast Recovery Area).
- If no value is specified for DB_CREATE_ONLINE_LOG_DEST_*n*, but values are set for both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST, then the database creates one control file in each location. The location specified in DB_CREATE_FILE_DEST is the primary control file.
- If a value is specified only for DB_CREATE_FILE_DEST, then the database creates one control file in that location.
- If a value is specified only for DB_RECOVERY_FILE_DEST, then the database creates one control file in that location.

If the CONTROL_FILES parameter is not set and none of these initialization parameters are set, then the Oracle Database default action 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 the database creates an Oracle managed control file, and if there is a server parameter file, then the database 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 11, "Managing Control Files"

Specifying 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 redo log files. If the LOGFILE clause is omitted, then redo log files are created in the default redo log file destinations. In order of precedence, the default destination is defined as follows:

- If either the DB_CREATE_ONLINE_LOG_DEST_*n* is set, then the database creates a log file member in each directory specified, up to the value of the MAXLOGMEMBERS initialization parameter.
- If the DB_CREATE_ONLINE_LOG_DEST_*n* parameter is not set, but both the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters are set, then the database creates one Oracle managed log file member in each of those locations. The log file in the DB_CREATE_FILE_DEST destination is the first member.
- If only the DB_CREATE_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.

- If only the DB_RECOVERY_FILE_DEST initialization parameter is specified, then the database creates a log file member in that location.

The default size of an Oracle managed redo log file is 100 MB.

Optionally, you can create Oracle managed redo log files, and override default attributes, by including the LOGFILE clause but omitting a filename. Redo log files are created the same way, 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 12, "Managing the Redo Log"

Specifying the SYSTEM and SYSAUX Tablespace Datafiles at Database Creation

The DATAFILE or SYSAUX DATAFILE clause is not required in the CREATE DATABASE statement, and omitting it provides a simple means of creating Oracle managed datafiles for the SYSTEM and SYSAUX tablespaces. If the DATAFILE clause is omitted, then one of the following actions occurs:

- If DB_CREATE_FILE_DEST is set, then one Oracle managed datafile for the SYSTEM tablespace and another for the SYSAUX tablespace are created in the DB_CREATE_FILE_DEST directory.
- If DB_CREATE_FILE_DEST is not set, then the database creates one SYSTEM and one SYSAUX tablespace datafile whose names and sizes are operating system dependent. Any SYSTEM or SYSAUX tablespace datafile created in this manner is not an Oracle managed file.

By default, Oracle managed datafiles, including those for the SYSTEM and SYSAUX tablespaces, are 100MB and autoextensible. When autoextension is required, the database extends the datafile by its existing size or 100 MB, whichever is smaller. You can also explicitly specify the autoextensible unit using the NEXT parameter of the STORAGE clause when you specify the datafile (in a CREATE or ALTER TABLESPACE operation).

Optionally, you can create an Oracle managed datafile for the SYSTEM or SYSAUX 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 or SYSAUX 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 (the default), then a

default undo tablespace named SYS_UNDOTS is created and a 20 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 16, "Managing Undo"

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/app/oracle/oradata that is autoextensible up to an unlimited size.
- A SYSAUX tablespace datafile in directory /u01/app/oracle/oradata that is autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two online log groups with two members of 100 MB each, one each in /u02/oradata and /u03/oradata.
- If automatic undo management mode is enabled (the default), then an undo tablespace datafile in directory /u01/app/oracle/oradata that is 20 MB and autoextensible up to an unlimited size. An undo tablespace named SYS_UNDOTS is created.
- If no CONTROL_FILES initialization parameter is specified, then two control files, one each in /u02/oradata and /u03/oradata. The control file in /u02/oradata is the primary control file.

The following parameter settings relating to Oracle Managed Files, are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
CREATE DATABASE sample;
```

To create the database with a locally managed SYSTEM tablespace, add the EXTENT MANAGEMENT LOCAL clause:

```
CREATE DATABASE sample EXTENT MANAGEMENT LOCAL;
```

Without this clause, the SYSTEM tablespace is dictionary managed. Oracle recommends that you create a locally managed SYSTEM tablespace.

CREATE DATABASE: Example 2 This example creates a database with the following Oracle Managed Files:

- A SYSTEM tablespace datafile in directory /u01/app/oracle/oradata that is autoextensible up to an unlimited size.
- A SYSAUX tablespace datafile in directory /u01/app/oracle/oradata that is autoextensible up to an unlimited size. The tablespace is locally managed with automatic segment-space management.
- Two redo log files of 100 MB each in directory /u01/app/oracle/oradata. They are not multiplexed.
- An undo tablespace datafile in directory /u01/app/oracle/oradata that is 20 MB and autoextensible up to an unlimited size. An undo tablespace named SYS_UNDOTS is created.
- A control file in /u01/app/oracle/oradata.

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:

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata';
CREATE DATABASE sample2 EXTENT MANAGEMENT LOCAL;
```

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 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/app/oracle/oradata. Because SIZE is specified, the file in not autoextensible.
- A 200 MB SYSAUX tablespace datafile in directory /u01/app/oracle/oradata. Because SIZE is specified, the file in not autoextensible. The tablespace is locally managed with automatic segment-space management.
- Two redo log groups with two members of 100 MB each, one each in directories /u02/oradata and /u03/oradata.
- For the default temporary tablespace dflt_ts, a 10 MB tempfile in directory /u01/app/oracle/oradata. Because SIZE is specified, the file in not autoextensible.

- For the undo tablespace `undo_ts`, a 100 MB datafile in directory `/u01/app/oracle/oradata`. 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` and `/u03/oradata`. The control file in `/u02/oradata` is the primary control file.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/app/oracle/oradata'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
```

The following statement is issued at the SQL prompt:

```
CREATE DATABASE sample3
EXTENT MANAGEMENT LOCAL
DATAFILE SIZE 400M
SYSAUX DATAFILE SIZE 200M
DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
UNDO TABLESPACE undo_ts DATAFILE SIZE 100M;
```

See Also: "Creating a Locally Managed SYSTEM Tablespace" on page 2-17

Creating Datafiles for Tablespaces Using Oracle Managed Files

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 permanent 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 permanent 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 and SYSAUX Tablespace Datafiles at Database Creation" on page 17-9
- "Specifying the Undo Tablespace Datafile at Database Creation" on page 17-9
- Chapter 14, "Managing Tablespaces"

CREATE TABLESPACE: Examples

The following are some examples of creating tablespaces with Oracle Managed Files.

> **See Also:** *Oracle Database SQL Language Reference* for a description of the `CREATE TABLESPACE` statement

CREATE TABLESPACE: Example 1 The following example sets the default location for datafile creations to `/u01/oradata` 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';
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`. The datafile initial size is 400 MB, and because the SIZE clause is specified, the datafile is not autoextensible.

The following parameter setting is included in the initialization parameter file:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M;
```

CREATE TABLESPACE: Example 3 This example creates a tablespace named `tbs_3` with an autoextensible datafile in the directory `/u01/oradata` 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'
```

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` 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';
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`. 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'
```

The following statement is issued at the SQL prompt:

```
SQL> CREATE UNDO TABLESPACE undotbs_1;
```

> **See Also:** *Oracle Database SQL Language Reference* for a description of the `CREATE UNDO TABLESPACE` statement

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'
```

The following statement is entered at the SQL prompt:

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

> **See Also:** *Oracle Database SQL Language Reference* for a description of the `ALTER TABLESPACE` statement

Creating Tempfiles for Temporary Tablespaces Using Oracle Managed Files

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 17-10

CREATE TEMPORARY TABLESPACE: Example

The following example sets the default location for datafile creations to `/u01/oradata` 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';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

See Also: *Oracle Database SQL Language Reference* for a description of the `CREATE TABLESPACE` statement

ALTER TABLESPACE... ADD TEMPFILE: Example

The following example sets the default location for datafile creations to `/u03/oradata` and then adds a tempfile in the default location to a tablespace named `temptbs_1`. The tempfile initial size is 100 MB. It is autoextensible with an unlimited maximum size.

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

See Also: *Oracle Database SQL Language Reference* for a description of the `ALTER TABLESPACE` statement

Creating Control Files Using Oracle Managed 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. The default destination is determined according to the precedence documented in "Specifying Control Files at Database Creation" on page 17-8.

If Oracle Database creates an Oracle managed control file, and there is a server parameter file, then the database 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 database-generated filenames for the files must be supplied in the `DATAFILE` clause of the statement.

If the 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 database-generated filenames for the Oracle managed redo log files must be supplied in the `LOGFILE` clause.
- If the `RESETLOGS` keyword is used, then the redo log file names can be supplied as with the `CREATE DATABASE` statement. See "Specifying Redo Log Files at Database Creation" on page 17-8.

The sections that follow contain examples of using the `CREATE CONTROLFILE` statement with Oracle Managed Files.

See Also:

- *Oracle Database SQL Language Reference* for a description of the `CREATE CONTROLFILE` statement
- "Specifying Control Files at Database Creation" on page 17-8

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 redo log files:

```
CREATE CONTROLFILE
     DATABASE sample
     LOGFILE
```

```
  GROUP 1 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_1_o220rtt9_.log',
           '/u02/oradata/SAMPLE/onlinelog/o1_mf_1_v2o0b2i3_.log')
            SIZE 100M,
  GROUP 2 ('/u01/oradata/SAMPLE/onlinelog/o1_mf_2_p22056iw_.log',
           '/u02/oradata/SAMPLE/onlinelog/o1_mf_2_p02rcyg3_.log')
            SIZE 100M
NORESETLOGS
DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_xu34ybm2_.dbf'
         SIZE 100M,
         '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_aawbmz51_.dbf'
         SIZE 100M,
         '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_apqbmz51_.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. Some combination of `DB_CREATE_FILE_DEST`, `DB_RECOVERY_FILE_DEST`, and `DB_CREATE_ONLINE_LOG_DEST_n` or must be set.

```
CREATE CONTROLFILE
     DATABASE sample
     RESETLOGS
     DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_aawbmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_axybmz51_.dbf',
              '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_azzbmz51_.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 redo log files. This is discussed in "Using the ALTER DATABASE OPEN RESETLOGS Statement" on page 17-17. If the previous log files are Oracle Managed Files, then they are not deleted.

Creating Redo Log Files Using Oracle Managed Files

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`

> **See Also:** *Oracle Database SQL Language Reference* for a description of the `ALTER DATABASE` statement

Using the ALTER DATABASE ADD LOGFILE Statement

The `ALTER DATABASE ADD LOGFILE` statement lets you later add a new group to your current 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. The default destination is determined according to the precedence documented in "Specifying Redo Log Files at Database Creation" on page 17-8.

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 redo log file members by specifying complete filenames.

See Also:

- "Specifying Redo Log Files at Database Creation" on page 17-8
- "Creating Control Files Using Oracle Managed Files" on page 17-15

Adding New Redo Log Files: Example The following example creates a log group with a member in `/u01/oradata` and another member in `/u02/oradata`. The size of each log file is 100 MB.

The following parameter settings are included in the initialization parameter file:

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata'
```

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 nonexistent filenames, then the database creates 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 Redo Log Files at Database Creation" on page 17-8.

Creating Archived Logs Using Oracle Managed Files

Archived logs are created in the `DB_RECOVERY_FILE_DEST` location when:

- The `ARC` or `LGWR` background process archives an online redo log or
- An `ALTER SYSTEM ARHIVE LOG CURRENT` statement is issued.

For example, assume that the following parameter settings are included in the initialization parameter file:

```
DB_RECOVERY_FILE_DEST_SIZE = 20G
DB_RECOVERY_FILE_DEST      = '/u01/oradata'
LOG_ARCHIVE_DEST_1         = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

Operation 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 database-generated filenames:

```
SQL> ALTER DATABASE
  2> RENAME FILE '/u01/oradata/mydb/datafile/o1_mf_tbs01_ziw3bopb_.dbf'
  3> TO '/u01/oradata/mydb/tbs0101.dbf';

SQL> ALTER DATABASE
  2> DROP LOGFILE '/u01/oradata/mydb/onlinelog/o1_mf_1_wo94n2xi_.log';

SQL> ALTER TABLE emp
  2> ALLOCATE EXTENT
  3> (DATAFILE '/u01/oradata/mydb/datafile/o1_mf_tbs1_2ixfh90q_.dbf');
```

You can backup and restore Oracle managed datafiles, tempfiles, and control files as you would corresponding non Oracle Managed Files. Using database-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 managed by the database, 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

You can also use these statements, which always delete files, Oracle managed or not:

- ALTER TABLESPACE ... DROP DATAFILE
- ALTER TABLESPACE ... DROP TEMPFILE

Dropping Redo Log Files

When an Oracle managed 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 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 redo log files in a standby database can be managed by the database. 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 Redo Logs

In this scenario, a DBA creates a database where the datafiles and redo log files are created in separate directories. The 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 (the default) is also specified.

   ```
   DB_CREATE_FILE_DEST = '/u01/oradata'
   DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata'
   DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata'
   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 redo log file and control file creation. Each 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 dflttmp;
   ```

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 in this scenario). The filename is uniquely generated by the database. The file is autoextensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle managed file. A similar datafile is created for the SYSAUX tablespace.

Because a LOGFILE clause is not present, two 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 the database. 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 the database. 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 UNDOTBS is created in the directory specified by DB_CREATE_FILE_DEST. The datafile is a 20 MB datafile that is autoextensible. It is an Oracle managed file.

Lastly, a default temporary tablespace named dflttmp is specified. Because DB_CREATE_FILE_DEST is included in the parameter file, the tempfile for dflttmp 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
               /datafile
                    /o1_mf_system_cmr7t30p_.dbf
                    /o1_mf_sysaux_cmr7t88p_.dbf
                    /o1_mf_sys_undo_2ixfh90q_.dbf
                    /o1_mf_dflttmp_157se6ff_.tmp
/u02
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_0orrm31z_.log
                    /o1_mf_2_2xyz16am_.log
               /controlfile
                    /o1_mf_cmr7t30p_.ctl
/u03
     /oradata
          /SAMPLE
               /onlinelog
                    /o1_mf_1_ixfvm8w9_.log
                    /o1_mf_2_q89tmp28_.log
               /controlfile
                    /o1_mf_x1sr8t36_.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/datafile/o1_mf_system_cmr7t30p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf
/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf

3 rows selected
```

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:

```
CREATE CONTROLFILE REUSE
  DATABASE sample
  LOGFILE
    GROUP 1('/u02/oradata/SAMPLE/onlinelog/o1_mf_1_0orrm31z_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_1_ixfvm8w9_.log'),
    GROUP 2('/u02/oradata/SAMPLE/onlinelog/o1_mf_2_2xyz16am_.log',
            '/u03/oradata/SAMPLE/onlinelog/o1_mf_2_q89tmp28_.log')
  NORESETLOGS
  DATAFILE '/u01/oradata/SAMPLE/datafile/o1_mf_system_cmr7t30p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sysaux_cmr7t88p_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_sys_undo_2ixfh90q_.dbf',
           '/u01/oradata/SAMPLE/datafile/o1_mf_dflttmp_157se6ff_.tmp'
  MAXLOGFILES 5
  MAXLOGHISTORY 100
  MAXDATAFILES 10
  MAXINSTANCES 2
  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 redo log

   To create a new group of 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 group. 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 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. A datafile is created with an initial 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, the database 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';
   ```

6. Archiving redo information

   Archiving of 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: Create and Manage a Database with Database and Fast Recovery Areas

In this scenario, a DBA creates a database where the control files and redo log files are multiplexed. Archived logs and RMAN backups are created in the Fast Recovery Area. The following tasks are involved in creating and maintaining this database:

1. Setting the initialization parameters

The DBA includes the following generic file creation defaults:

```
DB_CREATE_FILE_DEST = '/u01/oradata'
DB_RECOVERY_FILE_DEST_SIZE = 10G
DB_RECOVERY_FILE_DEST = '/u02/oradata'
LOG_ARCHIVE_DEST_1 = 'LOCATION = USE_DB_RECOVERY_FILE_DEST'
```

The `DB_CREATE_FILE_DEST` parameter sets the default file system directory for datafiles, tempfiles, control files, and redo logs.

The `DB_RECOVERY_FILE_DEST` parameter sets the default file system directory for control files, redo logs, and RMAN backups.

The `LOG_ARCHIVE_DEST_1` configuration '`LOCATION=USE_DB_RECOVERY_FILE_DEST`' redirects archived logs to the `DB_RECOVERY_FILE_DEST` location.

The `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` parameters set the default directory for log file and control file creation. Each redo log and control file is multiplexed across the two directories.

2. Creating a database
3. Managing control files
4. Managing the redo log
5. Managing tablespaces

   Tasks 2, 3, 4, and 5 are the same as in Scenario 1, except that the control files and redo logs are multiplexed across the `DB_CREATE_FILE_DEST` and `DB_RECOVERY_FILE_DEST` locations.

6. Archiving redo log information

   Archiving online logs is no different for Oracle Managed Files than it is for unmanaged files. The archived logs are created in `DB_RECOVERY_FILE_DEST` and are Oracle Managed Files.

7. Backup, restore, and recover

   An Oracle managed file is compatible with standard operating system files, so 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. When no format option is specified, all disk backups by RMAN are created in the `DB_RECOVERY_FILE_DEST` location. The backups are Oracle Managed Files.

Scenario 3: Adding 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`.

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';
   ```

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 tbs_2 tablespace is dropped, its datafiles are automatically deleted.

Part III

Schema Objects

Part III describes how to create and manage schema objects in Oracle Database. It includes the following chapters:

- Chapter 18, "Managing Schema Objects"
- Chapter 19, "Managing Space for Schema Objects"
- Chapter 20, "Managing Tables"
- Chapter 21, "Managing Indexes"
- Chapter 22, "Managing Clusters"
- Chapter 23, "Managing Hash Clusters"
- Chapter 24, "Managing Views, Sequences, and Synonyms"
- Chapter 25, "Repairing Corrupted Data"

18

Managing Schema Objects

In this chapter:

- Creating Multiple Tables and Views in a Single Operation
- Analyzing Tables, Indexes, and Clusters
- Truncating Tables and Clusters
- Enabling and Disabling Triggers
- Managing Integrity Constraints
- Renaming Schema Objects
- Managing Object Dependencies
- Managing Object Name Resolution
- Switching to a Different Schema
- Managing Editions
- Displaying Information About Schema Objects

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. 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 Database extensions to the ANSI `CREATE TABLE` and `CREATE VIEW` statements, including the `STORAGE` clause.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and other information about the `CREATE SCHEMA` statement

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:** Do not use the `COMPUTE` and `ESTIMATE` clauses of `ANALYZE` to collect optimizer statistics. These clauses have been deprecated. Instead, use the `DBMS_STATS` package, which lets you collect statistics in parallel, collect global statistics for partitioned objects, and fine tune your statistics collection in other ways. The cost-based optimizer, which depends upon statistics, will eventually use only statistics that have been collected by `DBMS_STATS`. See *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_STATS` package.
>
> 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:

- Using DBMS_STATS to Collect Table and Index Statistics
- Validating Tables, Indexes, Clusters, and Materialized Views
- Listing Chained Rows of Tables and Clusters

Using DBMS_STATS to Collect Table and Index Statistics

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 statistics unrelated to the optimizer, such as empty blocks, average space, and so forth.

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:

- *Oracle Database Performance Tuning Guide* for information about using DBMS_STATS to gather statistics for the optimizer
- *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_STATS 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 dependent 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;
```

By default the CASCADE option performs a complete validation. Because this operation can be resource intensive, you can perform a faster version of the validation by using the FAST clause. This version checks for the existence of corruptions using an optimized check algorithm, but does not report details about the corruption. If the FAST check finds a corruption, you can then use the CASCADE option without the FAST clause to locate it. The following statement performs a fast validation on the emp table and all associated indexes:

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE FAST;
```

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;
```

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

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.

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 the database. 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 depends on the compatibility level of your database and the type of table you are analyzing. See the *Oracle Database SQL Language 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:**
>
> - *Oracle Database Reference* for a description of the CHAINED_ROWS table
> - "Using the Segment Advisor" on page 19-12 for information on how the Segment Advisor reports tables with excess row chaining.

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       ... AAAAluAAHAAAAA1AAA  04-MAR-96
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAB  04-MAR-96
SCOTT       ORDER_HIST       ... AAAAluAAHAAAAA1AAC  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.

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 large `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 must 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.

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 undo 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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and other information about the DELETE statement

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 undo 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.

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 TRUNCATE statement has several options that control whether space currently allocated for a table or cluster is returned to the containing tablespace after truncation.

These options also apply 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.

These TRUNCATE options are:

- DROP STORAGE, the default option, 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.
- DROP ALL STORAGE drops the segment. In addition to the TRUNCATE TABLE statement, DROP ALL STORAGE also applies to the ALTER TABLE TRUNCATE (SUB)PARTITION statement. This option also drops any dependent object segments associated with the partition being truncated.

  DROP ALL STORAGE is not supported for clusters.

  Note: This functionality is available with Oracle Database 11*g* release 2 (11.2.0.2).

  ```
  TRUNCATE TABLE emp DROP ALL STORAGE;
  ```

- REUSE STORAGE 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;
  ```

See Also:

- *Oracle Database SQL Language Reference* for syntax and other information about the TRUNCATE TABLE and TRUNCATE CLUSTER statements
- *Oracle Database Security Guide* for information about auditing

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 the database 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 *Oracle Database SQL Language 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:

- *Oracle Database Concepts* for a more detailed description of triggers
- *Oracle Database SQL Language Reference* for syntax of the `CREATE TRIGGER` statement
- *Oracle Database PL/SQL Language Reference* 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;
```

See Also: *Oracle Database SQL Language Reference* for syntax and other information about the `ALTER TRIGGER` statement

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:

- *Oracle Database Concepts* for a more thorough discussion of integrity constraints
- *Oracle Database Advanced Application Developer's Guide* 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 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 *Oracle Database SQL Language 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 preceding bullet list.

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 18-14. 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.

Efficient Use of Integrity Constraints: A Procedure

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 definition, the database 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.

Note: An efficient procedure for enabling a constraint that can make use of parallelism is described in "Efficient Use of Integrity Constraints: A Procedure" on page 18-11.

See Also: "Creating an Index Associated with a Constraint" on page 21-9

Modifying, Renaming, or Dropping Existing Integrity Constraints

You can use the ALTER TABLE statement to enable, disable, modify, or drop a constraint. When the database 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 the database 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.

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 depends on the type of table you are analyzing. See the *Oracle Database SQL Language 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`, then the following query will display those exceptions:

```
SELECT * FROM EXCEPTIONS;
```

The following exceptions are shown:

```
fROWID               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 statement and results:

```
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 marking the constraint `ENABLE NOVALIDATE` before you start eliminating exceptions.

See Also: *Oracle Database Reference* for a description of the `EXCEPTIONS` table

Viewing Constraint Information

Oracle Database 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: *Oracle Database Reference* contains descriptions of the columns in these views

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
- Rename the object using the `ALTER` ... `RENAME` statement (for indexes and triggers)

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.

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 a stored PL/SQL program unit, public synonym, 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:** *Oracle Database SQL Language Reference* for syntax of the `RENAME` statement

Managing Object Dependencies

This section provides background information about object dependencies and object invalidation, and explains how invalid objects can be revalidated. The following topics are included:

- About Object Dependencies and Object Invalidation
- Manually Recompiling Invalid Objects with DDL
- Manually Recompiling Invalid Objects with PL/SQL Package Procedures

About Object Dependencies and Object Invalidation

Some types of schema objects reference other objects. For example, a view contains a query that references tables or other views, and a PL/SQL subprogram might invoke other subprograms and might use static SQL to reference tables or views. An object that references another object is called a **dependent object**, and an object being referenced is a **referenced object**. These references are established at compile time, and if the compiler cannot resolve them, the dependent object being compiled is marked *invalid*.

Oracle Database provides an automatic mechanism to ensure that a dependent object is always up to date with respect to its referenced objects. When a dependent object is created, the database tracks dependencies between the dependent object and its referenced objects. When a referenced object is changed in a way that might affect a dependent object, the dependent object is marked invalid. An invalid dependent object must be recompiled against the new definition of a referenced object before the dependent object can be used. Recompilation occurs automatically when the invalid dependent object is referenced.

It is important to be aware of changes that can invalidate schema objects, because invalidation affects applications running on the database. This section describes how objects become invalid, how you can identify invalid objects, and how you can validate invalid objects.

Object Invalidation

In a typical running application, you would not expect to see views or stored procedures become invalid, because applications typically do not change table structures or change view or stored procedure definitions during normal execution. Changes to tables, views, or PL/SQL units typically occur when an application is

patched or upgraded using a patch script or ad-hoc DDL statements. Dependent objects might be left invalid after a patch has been applied to change a set of referenced objects.

Use the following query to display the set of invalid objects in the database:

```
SELECT object_name, object_type FROM dba_objects
WHERE status = 'INVALID';
```

The Database Home page in Enterprise Manager displays an alert when schema objects become invalid.

Object invalidation affects applications in two ways. First, an invalid object must be revalidated before it can be used by an application. Revalidation adds latency to application execution. If the number of invalid objects is large, the added latency on the first execution can be significant. Second, invalidation of a procedure, function or package can cause exceptions in other sessions concurrently executing the procedure, function or package. If a patch is applied when the application is in use in a different session, the session executing the application notices that an object in use has been invalidated and raises one of the following 4 exceptions: ORA-04061, ORA-04064, ORA-04065 or ORA-04068. These exceptions must be remedied by restarting application sessions following a patch.

You can force the database to recompile a schema object using the appropriate SQL statement with the `COMPILE` clause. See "Manually Recompiling Invalid Objects with DDL" on page 18-18 for more information.

If you know that there are a large number of invalid objects, use the `UTL_RECOMP` PL/SQL package to perform a mass recompilation. See "Manually Recompiling Invalid Objects with PL/SQL Package Procedures" on page 18-19 for details.

The following are some general rules for the invalidation of schema objects:

- Between a referenced object and each of its dependent objects, the database tracks the elements of the referenced object that are involved in the dependency. For example, if a single-table view selects only a subset of columns in a table, only those columns are involved in the dependency. For each dependent of an object, if a change is made to the definition of any element involved in the dependency (including dropping the element), the dependent object is invalidated. Conversely, if changes are made only to definitions of elements that are not involved in the dependency, the dependent object remains valid.

  In many cases, therefore, developers can avoid invalidation of dependent objects and unnecessary extra work for the database if they exercise care when changing schema objects.

- Dependent objects are *cascade invalidated*. If any object becomes invalid for any reason, all of that object's dependent objects are immediately invalidated.

- If you revoke any object privileges on a schema object, dependent objects are cascade invalidated.

> **See Also:** *Oracle Database Concepts* for more detailed information about schema object dependencies

Manually Recompiling Invalid Objects with DDL

You can use an `ALTER` statement to manually recompile a single schema object. For example, to recompile package body `Pkg1`, you would execute the following DDL statement:

```
ALTER PACKAGE pkg1 COMPILE REUSE SETTINGS;
```

> **See Also:** *Oracle Database SQL Language Reference* for syntax and other information about the various ALTER statements

Manually Recompiling Invalid Objects with PL/SQL Package Procedures

Following an application upgrade or patch, it is good practice to revalidate invalid objects to avoid application latencies that result from on-demand object revalidation. Oracle provides the UTL_RECOMP package to assist in object revalidation. The RECOMP_SERIAL procedure recompiles all invalid objects in a specified schema, or all invalid objects in the database if you do not supply the schema name argument. The RECOMP_PARALLEL procedure does the same, but in parallel, employing multiple CPUs.

Examples

Execute the following PL/SQL block to revalidate all invalid objects in the database, in parallel and in dependency order:

```
begin
   utl_recomp.recomp_parallel();
end;
/
```

You can also revalidate individual invalid objects using the package DBMS_UTILITY. The following PL/SQL block revalidates the procedure UPDATE_SALARY in schema HR:

```
begin
   dbms_utility.validate('HR', 'UPDATE_SALARY', namespace=>1);
end;
/
```

The following PL/SQL block revalidates the package body HR.ACCT_MGMT:

```
begin
   dbms_utility.validate('HR', 'ACCT_MGMT', namespace=>2);
end;
/
```

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information on the UTL_RECOMP and DBMS_UTILITY packages.

Managing Object Name Resolution

Object names referenced in SQL statements can consist of several pieces, separated by periods. The following describes how the database resolves an object name.

1. Oracle Database 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, the database 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.** The database 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.** The database 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, the database returns an error.

If no schema is found in step c, the object cannot be qualified and the database 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 database resolves the reference locally. For example, it resolves a synonym to global object name of a remote table. The partially resolved statement is shipped to the remote database, and the remote database completes the resolution of the object as described here.

Because of how the database 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, the database 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, the database 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, the database attempts to recompile the view. As the database 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, the database 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.

> **See Also:** "Schema Objects and Database Links" on page 31-14 for information about name resolution in a distributed database

Switching to a Different Schema

The following statement sets the schema of the current session to the schema name specified in the statement.

```
ALTER SESSION SET CURRENT_SCHEMA = <schema name>
```

In subsequent SQL statements, Oracle Database uses this schema name as the schema qualifier when the qualifier is omitted. In addition, the database uses the temporary tablespace of the specified schema for sorts, joins, and storage of temporary database objects. The session retains its original privileges and does not acquire any extra privileges by the preceding `ALTER SESSION` statement.

In the following example, provide the password when prompted:

```
CONNECT scott
ALTER SESSION SET CURRENT_SCHEMA = joe;
SELECT * FROM emp;
```

Because `emp` is not schema-qualified, the table name is resolved under schema `joe`. But if `scott` does not have select privilege on table `joe.emp`, then `scott` cannot execute the `SELECT` statement.

Managing Editions

Application developers who are upgrading their applications using edition-based redefinition may ask you to perform edition-related tasks that require DBA privileges.

In this section:

- About Editions and Edition-Based Redefinition
- DBA Tasks for Edition-Based Redefinition
- Setting the Database Default Edition
- Querying the Database Default Edition
- Setting the Edition Attribute of a Database Service
- Using an Edition
- Editions Data Dictionary Views

About Editions and Edition-Based Redefinition

Edition-based redefinition enables you to upgrade an application's database objects while the application is in use, thus minimizing or eliminating down time. This is

accomplished by changing (redefining) database objects in a private environment known as an **edition**. Only when all changes have been made and tested do you make the new version of the application available to users.

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for a complete discussion of edition-based redefinition

DBA Tasks for Edition-Based Redefinition

Table 18–1 summarizes the edition-related tasks that require privileges typically granted only to DBAs. Any user that is granted the `DBA` role can perform these tasks.

Table 18–1 DBA Tasks for Edition-Based Redefinition

Task	See
Grant or revoke privileges to create, alter, and drop editions	The `CREATE EDITION` and `DROP EDITION` commands in *Oracle Database SQL Language Reference*
Enable editions for a schema	*Oracle Database Advanced Application Developer's Guide*
Set the database default edition	"Setting the Database Default Edition" on page 18-22
Set the edition attribute of a database service	"Setting the Edition Attribute of a Database Service" on page 18-23

Setting the Database Default Edition

There is always a default edition for the database. This is the edition that a database session initially uses if it does not explicitly indicate an edition when connecting.

To set the database default edition:

1. Connect to the database as a user with the `ALTER DATABASE` privilege.
2. Enter the following statement:

```
ALTER DATABASE DEFAULT EDITION = edition_name;
```

> **See Also:** "Connecting to the Database with SQL*Plus" on page 1-7

Querying the Database Default Edition

The database default edition is stored as a database property.

To query the database default edition:

1. Connect to the database as any user.
2. Enter the following statement:

```
SELECT PROPERTY_VALUE FROM DATABASE_PROPERTIES WHERE
   PROPERTY_NAME = 'DEFAULT_EDITION';

PROPERTY_VALUE
------------------------------
ORA$BASE
```

> **Note:** The property name `DEFAULT_EDITION` is case sensitive and must be supplied as upper case.

Setting the Edition Attribute of a Database Service

> **Note:** This functionality is available starting with Oracle Database 11*g* release 2 (11.2.0.2).

You can set the edition attribute of a database service when you create the service, or you can modify an existing database service to set its edition attribute. When you set the edition attribute of a service, all subsequent connections that specify the service, such as client connections and `DBMS_SCHEDULER` jobs, use this edition as the initial session edition. However, if a session connection specifies a different edition, then the edition specified in the session connection is used for the session edition. To check the edition attribute of a database service, query the `EDITION` column in the `ALL_SERVICES` view or the `DBA_SERVICES` view.

> **Note:** The number of database services for an instance has an upper limit. See *Oracle Database Reference* for more information about this limit.

Setting the Edition Attribute During Database Service Creation

Follow the instructions in "Creating Database Services" on page 2-41 and use the appropriate option for setting the edition attribute for the database service:

- If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to create the database service and specify the `-t` option to set its edition attribute.

  For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example creates a new database service named `crmbatch` and sets the edition attribute of the database service to `e2`:

  ```
  srvctl add service -d dbcrm -s crmbatch -t e2
  ```

- If your single-instance database is not being managed by Oracle Restart, use the `DBMS_SERVICE.CREATE_SERVICE` procedure, and specify the `edition` parameter to set the edition attribute of the database service.

Setting the Edition Attribute of an Existing Database Service

You can use the `SRVCTL` utility or the `DBMS_SERVICE` package to set the edition attribute of an existing database service.

To set the edition attribute of an existing database service:

1. Stop the database service.
2. Set the edition attribute of the database service using the appropriate option:
   - If your single-instance database is being managed by Oracle Restart, use the `SRVCTL` utility to modify the database service and specify the `-t` option to set its edition attribute.

     For the database with the `DB_UNIQUE_NAME` of `dbcrm`, this example modifies a database service named `crmbatch` and sets the edition attribute of the service to `e3`:

     ```
     srvctl modify service -d dbcrm -s crmbatch -t e3
     ```

- If your single-instance database is not being managed by Oracle Restart, use the `DBMS_SERVICE.MODIFY_SERVICE` procedure, and specify the `edition` parameter to set the edition attribute of the database service. Ensure that the `modify_edition` parameter is set to `TRUE` when you run the `MODIFY_SERVICE` procedure.

3. Start the database service.

See Also:

- Chapter 4, "Configuring Automatic Restart of an Oracle Database" for information managing database services using Oracle Restart
- *Oracle Database PL/SQL Packages and Types Reference* for information about managing database services using the `DBMS_SERVICE` package

Using an Edition

To view or modify objects in a particular edition, you must *use* the edition first. You can specify an edition to use when you connect to the database. If you do not specify an edition, your session starts in the database default edition. To use a different edition, submit the following statement:

```
ALTER SESSION SET EDITION=edition_name;
```

The following statements first set the current edition to `e2` and then to `ora$base`:

```
ALTER SESSION SET EDITION=e2;
...
ALTER SESSION SET EDITION=ora$base;
```

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information about using editions, and for instructions for determining the current edition
- "Connecting to the Database with SQL*Plus" on page 1-7

Editions Data Dictionary Views

There are several data dictionary views that aid with managing editions. The following table lists three of them. For a complete list, see *Oracle Database Advanced Application Developer's Guide*.

View	Description
`*_EDITIONS`	Lists all editions in the database. (Note: `USER_EDITIONS` does not exist.)
`*_OBJECTS`	Describes every object in the database that is visible (actual or inherited) in the current edition.
`*_OBJECTS_AE`	Describes every actual object in the database, across all editions.

Displaying Information About Schema Objects

Oracle Database provides a PL/SQL package that enables you to determine the DDL that created an object and data dictionary views that you can use to display information about schema objects. Packages and views that are unique to specific

types of schema objects are described in the associated chapters. This section describes views and packages that are generic in nature and apply to multiple schema objects.

Using a PL/SQL Package to Display Information About Schema Objects

The Oracle-supplied PL/SQL package procedure DBMS_METADATA.GET_DDL lets you obtain metadata (in the form of DDL used to create the object) about a schema object.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_METADATA package

Example: 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 the following statements 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');
```

The output from DBMS_METADATA.GET_DDL is a LONG data type. When using SQL*Plus, your output may be truncated by default. Issue the following SQL*Plus command before issuing the DBMS_METADATA.GET_DDL statement to ensure that your output is not truncated:

```
SQL> SET LONG 9999
```

Schema Objects Data Dictionary Views

These views display general 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	List all dependencies between procedures, packages, functions, package bodies, and triggers, including dependencies on views without any database links.

> **See Also:** *Oracle Database Reference* for a complete description of data dictionary views

The following are examples of using some of these views:

- Example 1: Displaying Schema Objects By Type
- Example 2: Displaying Dependencies of Views and Synonyms

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;
```

The following is the query output:

```
OBJECT_NAME                OBJECT_TYPE
-------------------------  -------------------
EMP_DEPT                   CLUSTER
EMP                        TABLE
DEPT                       TABLE
EMP_DEPT_INDEX             INDEX
PUBLIC_EMP                 SYNONYM
EMP_MGR                    VIEW
```

Example 2: 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_DEPENDENCIES`, `USER_DEPENDENCIES`, and `DBA_DEPENDENCIES` data dictionary views can be used to reveal the dependencies for a view. The `ALL_SYNONYMS`, `USER_SYNONYMS`, and `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 user `jward`:

```
SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
    FROM DBA_SYNONYMS
    WHERE OWNER = 'JWARD';
```

The following is the query output:

```
TABLE_OWNER             TABLE_NAME   SYNONYM_NAME
----------------------  -----------  -----------------
SCOTT                   DEPT         DEPT
SCOTT                   EMP          EMP
```

19

Managing Space for Schema Objects

In this chapter:

- Managing Tablespace Alerts
- Managing Resumable Space Allocation
- Reclaiming Wasted Space
- Dropping Unused Object Storage
- Understanding Space Usage of Data Types
- Displaying Information About Space Usage for Schema Objects
- Capacity Planning for Database Objects

Managing Tablespace Alerts

Oracle Database provides proactive help in managing disk space for tablespaces by alerting you when available space is running low. Two alert thresholds are defined by default: **warning** and **critical**. The warning threshold is the limit at which space is beginning to run low. The critical threshold is a serious limit that warrants your immediate attention. The database issues alerts at both thresholds.

There are two ways to specify alert thresholds for both locally managed and dictionary managed tablespaces:

- By percent full

  For both warning and critical thresholds, when space used becomes greater than or equal to a percent of total space, an alert is issued.

- By free space remaining (in kilobytes (KB))

  For both warning and critical thresholds, when remaining space falls below an amount in KB, an alert is issued. Free-space-remaining thresholds are more useful for very large tablespaces.

Alerts for locally managed tablespaces are server-generated. For dictionary managed tablespaces, Enterprise Manager provides this functionality. See "Monitoring Database Operations with Server-Generated Alerts" on page 8-4 for more information.

New tablespaces are assigned alert thresholds as follows:

- **Locally managed tablespace**—When you create a new locally managed tablespace, it is assigned the default threshold values defined for the database. A newly created database has a default of 85% full for the warning threshold and 97% full for the critical threshold. Defaults for free space remaining thresholds for

a new database are both zero (disabled). You can change these database defaults, as described later in this section.

- **Dictionary managed tablespace**—When you create a new dictionary managed tablespace, it is assigned the threshold values that Enterprise Manager lists for "All others" in the metrics categories "Tablespace Free Space (MB) (dictionary managed)" and "Tablespace Space Used (%) (dictionary managed)." You change these values on the Metric and Policy Settings page.

> **Note:** In a database that is upgraded from version 9.x or earlier to 10.x, database defaults for all locally managed tablespace alert thresholds are set to zero. This setting effectively disables the alert mechanism to avoid excessive alerts in a newly migrated database.

Setting Alert Thresholds

For each tablespace, you can set just percent-full thresholds, just free-space-remaining thresholds, or both types of thresholds simultaneously. Setting either type of threshold to zero disables it.

The ideal setting for the warning threshold is one that issues an alert early enough for you to resolve the problem before it becomes critical. The critical threshold should be one that issues an alert still early enough so that you can take immediate action to avoid loss of service.

To set alert threshold values for locally managed tablespaces:

- Do one of the following:
  - Use the Tablespaces page of Enterprise Manager.

    See *Oracle Database 2 Day DBA* for instructions.
  - Use the `DBMS_SERVER_ALERT.SET_THRESHOLD` package procedure.

    See *Oracle Database PL/SQL Packages and Types Reference* for details.

To set alert threshold values for dictionary managed tablespaces:

- Use the Tablespaces page of Enterprise Manager.

  See *Oracle Database 2 Day DBA* for instructions.

Example—Locally Managed Tablespace

The following example sets the free-space-remaining thresholds in the `USERS` tablespace to 10 MB (warning) and 2 MB (critical), and disables the percent-full thresholds.

```
BEGIN
DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_BYT_FREE,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_LE,
   warning_value           => '10240',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_LE,
   critical_value          => '2048',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');
```

```
DBMS_SERVER_ALERT.SET_THRESHOLD(
   metrics_id              => DBMS_SERVER_ALERT.TABLESPACE_PCT_FULL,
   warning_operator        => DBMS_SERVER_ALERT.OPERATOR_GT,
   warning_value           => '0',
   critical_operator       => DBMS_SERVER_ALERT.OPERATOR_GT,
   critical_value          => '0',
   observation_period      => 1,
   consecutive_occurrences => 1,
   instance_name           => NULL,
   object_type             => DBMS_SERVER_ALERT.OBJECT_TYPE_TABLESPACE,
   object_name             => 'USERS');
END;
/
```

Note: When setting nonzero values for percent-full thresholds, use the greater-than-or-equal-to operator, `OPERATOR_GE`.

Restoring a Tablespace to Database Default Thresholds

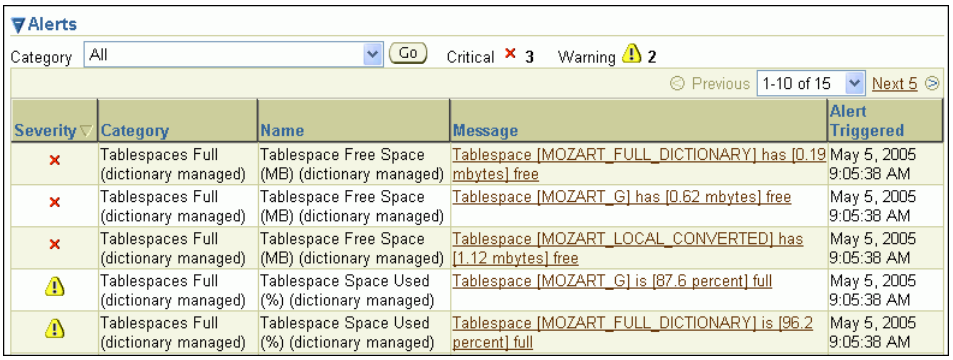
After explicitly setting values for locally managed tablespace alert thresholds, you can cause the values to revert to the database defaults by setting them to `NULL` with `DBMS_SERVER_ALERT.SET_THRESHOLD`.

Modifying Database Default Thresholds

To modify database default thresholds for locally managed tablespaces, invoke `DBMS_SERVER_ALERT.SET_THRESHOLD` as shown in the previous example, but set `object_name` to `NULL`. All tablespaces that use the database default are then switched to the new default.

Viewing Alerts

You view alerts by accessing the home page of Enterprise Manager Database Control.



Alerts

Category All Go Critical 3 Warning 2

Previous 1-10 of 15 Next 5

Severity	Category	Name	Message	Alert Triggered
×	Tablespaces Full (dictionary managed)	Tablespace Free Space (MB) (dictionary managed)	Tablespace [MOZART_FULL_DICTIONARY] has [0.19 mbytes] free	May 5, 2005 9:05:38 AM
×	Tablespaces Full (dictionary managed)	Tablespace Free Space (MB) (dictionary managed)	Tablespace [MOZART_G] has [0.62 mbytes] free	May 5, 2005 9:05:38 AM
×	Tablespaces Full (dictionary managed)	Tablespace Free Space (MB) (dictionary managed)	Tablespace [MOZART_LOCAL_CONVERTED] has [1.12 mbytes] free	May 5, 2005 9:05:38 AM
⚠	Tablespaces Full (dictionary managed)	Tablespace Space Used (%) (dictionary managed)	Tablespace [MOZART_G] is [87.6 percent] full	May 5, 2005 9:05:38 AM
⚠	Tablespaces Full (dictionary managed)	Tablespace Space Used (%) (dictionary managed)	Tablespace [MOZART_FULL_DICTIONARY] is [96.2 percent] full	May 5, 2005 9:05:38 AM

You can also view alerts for locally managed tablespaces with the `DBA_OUTSTANDING_ALERTS` view. See "Server-Generated Alerts Data Dictionary Views" on page 8-6 for more information.

Limitations

Threshold-based alerts have the following limitations:

- Alerts are not issued for locally managed tablespaces that are offline or in read-only mode. However, the database reactivates the alert system for such tablespaces after they become read/write or available.

- When you take a tablespace offline or put it in read-only mode, you should disable the alerts for the tablespace by setting the thresholds to zero. You can then reenable the alerts by resetting the thresholds when the tablespace is once again online and in read/write mode.

See Also:

- "Monitoring Database Operations with Server-Generated Alerts" on page 8-4 for additional information on server-generated alerts in general
- *Oracle Database PL/SQL Packages and Types Reference* for information on the procedures of the `DBMS_SERVER_ALERT` package and how to use them
- "Reclaiming Wasted Space" on page 19-12 for various ways to reclaim space that is no longer being used in the tablespace
- "Purging Objects in the Recycle Bin" on page 20-53 for information on reclaiming recycle bin space

Managing Resumable Space Allocation

Oracle Database provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. Therefore, you can 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
- Operation-Suspended Alert
- 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 space allocation works, and specifically defines qualifying statements and error conditions.

How Resumable Space Allocation Works

The following is an overview of how resumable space allocation works. Details are contained in later sections.

1. A statement executes in a resumable mode only if its session has been enabled for resumable space allocation by one of the following actions:
   - The `RESUMABLE_TIMEOUT` initialization parameter is set to a nonzero value.
   - The `ALTER SESSION ENABLE RESUMABLE` statement is issued.
2. A resumable statement is suspended when one of the following conditions occur (these conditions result in corresponding errors being signalled for non-resumable statements):

- Out of space condition
- Maximum extents reached condition
- Space quota exceeded condition.

3. When the execution of a resumable statement is suspended, 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.
   - The system issues the Resumable Session Suspended alert.
   - 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 the `DBA_` or `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 is resolved (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution and the Resumable Session Suspended alert is cleared.
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?

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 `OCIStmtExecute()` and `OCIStmtFetch()` are candidates.

- DML

  `INSERT`, `UPDATE`, and `DELETE` statements are candidates. The interface used to execute them does not matter; it can be OCI, SQLJ, 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/undo segment/cluster/LOB/table partition/index partition in a tablespace. For example, the following errors fall in this category:

  ```
  ORA-01653 unable to extend table ... in tablespace ...
  ORA-01654 unable to extend index ... in tablespace ...
  ```

- Maximum extents reached condition

  The number of extents in a table/index/temporary 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-01631 max # extents ... reached in table ...
  ORA-01632 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-01536 space quote exceeded for tablespace string
  ```

Resumable Space Allocation and Distributed Operations

In a distributed environment, if a user enables or disables resumable space allocation, or if you, as a DBA, alter the `RESUMABLE_TIMEOUT` initialization parameter, only the local instance is affected. In a distributed transaction, sessions or remote instances are suspended only if `RESUMABLE` has been enabled in the remote instance.

Parallel Execution and Resumable Space Allocation

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 non-correctable 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 the DBA_ or 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. There are two means of enabling and disabling resumable space allocation. You can control it at the system level with the RESUMABLE_TIMEOUT initialization parameter, or users can enable it at the session level using clauses of the ALTER SESSION statement.

> **Note:** Because suspended statements can hold up some system resources, users must be granted the RESUMABLE system privilege before they are allowed to enable resumable space allocation and execute resumable statements.

Setting the RESUMABLE_TIMEOUT Initialization Parameter

You can enable resumable space allocation system wide and specify a timeout interval by setting the RESUMABLE_TIMEOUT initialization parameter. For example, the following setting of the RESUMABLE_TIMEOUT parameter in the initialization parameter file causes all sessions to initially be enabled for resumable space allocation and sets the timeout period to 1 hour:

```
RESUMABLE_TIMEOUT  = 3600
```

If this parameter is set to 0, then resumable space allocation is disabled initially for all sessions. This is the default.

You can use the ALTER SYSTEM SET statement to change the value of this parameter at the system level. For example, the following statement will disable resumable space allocation for all sessions:

```
ALTER SYSTEM SET RESUMABLE_TIMEOUT=0;
```

Within a session, a user can issue the ALTER SESSION SET statement to set the RESUMABLE_TIMEOUT initialization parameter and enable resumable space allocation, change a timeout value, or to disable resumable mode.

Using ALTER SESSION to Enable and Disable Resumable Space Allocation

A user can enable resumable mode for a session, using the following SQL statement:

```
ALTER SESSION ENABLE RESUMABLE;
```

To disable resumable mode, a user issues the following statement:

```
ALTER SESSION DISABLE RESUMABLE;
```

The default for a new session is resumable mode disabled, unless the RESUMABLE_TIMEOUT initialization parameter is set to a nonzero value.

The user can also specify a timeout interval, and can provide a name used to identify a resumable statement. These are discussed separately in following sections.

> **See Also:** "Using a LOGON Trigger to Set Default Resumable Mode" on page 19-8

Specifying a Timeout Interval A timeout period, after which a suspended statement will error if no intervention has taken place, can be specified when resumable mode is enabled. 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 when using the `ENABLE RESUMABLE TIMEOUT` clause to enable resumable mode is 7200 seconds.

> **See Also:** "Setting the RESUMABLE_TIMEOUT Initialization Parameter" on page 19-7 for other methods of changing the timeout interval for resumable space allocation

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.

Using a LOGON Trigger to Set Default Resumable Mode

Another method of setting default resumable mode, other than setting the `RESUMABLE_TIMEOUT` initialization parameter, is that you 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 Database does not guarantee the order of trigger invocation.

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 Database provides alternative methods for notifying users of the error and for providing information about the circumstances.

Notifying Users: The 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 non-resumable 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 terminate suspended statements and modify resumable timeout values. In the following example, the default system timeout is changed by creating a system wide AFTER SUSPEND trigger that calls DBMS_RESUMABLE to set the timeout to 3 hours:

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
   DBMS_RESUMABLE.SET_TIMEOUT(10800);
END;
/
```

> **See Also:** *Oracle Database PL/SQL Language Reference* for information about triggers and system events

Using Views to Obtain Information About Suspended Statements

The following views can be queried to obtain information about the status of resumable statements:

View	Description
DBA_RESUMABLE USER_RESUMABLE	These views contain rows for all currently executing or suspended resumable statements. They can be used by a DBA, AFTER SUSPEND trigger, or another session to monitor the progress of, or obtain specific information about, resumable statements.
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:** *Oracle Database Reference* for specific information about the columns contained in these views

Using the DBMS_RESUMABLE Package

The DBMS_RESUMABLE package helps control resumable space allocation. The following procedures can be invoked:

Procedure	Description
ABORT(sessionID)	This procedure aborts a suspended resumable statement. The parameter sessionID is the session ID in which the statement is executing. For parallel DML/DDL, sessionID is any session ID which participates in the parallel DML/DDL. Oracle Database guarantees that the ABORT operation always succeeds. It may be called either inside or outside of the AFTER SUSPEND trigger. The caller of ABORT must be the owner of the session with sessionID, have ALTER SYSTEM privilege, or have DBA privileges.

Procedure	Description
GET_SESSION_ TIMEOUT(sessionID)	This function returns the current timeout value of resumable space allocation 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 space allocation for the session with sessionID. The parameter timeout is in seconds. The new timeout setting will applies 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 space allocation for the current session. The returned value is in seconds.
SET_TIMEOUT(timeout)	This procedure sets a timeout value for resumable space allocation for the current session. The parameter timeout is in seconds. The new timeout setting applies to the session immediately.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details about the DBMS_RESUMABLE package.

Operation-Suspended Alert

When a resumable session is suspended, an operation-suspended alert is issued on the object that needs allocation of resource for the operation to complete. Once the resource is allocated and the operation completes, the operation-suspended alert is cleared. See "Managing Tablespace Alerts" on page 19-1 for more information on system-generated alerts.

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 an undo 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 undo segments, log error, send email
   -- to DBA, and abort the statement. Otherwise, set timeout to 8 hours.
   --
   -- sys.rbs_error is a table which is to be
   -- created by a DBA manually and defined as
   -- (sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
   -- suspend_time DATE)
   */

   IF OBJECT_TYPE = 'UNDO 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 receipient through UTL_SMTP package
        msg_body:='Subject: Space Error Occurred

                   Space limit reached for undo 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;
```

/

Reclaiming Wasted Space

This section explains how to reclaim wasted space, and also introduces the Segment Advisor, which is the Oracle Database component that identifies segments that have space available for reclamation. The following topics are covered:

- Understanding Reclaimable Unused Space
- Using the Segment Advisor
- Shrinking Database Segments Online
- Deallocating Unused Space

Understanding Reclaimable Unused Space

Over time, updates and deletes on objects within a tablespace can create pockets of empty space that individually are not large enough to be reused for new data. This type of empty space is referred to as fragmented free space.

Objects with fragmented free space can result in much wasted space, and can impact database performance. The preferred way to defragment and reclaim this space is to perform an **online segment shrink**. This process consolidates fragmented free space below the high water mark and compacts the segment. After compaction, the high water mark is moved, resulting in new free space above the high water mark. That space above the high water mark is then deallocated. The segment remains available for queries and DML during most of the operation, and no extra disk space need be allocated.

You use the **Segment Advisor** to identify segments that would benefit from online segment shrink. Only segments in locally managed tablespaces with automatic segment space management (ASSM) are eligible. Other restrictions on segment type exist. For more information, see "Shrinking Database Segments Online" on page 19-26.

If a table with reclaimable space is not eligible for online segment shrink, or if you want to make changes to logical or physical attributes of the table while reclaiming space, then you can use **online table redefinition** as an alternative to segment shrink. Online redefinition is also referred to as **reorganization**. Unlike online segment shrink, it requires extra disk space to be allocated. See "Redefining Tables Online" on page 20-33 for more information.

Using the Segment Advisor

The Segment Advisor identifies segments that have space available for reclamation. It performs its analysis by examining usage and growth statistics in the Automatic Workload Repository (AWR), and by sampling the data in the segment. It is configured to run during maintenance windows as an automated maintenance task, and you can also run it on demand (manually). The Segment Advisor automated maintenance task is known as the Automatic Segment Advisor.

The Segment Advisor generates the following types of advice:

- If the Segment Advisor determines that an object has a significant amount of free space, it recommends online segment shrink. If the object is a table that is not eligible for shrinking, as in the case of a table in a tablespace without automatic segment space management, the Segment Advisor recommends online table redefinition.

- If the Segment Advisor determines that a table could benefit from compression with the OLTP compression method, it makes a recommendation to that effect. (Automatic Segment Advisor only. See "Automatic Segment Advisor" on page 19-13.)
- If the Segment Advisor encounters a table with row chaining above a certain threshold, it records that fact that the table has an excess of chained rows.

> **Note:** The Segment Advisor flags only the type of row chaining that results from updates that increase row length.

If you receive a space management alert, or if you decide that you want to reclaim space, you should start with the Segment Advisor.

To use the Segment Advisor:

1. Check the results of the Automatic Segment Advisor.

   To understand the Automatic Segment Advisor, see "Automatic Segment Advisor", later in this section. For details on how to view results, see "Viewing Segment Advisor Results" on page 19-18.

2. (Optional) Obtain updated results on individual segments by rerunning the Segment Advisor manually.

   See "Running the Segment Advisor Manually", later in this section.

Automatic Segment Advisor

The Automatic Segment Advisor is an automated maintenance task that is configured to run during all maintenance windows.

The Automatic Segment Advisor does not analyze every database object. Instead, it examines database statistics, samples segment data, and then selects the following objects to analyze:

- Tablespaces that have exceeded a critical or warning space threshold
- Segments that have the most activity
- Segments that have the highest growth rate

In addition, the Automatic Segment Advisor evaluates tables that are at least 10MB and that have at least three indexes to determine the amount of space saved if the tables are compressed with the OLTP compression method.

If an object is selected for analysis but the maintenance window expires before the Segment Advisor can process the object, the object is included in the next Automatic Segment Advisor run.

You cannot change the set of tablespaces and segments that the Automatic Segment Advisor selects for analysis. You can, however, enable or disable the Automatic Segment Advisor task, change the times during which the Automatic Segment Advisor is scheduled to run, or adjust automated maintenance task system resource utilization. See "Configuring the Automatic Segment Advisor" on page 19-24 for more information.

See Also:

- "Viewing Segment Advisor Results" on page 19-18
- Chapter 26, "Managing Automated Database Maintenance Tasks"
- "Consider Using Table Compression" on page 20-5 for more information on OLTP compression

Running the Segment Advisor Manually

You can manually run the Segment Advisor at any time with Enterprise Manager or with PL/SQL package procedure calls. Reasons to manually run the Segment Advisor include the following:

- You want to analyze a tablespace or segment that was not selected by the Automatic Segment Advisor.
- You want to repeat the analysis of an individual tablespace or segment to get more up-to-date recommendations.

You can request advice from the Segment Advisor at three levels:

- **Segment level**—Advice is generated for a single segment, such as an unpartitioned table, a partition or subpartition of a partitioned table, an index, or a LOB column.
- **Object level**—Advice is generated for an entire object, such as a table or index. If the object is partitioned, advice is generated on all the partitions of the object. In addition, if you run Segment Advisor manually from Enterprise Manager, you can request advice on the object's dependent objects, such as indexes and LOB segments for a table.
- **Tablespace level**—Advice is generated for every segment in a tablespace.

The `OBJECT_TYPE` column of Table 19–2 on page 19-17 shows the types of objects for which you can request advice.

Running the Segment Advisor Manually with Enterprise Manager You must have the `OEM_ADVISOR` role to run the Segment Advisor manually with Enterprise Manager. There are two ways to run the Segment Advisor:

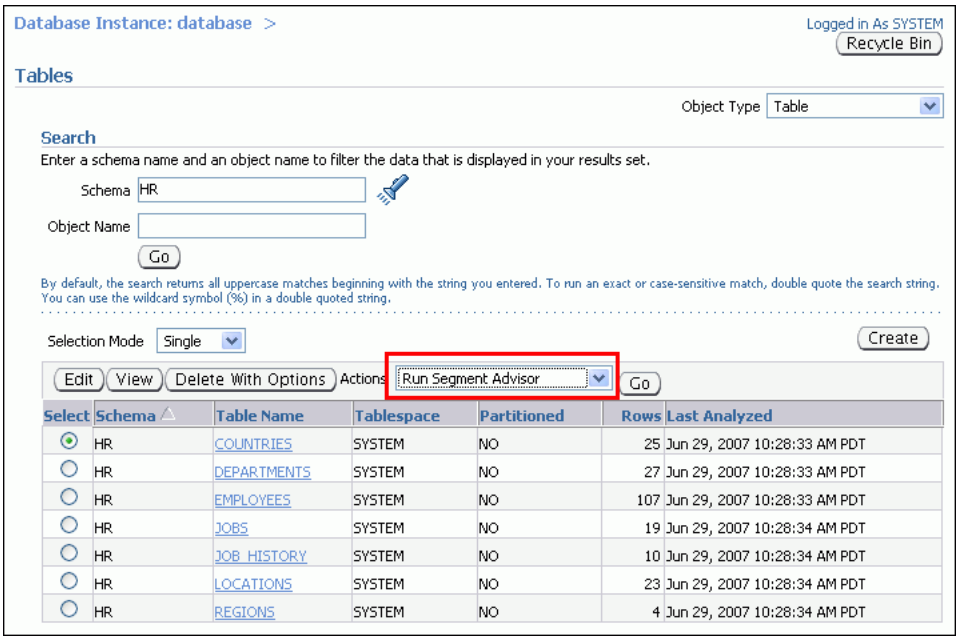
- Using the Segment Advisor Wizard

  This method enables you to request advice at the tablespace level or object level. At the object level, you can request advice on tables, indexes, table partitions, and index partitions. Dependent objects such as LOB segments cannot be included in the analysis.

- Using the Run Segment Advisor command on a schema object page.

  For example, if you display a list of tables on the Tables page (accessible from the Schema page), you can select a table and then select the **Run Segment Advisor** command from the Actions menu.

Figure 19–1 Tables page



This method enables you to include the schema object's dependent objects in the Segment Advisor run. For example, if you select a table and select the **Run Segment Advisor** command, Enterprise Manager displays the table's dependent objects, such as partitions, index segments, LOB segments, and so on. You can then select dependent objects to include in the run.

In both cases, Enterprise Manager creates the Segment Advisor task as an Oracle Database Scheduler job. You can schedule the job to run immediately, or can take advantage of advanced scheduling features offered by the Scheduler.

To run the Segment Advisor manually with the Segment Advisor Wizard:

1. From the database Home page, under Related Links, click **Advisor Central**.

   The Advisor Central page appears. (See Figure 19–2.)

2. Under Advisors, click **Segment Advisor**.

   The first page of the Segment Advisor wizard appears.

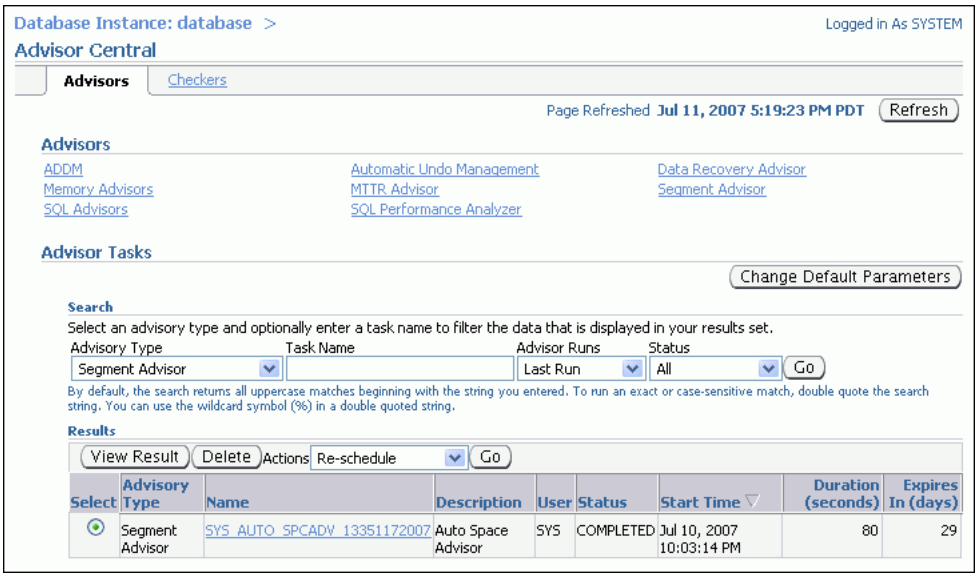
3. Follow the wizard steps to schedule the Segment Advisor job, and then click **Submit** on the final wizard page.

   The Advisor Central page reappears, with the new Segment Advisor job at the top of the list under the Results heading. The job status is `SCHEDULED` or `RUNNING`. (If you do not see your job, use the search fields above the list to display it.)

4. Check the status of the job. If it is not `COMPLETED`, click the **Refresh** button at the top of the page repeatedly. (Do not use your browser's Refresh icon.)

   When the job status changes to `COMPLETED`, select the job by clicking in the **Select** column, and then click **View Result**.

Figure 19–2 Advisor Central page



See Also: Chapter 29, "Scheduling Jobs with Oracle Scheduler" for more information about the advanced scheduling features of the Scheduler.

Running the Segment Advisor Manually with PL/SQL You can also run the Segment Advisor with the `DBMS_ADVISOR` package. You use package procedures to create a Segment Advisor task, set task arguments, and then execute the task. You must have the `ADVISOR` privilege. Table 19–1 shows the procedures that are relevant for the Segment Advisor. See *Oracle Database PL/SQL Packages and Types Reference* for more details on these procedures.

Table 19–1 DBMS_ADVISOR package procedures relevant to the Segment Advisor

Package Procedure Name	Description
`CREATE_TASK`	Use this procedure to create the Segment Advisor task. Specify 'Segment Advisor' as the value of the `ADVISOR_NAME` parameter.
`CREATE_OBJECT`	Use this procedure to identify the target object for segment space advice. The parameter values of this procedure depend upon the object type. Table 19–2 lists the parameter values for each type of object. Note: To request advice on an IOT overflow segment, use an object type of `TABLE`, `TABLE PARTITION`, or `TABLE SUBPARTITION`. Use the following query to find the overflow segment for an IOT and to determine the overflow segment table name to use with `CREATE_OBJECT`: `select table_name, iot_name, iot_type from dba_tables;`
`SET_TASK_PARAMETER`	Use this procedure to describe the segment advice that you need. Table 19–3 shows the relevant input parameters of this procedure. Parameters not listed here are not used by the Segment Advisor.
`EXECUTE_TASK`	Use this procedure to execute the Segment Advisor task.

Table 19–2 Input for DBMS_ADVISOR.CREATE_OBJECT

Input Parameter				
OBJECT_TYPE	**ATTR1**	**ATTR2**	**ATTR3**	**ATTR4**
TABLESPACE	*tablespace name*	NULL	NULL	Unused. Specify NULL.
TABLE	*schema name*	*table name*	NULL	Unused. Specify NULL.
INDEX	*schema name*	*index name*	NULL	Unused. Specify NULL.
TABLE PARTITION	*schema name*	*table name*	*table partition name*	Unused. Specify NULL.
INDEX PARTITION	*schema name*	*index name*	*index partition name*	Unused. Specify NULL.
TABLE SUBPARTITION	*schema name*	*table name*	*table subpartition name*	Unused. Specify NULL.
INDEX SUBPARTITION	*schema name*	*index name*	*index subpartition name*	Unused. Specify NULL.
LOB	*schema name*	*segment name*	NULL	Unused. Specify NULL.
LOB PARTITION	*schema name*	*segment name*	*lob partition name*	Unused. Specify NULL.
LOB SUBPARTITION	*schema name*	*segment name*	*lob subpartition name*	Unused. Specify NULL.

Table 19–3 Input for DBMS_ADVISOR.SET_TASK_PARAMETER

Input Parameter	Description	Possible Values	Default Value
time_limit	The time limit for the Segment Advisor run, specified in seconds.	Any number of seconds	UNLIMITED
recommend_all	Whether the Segment Advisor should generate findings for all segments.	TRUE: Findings are generated on all segments specified, whether or not space reclamation is recommended. FALSE: Findings are generated only for those objects that generate recommendations for space reclamation.	TRUE

Example The example that follows shows how to use the DBMS_ADVISOR procedures to run the Segment Advisor for the sample table hr.employees. The user executing these package procedures must have the EXECUTE object privilege on the package or the ADVISOR system privilege.

Note that passing an object type of TABLE to DBMS_ADVISOR.CREATE_OBJECT amounts to an object level request. If the table is not partitioned, the table segment is analyzed (without any dependent segments like index or LOB segments). If the table is partitioned, the Segment Advisor analyzes all table partitions and generates separate findings and recommendations for each.

```
variable id number;
begin
  declare
  name varchar2(100);
  descr varchar2(500);
  obj_id number;
  begin
  name:='Manual_Employees';
  descr:='Segment Advisor Example';

  dbms_advisor.create_task (
    advisor_name     => 'Segment Advisor',
    task_id          => :id,
```

```
    task_name        => name,
    task_desc        => descr);

  dbms_advisor.create_object (
    task_name        => name,
    object_type      => 'TABLE',
    attr1            => 'HR',
    attr2            => 'EMPLOYEES',
    attr3            => NULL,
    attr4            => NULL,
    attr5            => NULL,
    object_id        => obj_id);

  dbms_advisor.set_task_parameter(
    task_name        => name,
    parameter        => 'recommend_all',
    value            => 'TRUE');

  dbms_advisor.execute_task(name);
  end;
end;
/
```

Viewing Segment Advisor Results

The Segment Advisor creates several types of results: recommendations, findings, actions, and objects. You can view results in the following ways:

- With Enterprise Manager
- By querying the `DBA_ADVISOR_*` views
- By calling the `DBMS_SPACE.ASA_RECOMMENDATIONS` procedure

Table Table 19–4 describes the various result types and their associated `DBA_ADVISOR_*` views.

Table 19–4 Segment Advisor Result Types

Result Type	Associated View	Description
Recommendations	DBA_ADVISOR_RECOMMENDATIONS	If a segment would benefit from a segment shrink, reorganization, or compression, the Segment Advisor generates a recommendation for the segment. Table 19–5 shows examples of generated findings and recommendations.
Findings	DBA_ADVISOR_FINDINGS	Findings are a report of what the Segment Advisor observed in analyzed segments. Findings include space used and free space statistics for each analyzed segment. Not all findings result in a recommendation. (There may be only a few recommendations, but there could be many findings.) When running the Segment Advisor manually with PL/SQL, if you specify 'TRUE' for recommend_all in the SET_TASK_PARAMETER procedure, then the Segment Advisor generates a finding for each segment that qualifies for analysis, whether or not a recommendation is made for that segment. For row chaining advice, the Automatic Segment Advisor generates findings only, and not recommendations. If the Automatic Segment Advisor has no space reclamation recommendations to make, it does not generate findings. However, the Automatic Segment Advisor may generate findings for tables that could benefit from OLTP compression.
Actions	DBA_ADVISOR_ACTIONS	Every recommendation is associated with a suggested action to perform: either segment shrink, online redefinition (reorganization), or compression. The DBA_ADVISOR_ACTIONS view provides either the SQL that you can use to perform a segment shrink or table compression, or a suggestion to reorganize the object.
Objects	DBA_ADVISOR_OBJECTS	All findings, recommendations, and actions are associated with an object. If the Segment Advisor analyzes multiple segments, as with a tablespace or partitioned table, then one entry is created in the DBA_ADVISOR_OBJECTS view for each analyzed segment. Table 19–2 defines the columns in this view to query for information on the analyzed segments. You can correlate the objects in this view with the objects in the findings, recommendations, and actions views.

See Also:

- *Oracle Database Reference* for details on the DBA_ADVISOR_* views.
- *Oracle Database PL/SQL Packages and Types Reference* for details on the DBMS_SPACE.ASA_RECOMMENDATIONS function.

Viewing Segment Advisor Results with Enterprise Manager

With Oracle Enterprise Manager, you can view Segment Advisor results for both Automatic Segment Advisor runs and manual Segment Advisor runs. You can view the following types of results:

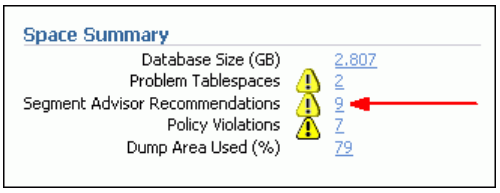
- All recommendations (multiple automatic and manual Segment Advisor runs)
- Recommendations from the last Automatic Segment Advisor run

- Recommendations from a specific run
- Row chaining findings

You can also view a list of the segments that were analyzed by the last Automatic Segment Advisor run.

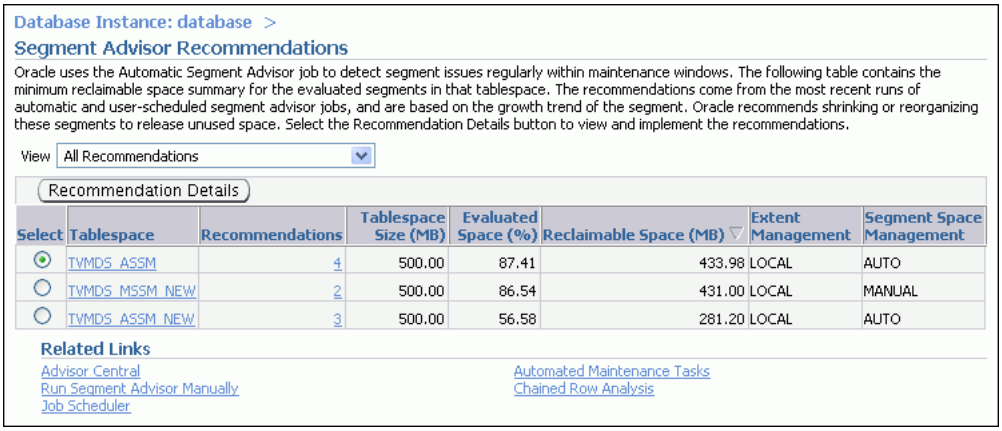
To view Segment Advisor results with Enterprise Manager—All runs:

1. On the database Home page, under the Space Summary heading, click the numeric link next to the title **Segment Advisor Recommendations**.



   The Segment Advisor Recommendations page appears. Recommendations are organized by tablespace.

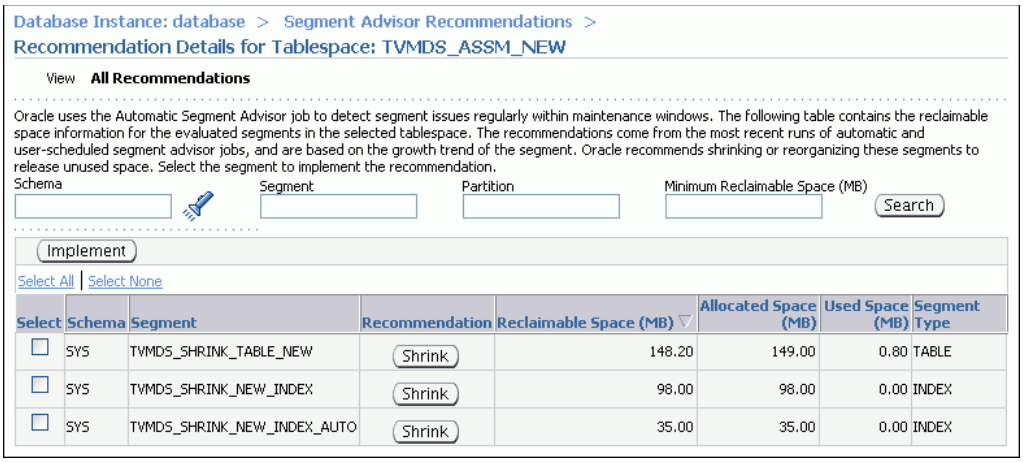
Figure 19–3 Segment Advisor Recommendations page



2. If any recommendations are present, select a tablespace, and then click **Recommendation Details**.

   The Recommendation Details page appears. You can initiate the recommended activity from this page (shrink or reorganize).

Figure 19–4 Recommendation Details page



Tip: The list entries are sorted in descending order by reclaimable space. You can click column headings to change the sort order or to change from ascending to descending order.

To view Segment Advisor results with Enterprise Manager—Last Automatic Segment Advisor run:

1. On the database Home page, under the Space Summary heading, click the numeric link next to the title **Segment Advisor Recommendations**.

   The Segment Advisor Recommendations page appears. (See Figure 19–3.)

2. In the View list, select **Recommendations from Last Automatic Run**.

3. If any recommendations are present, select a tablespace and click **Recommendation Details**.

   The Recommendation Details page appears. (See Figure 19–4.) You can initiate the recommended activity from this page (shrink or reorganize).

To view Segment Advisor results with Enterprise Manager—Specific run:

1. Start at the Advisor Central page.

   If you ran the Segment Advisor with the Enterprise Manager wizard, the Advisor Central page appears after you submit the Segment Advisor task. Otherwise, to get to this page, on the Database Home page, under Related Links, click **Advisor Central**.

2. Check that your task appears in the list under the Results heading. If it does not, complete these steps (See Figure 19–2 on page 19-16):

   a. In the Search section of the page, under Advisor Tasks, select **Segment Advisor** in the Advisory Type list.

   b. In the Advisor Runs list, select **All** or the desired time period.

   c. (Optional) Enter a task name.

   d. Click **Go**.

   Your Segment Advisor task appears in the Results section.

3. Check the status of the job. If it is not COMPLETED, click the **Refresh** button at the top of the page until your task status shows COMPLETED. (Do not use your browser's refresh icon.)
4. Click the task name.

   The Segment Advisor Task page appears, with recommendations organized by tablespace.
5. Select a tablespace in the list, and then click **Recommendation Details**.

   The Recommendation Details page appears. (See Figure 19–4.) You can initiate the recommended activity from this page (shrink or reorganize).

To view row chaining findings:

1. On the database Home page, under the Space Summary heading, click the numeric link next to the title **Segment Advisor Recommendations**.

   The Segment Advisor Recommendations page appears. (See Figure 19–3.)
2. Under the Related Links heading, click **Chained Row Analysis**.

   The Chained Row Analysis page appears, showing all segments that have chained rows, with a chained rows percentage for each.

Viewing Segment Advisor Results by Querying the DBA_ADVISOR_* Views

The headings of Table 19–5 show the columns in the DBA_ADVISOR_* views that contain output from the Segment Advisor. See *Oracle Database Reference* for a description of these views. The table contents summarize the possible outcomes. In addition, Table 19–2 on page 19-17 defines the columns in the DBA_ADVISOR_OBJECTS view that contain information on the analyzed segments.

Before querying the DBA_ADVISOR_* views, you can check that the Segment Advisor task is complete by querying the STATUS column in DBA_ADVISOR_TASKS.

```
select task_name, status from dba_advisor_tasks
   where owner = 'STEVE' and advisor_name = 'Segment Advisor';

TASK_NAME                      STATUS
------------------------------ -----------
Manual Employees               COMPLETED
```

The following example shows how to query the DBA_ADVISOR_* views to retrieve findings from all Segment Advisor runs submitted by user STEVE:

```
select af.task_name, ao.attr2 segname, ao.attr3 partition, ao.type, af.message
  from dba_advisor_findings af, dba_advisor_objects ao
  where ao.task_id = af.task_id
  and ao.object_id = af.object_id
  and ao.owner = 'STEVE';


TASK_NAME          SEGNAME      PARTITION       TYPE             MESSAGE
------------------ ------------ --------------- ---------------- --------------------------
Manual_Employees   EMPLOYEES                    TABLE            The free space in the obje
                                                                 ct is less than 10MB.

Manual_Salestable4 SALESTABLE4  SALESTABLE4_P1  TABLE PARTITION  Perform shrink, estimated
                                                                 savings is 74444154 bytes.
```

```
Manual_Salestable4 SALESTABLE4  SALESTABLE4_P2  TABLE PARTITION  The free space in the obje
                                                                 ct is less than 10MB.
```

Table 19–5 Segment Advisor Outcomes: Summary

MESSAGE column of DBA_ADVISOR_FINDINGS	MORE_INFO column of DBA_ADVISOR_FINDINGS	BENEFIT_TYPE column of DBA_ADVISOR_RECOMMENDATIONS	ATTR1 column of DBA_ADVISOR_ACTIONS
Insufficient information to make a recommendation.	-	-	-
The free space in the object is less than 10MB.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-
The object has some free space but cannot be shrunk because...	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-
The free space in the object is less than the size of the last extent.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	-	-
Perform shrink, estimated savings is *xxx* bytes.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Perform shrink, estimated savings is *xxx* bytes.	The command to execute. For example: `ALTER` *`object`* `SHRINK SPACE`;)
Enable row movement of the table *schema.table* and perform shrink, estimated savings is *xxx* bytes.	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Enable row movement of the table *schema.table* and perform shrink, estimated savings is *xxx* bytes	The command to execute. For example: `ALTER` *`object`* `SHRINK SPACE`;)
Perform re-org on the object *object*, estimated savings is *xxx* bytes. (Note: This finding is for objects with reclaimable space that are not eligible for online segment shrink.)	Allocated Space:*xxx*: Used Space:*xxx*: Reclaimable Space :*xxx*	Perform re-org on the object *object*, estimated savings is *xxx* bytes.	Perform reorg
The object has chained rows that can be removed by re-org.	*xx* percent chained rows can be removed by re-org.	-	-
Compress object *object_name*, estimated savings is *xxx* bytes. (This outcome is generated by the Automatic Segment Advisor only)	Compress object *object_name*, estimated savings is *xxx* bytes.	-	The command to execute. For example: `ALTER TABLE T1 COMPRESS FOR OLTP` For this finding, see also the `ATTR2` column of `DBA_ADVISOR_ACTIONS`.

Viewing Segment Advisor Results with DBMS_SPACE.ASA_RECOMMENDATIONS

The `ASA_RECOMMENDATIONS` procedure in the `DBMS_SPACE` package returns a nested table object that contains findings or recommendations for Automatic Segment Advisor runs and, optionally, manual Segment Advisor runs. Calling this procedure may be easier than working with the `DBA_ADVISOR_*` views, because the procedure performs all the required joins for you and returns information in an easily consumable format.

The following query returns recommendations by the most recent run of the Auto Segment Advisor, with the suggested command to run to follow the recommendations:

```
select tablespace_name, segment_name, segment_type, partition_name,
recommendations, c1 from
table(dbms_space.asa_recommendations('FALSE', 'FALSE', 'FALSE'));
```

```
TABLESPACE_NAME                SEGMENT_NAME                   SEGMENT_TYPE
------------------------------ ------------------------------ --------------
PARTITION_NAME
------------------------------
RECOMMENDATIONS
--------------------------------------------------------------------------------
C1
--------------------------------------------------------------------------------
TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P2
Perform shrink, estimated savings is 57666422 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P2" shrink space

TVMDS_ASSM                     ORDERS1                        TABLE PARTITION
ORDERS1_P1
Perform shrink, estimated savings is 45083514 bytes.
alter table "STEVE"."ORDERS1" modify partition "ORDERS1_P1" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW                     TABLE

Perform shrink, estimated savings is 155398992 bytes.
alter table "STEVE"."ORDERS_NEW" shrink space

TVMDS_ASSM_NEW                 ORDERS_NEW_INDEX               INDEX

Perform shrink, estimated savings is 102759445 bytes.
alter index "STEVE"."ORDERS_NEW_INDEX" shrink space
```

See *Oracle Database PL/SQL Packages and Types Reference* for details on `DBMS_SPACE.ASA_RECOMMENDATIONS`.

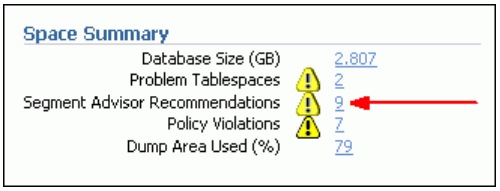
Configuring the Automatic Segment Advisor

The Automatic Segment Advisor is an automated maintenance task. As such, you can use Enterprise Manager or PL/SQL package procedure calls to modify when (and if) this task runs. You can also control the resources allotted to it by modifying the appropriate resource plans.

You can call PL/SQL package procedures to make these changes, but the easier way to is to use Enterprise Manager.

To configure the Automatic Segment Advisor task with Enterprise Manager:

1. Log in to Enterprise Manager as user `SYSTEM`.
2. On the Database Home page, under the Space Summary heading, click the numeric link next to the label **Segment Advisor Recommendations**.



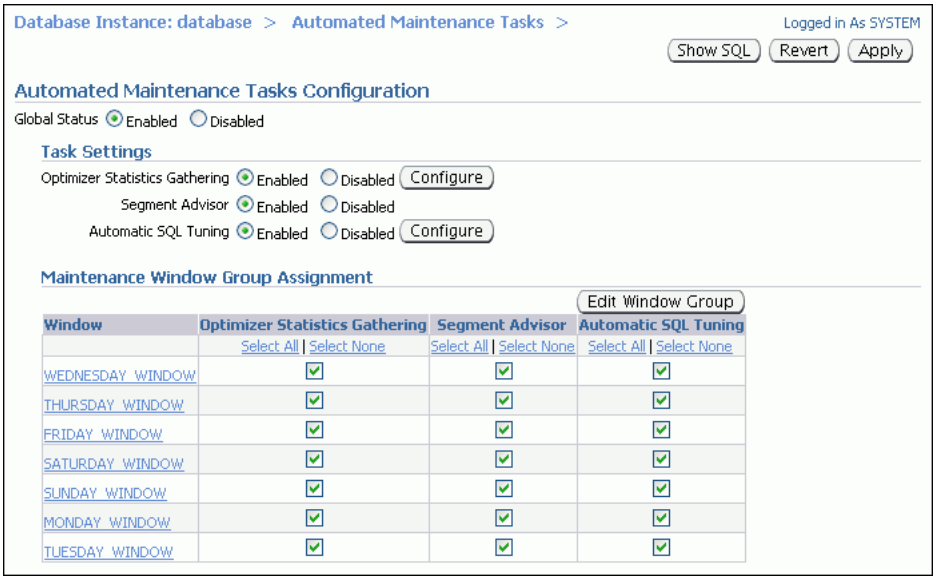
   The Segment Advisor Recommendations page appears.

3. Under the Related Links heading, click the link entitled **Automated Maintenance Tasks**.

   The Automated Maintenance Tasks page appears.

4. Click **Configure**.

   The Automated Maintenance Tasks Configuration page appears.



5. To completely disable the Automatic Segment Advisor, under Task Settings, select **Disabled** next to the Segment Advisor label, and then click **Apply**.

6. To disable the Automatic Segment Advisor for specific maintenance windows, clear the desired check boxes under the Segment Advisor column, and then click **Apply**.

7. To modify the start and end times and durations of maintenance windows, click **Edit Window Group**.

   The Edit Window Group page appears. Click the name of a maintenance window, and then click **Edit** to change the window's schedule.

   See Also:

   - Chapter 26, "Managing Automated Database Maintenance Tasks"

Viewing Automatic Segment Advisor Information

The following views display information specific to the Automatic Segment Advisor. For details, see *Oracle Database Reference*.

View	Description
DBA_AUTO_SEGADV_SUMMARY	Each row of this view summarizes one Automatic Segment Advisor run. Fields include number of tablespaces and segments processed, and number of recommendations made.
DBA_AUTO_SEGADV_CTL	Contains control information that the Automatic Segment Advisor uses to select and process segments. Each row contains information on a single object (tablespace or segment), including whether the object has been processed, and if so, the task ID under which it was processed and the reason for selecting it.

Shrinking Database Segments Online

You use online segment shrink to reclaim fragmented free space below the high water mark in an Oracle Database segment. The benefits of segment shrink are these:

- Compaction of data leads to better cache utilization, which in turn leads to better online transaction processing (OLTP) performance.
- The compacted data requires fewer blocks to be scanned in full table scans, which in turns leads to better decision support system (DSS) performance.

Segment shrink is an online, in-place operation. DML operations and queries can be issued during the data movement phase of segment shrink. Concurrent DML operations are blocked for a short time at the end of the shrink operation, when the space is deallocated. Indexes are maintained during the shrink operation and remain usable after the operation is complete. Segment shrink does not require extra disk space to be allocated.

Segment shrink reclaims unused space both above and below the high water mark. In contrast, space deallocation reclaims unused space only above the high water mark. In shrink operations, by default, the database compacts the segment, adjusts the high water mark, and releases the reclaimed space.

Segment shrink requires that rows be moved to new locations. Therefore, you must first enable row movement in the object you want to shrink and disable any rowid-based triggers defined on the object. You enable row movement in a table with the `ALTER TABLE ... ENABLE ROW MOVEMENT` command.

Shrink operations can be performed only on segments in locally managed tablespaces with automatic segment space management (ASSM). Within an ASSM tablespace, all segment types are eligible for online segment shrink except these:

- IOT mapping tables
- Tables with rowid based materialized views
- Tables with function-based indexes
- `SECUREFILE` LOBs
- Compressed tables

Note: Shrinking database segments online might cause dependent database objects to become invalid. See "About Object Dependencies and Object Invalidation" on page 18-17.

See Also: *Oracle Database SQL Language Reference* for more information on the `ALTER TABLE` command.

Invoking Online Segment Shrink

Before invoking online segment shrink, view the findings and recommendations of the Segment Advisor. For more information, see "Using the Segment Advisor" on page 19-12.

You invoke online segment shrink with Oracle Enterprise Manager or with SQL commands in SQL*Plus. The remainder of this section discusses the command line method.

> **Note:** You can invoke segment shrink directly from the Recommendation Details page in Enterprise Manager. (See Figure 19–4.) Or, to invoke segment shrink for an individual table in Enterprise Manager, display the table on the Tables page, select the table, and then click **Shrink Segment** in the Actions list. (See Figure 19–1.) Perform a similar operation in Enterprise Manager to shrink indexes, materialized views, and so on.

You can shrink space in a table, index-organized table, index, partition, subpartition, materialized view, or materialized view log. You do this using `ALTER TABLE`, `ALTER INDEX`, `ALTER MATERIALIZED VIEW`, or `ALTER MATERIALIZED VIEW LOG` statement with the `SHRINK SPACE` clause.

Two optional clauses let you control how the shrink operation proceeds:

- The `COMPACT` clause lets you divide the shrink segment operation into two phases. When you specify `COMPACT`, Oracle Database defragments the segment space and compacts the table rows but postpones the resetting of the high water mark and the deallocation of the space until a future time. This option is useful if you have long-running queries that might span the operation and attempt to read from blocks that have been reclaimed. The defragmentation and compaction results are saved to disk, so the data movement does not have to be redone during the second phase. You can reissue the `SHRINK SPACE` clause without the `COMPACT` clause during off-peak hours to complete the second phase.
- The `CASCADE` clause extends the segment shrink operation to all dependent segments of the object. For example, if you specify `CASCADE` when shrinking a table segment, all indexes of the table will also be shrunk. (You need not specify `CASCADE` to shrink the partitions of a partitioned table.) To see a list of dependent segments of a given object, you can run the `OBJECT_DEPENDENT_SEGMENTS` procedure of the `DBMS_SPACE` package.

As with other DDL operations, segment shrink causes subsequent SQL statements to be reparsed because of invalidation of cursors unless you specify the `COMPACT` clause.

Examples

Shrink a table and all of its dependent segments (including `BASICFILE` LOB segments):

```
ALTER TABLE employees SHRINK SPACE CASCADE;
```

Shrink a `BASICFILE` LOB segment only:

```
ALTER TABLE employees MODIFY LOB (perf_review) (SHRINK SPACE);
```

Shrink a single partition of a partitioned table:

```
ALTER TABLE customers MODIFY PARTITION cust_P1 SHRINK SPACE;
```

Shrink an IOT index segment and the overflow segment:

```
ALTER TABLE cities SHRINK SPACE CASCADE;
```

Shrink an IOT overflow segment only:

```
ALTER TABLE cities OVERFLOW SHRINK SPACE;
```

See Also:

- *Oracle Database SQL Language Reference* for the syntax and restrictions of the `ALTER TABLE`, `ALTER INDEX`, `ALTER MATERIALIZED VIEW`, and `ALTER MATERIALIZED VIEW LOG` statements with the `SHRINK SPACE` clause
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOB segments

Deallocating Unused Space

When you deallocate unused space, the database frees the unused space at the unused (high water mark) end of the database segment and makes the space available for other segments in the tablespace.

Prior to deallocation, you can run the `UNUSED_SPACE` procedure of the `DBMS_SPACE` package, which returns information about the position of the high water mark and the amount of unused space in a segment. For segments in locally managed tablespaces with automatic segment space management, use the `SPACE_USAGE` procedure for more accurate information on unused space.

See Also: *Oracle Database PL/SQL Packages and Types Reference* contains the description of the `DBMS_SPACE` package

The following statements deallocate unused space in a segment (table, index or cluster):

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

The `KEEP` clause is optional and lets you specify the amount of space retained in the segment. You can verify that the deallocated space is freed by examining the `DBA_FREE_SPACE` view.

See Also:

- *Oracle Database SQL Language Reference* for details on the syntax and semantics of deallocating unused space
- *Oracle Database Reference* for more information about the `DBA_FREE_SPACE` view

Dropping Unused Object Storage

Beginning with Oracle Database 11*g* release 2 (11.2.0.2), the `DBMS_SPACE_ADMIN` package includes the `DROP_EMPTY_SEGMENTS` procedure, which enables you to drop segments for empty tables and partitions that have been migrated from previous releases. This includes segments of dependent objects of the table, such as index segments, where possible.

The following example drops empty segments from every table in the database.

```
BEGIN
  DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS();
END;
```

The following drops empty segments from the `HR.EMPLOYEES` table, including dependent objects.

```
BEGIN
  DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS(
    schema_name  => 'HR',
    table_name   => 'EMPLOYEES');
END;
```

This procedure requires 11.2 compatibility level.

> **See Also:** See *Oracle Database PL/SQL Packages and Types Reference* for details about this procedure

Understanding Space Usage of Data Types

When creating tables and other data structures, you must know how much space they will require. Each data type has different space requirements. The *Oracle Database PL/SQL Language Reference* and *Oracle Database SQL Language Reference* contain extensive descriptions of data types and their space requirements.

Displaying Information About Space Usage for Schema Objects

Oracle Database provides data dictionary views and PL/SQL packages that allow you to display information about the space usage of 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 Object Space Usage

These DBMS_SPACE subprograms provide information about schema objects:

Package and Procedure/Function	Description
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.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for a description of the DBMS_SPACE package

Example: 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> VARIABLE lastusedblock 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
```

Schema Objects Space Usage Data Dictionary Views

These views display information about space usage in schema objects:

View	Description
DBA_SEGMENTS USER_SEGMENTS	DBA view describes storage allocated for all database segments. User view describes storage allocated for segments for the current user.
DBA_EXTENTS USER_EXTENTS	DBA view describes extents comprising all segments in the database. User view describes extents comprising segments for the current user.
DBA_FREE_SPACE USER_FREE_SPACE	DBA view lists free extents in all tablespaces. User view shows free space information for tablespaces for which the user has quota.

The following sections contain examples of using some of these views.

> **See Also:** *Oracle Database Reference* for a complete description of data dictionary views

Example 1: Displaying Segment Information

The following query returns the name and size of each index segment in schema `hr`:

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
    FROM DBA_SEGMENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              TABLESPACE_NAME    BYTES BLOCKS EXTENTS
------------------------- --------------- -------- ------ -------
COUNTRY_C_ID_PK           EXAMPLE            65536     32       1
DEPT_ID_PK                EXAMPLE            65536     32       1
DEPT_LOCATION_IX          EXAMPLE            65536     32       1
```

```
EMP_DEPARTMENT_IX         EXAMPLE            65536     32       1
EMP_EMAIL_UK              EXAMPLE            65536     32       1
EMP_EMP_ID_PK             EXAMPLE            65536     32       1
EMP_JOB_IX                EXAMPLE            65536     32       1
EMP_MANAGER_IX            EXAMPLE            65536     32       1
EMP_NAME_IX               EXAMPLE            65536     32       1
JHIST_DEPARTMENT_IX       EXAMPLE            65536     32       1
JHIST_EMPLOYEE_IX         EXAMPLE            65536     32       1
JHIST_EMP_ID_ST_DATE_PK   EXAMPLE            65536     32       1
JHIST_JOB_IX              EXAMPLE            65536     32       1
JOB_ID_PK                 EXAMPLE            65536     32       1
LOC_CITY_IX               EXAMPLE            65536     32       1
LOC_COUNTRY_IX            EXAMPLE            65536     32       1
LOC_ID_PK                 EXAMPLE            65536     32       1
LOC_STATE_PROVINCE_IX     EXAMPLE            65536     32       1
REG_ID_PK                 EXAMPLE            65536     32       1

19 rows selected.
```

Example 2: Displaying Extent Information

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 allocated to each index segment in the `hr` schema and the size of each of those extents:

```
SELECT SEGMENT_NAME, SEGMENT_TYPE, TABLESPACE_NAME, EXTENT_ID, BYTES, BLOCKS
    FROM DBA_EXTENTS
    WHERE SEGMENT_TYPE = 'INDEX'
    AND OWNER='HR'
    ORDER BY SEGMENT_NAME;
```

The query output is:

```
SEGMENT_NAME              SEGMENT_TYPE TABLESPACE_NAME EXTENT_ID    BYTES BLOCKS
------------------------- ------------ --------------- --------- -------- ------
COUNTRY_C_ID_PK           INDEX        EXAMPLE                 0    65536     32
DEPT_ID_PK                INDEX        EXAMPLE                 0    65536     32
DEPT_LOCATION_IX          INDEX        EXAMPLE                 0    65536     32
EMP_DEPARTMENT_IX         INDEX        EXAMPLE                 0    65536     32
EMP_EMAIL_UK              INDEX        EXAMPLE                 0    65536     32
EMP_EMP_ID_PK             INDEX        EXAMPLE                 0    65536     32
EMP_JOB_IX                INDEX        EXAMPLE                 0    65536     32
EMP_MANAGER_IX            INDEX        EXAMPLE                 0    65536     32
EMP_NAME_IX               INDEX        EXAMPLE                 0    65536     32
JHIST_DEPARTMENT_IX       INDEX        EXAMPLE                 0    65536     32
JHIST_EMPLOYEE_IX         INDEX        EXAMPLE                 0    65536     32
JHIST_EMP_ID_ST_DATE_PK   INDEX        EXAMPLE                 0    65536     32
JHIST_JOB_IX              INDEX        EXAMPLE                 0    65536     32
JOB_ID_PK                 INDEX        EXAMPLE                 0    65536     32
LOC_CITY_IX               INDEX        EXAMPLE                 0    65536     32
LOC_COUNTRY_IX            INDEX        EXAMPLE                 0    65536     32
LOC_ID_PK                 INDEX        EXAMPLE                 0    65536     32
LOC_STATE_PROVINCE_IX     INDEX        EXAMPLE                 0    65536     32
REG_ID_PK                 INDEX        EXAMPLE                 0    65536     32

19 rows selected.
```

For the `hr` schema, no segment has multiple extents allocated to it.

Example 3: Displaying the Free Space (Extents) in a Tablespace

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 the SMUNDO tablespace:

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
    FROM DBA_FREE_SPACE
    WHERE TABLESPACE_NAME='SMUNDO';
```

The query output is:

```
TABLESPACE_NAME  FILE_ID    BYTES BLOCKS
--------------- -------- -------- ------
SMUNDO                3    65536     32
SMUNDO                3    65536     32
SMUNDO                3    65536     32
SMUNDO                3    65536     32
SMUNDO                3    65536     32
SMUNDO                3    65536     32
SMUNDO                3   131072     64
SMUNDO                3   131072     64
SMUNDO                3    65536     32
SMUNDO                3  3407872   1664

10 rows selected.
```

Capacity Planning for Database Objects

Oracle Database provides two ways to plan capacity for database objects:

- With Enterprise Manager
- With the DBMS_SPACE PL/SQL package

This section discusses the PL/SQL method. See Enterprise Manager online help and *Oracle Database 2 Day DBA* for details on capacity planning with Enterprise Manager.

Three procedures in the DBMS_SPACE package enable you to predict the size of new objects and monitor the size of existing database objects. This section discusses those procedures and contains the following sections:

- Estimating the Space Use of a Table
- Estimating the Space Use of an Index
- Obtaining Object Growth Trends

Estimating the Space Use of a Table

The size of a database table can vary greatly depending on tablespace storage attributes, tablespace block size, and many other factors. The CREATE_TABLE_COST procedure of the DBMS_SPACE package lets you estimate the space use cost of creating a table. See *Oracle Database PL/SQL Packages and Types Reference* for details on the parameters of this procedure.

The procedure has two variants. The first variant uses average row size to estimate size. The second variant uses column information to estimate table size. Both variants require as input the following values:

- TABLESPACE_NAME: The tablespace in which the object will be created. The default is the SYSTEM tablespace.

- ROW_COUNT: The anticipated number of rows in the table.
- PCT_FREE: The percentage of free space you want to reserve in each block for future expansion of existing rows due to updates.

In addition, the first variant also requires as input a value for AVG_ROW_SIZE, which is the anticipated average row size in bytes.

The second variant also requires for each anticipated column values for COLINFOS, which is an object type comprising the attributes COL_TYPE (the data type of the column) and COL_SIZE (the number of characters or bytes in the column).

The procedure returns two values:

- USED_BYTES: The actual bytes used by the data, including overhead for block metadata, PCT_FREE space, and so forth.
- ALLOC_BYTES: The amount of space anticipated to be allocated for the object taking into account the tablespace extent characteristics.

Note: Beginning with release 11.2.0.2 of Oracle Database, the default size of the first extent of any new segment for a partitioned table is 8 MB instead of 64 KB. This helps improve performance of inserts and queries on partitioned tables. Although partitioned tables will start with a larger initial size, once sufficient data is inserted, the space consumption will be the same as in previous releases. You can override this default by setting the INITIAL size in the storage clause for the table. This new default only applies to table partitions and LOB partitions.

Estimating the Space Use of an Index

The CREATE_INDEX_COST procedure of the DBMS_SPACE package lets you estimate the space use cost of creating an index on an existing table.

The procedure requires as input the following values:

- DDL: The CREATE INDEX statement that would create the index. The table specified in this DDL statement must be an existing table.
- [Optional] PLAN_TABLE: The name of the plan table to use. The default is NULL.

The results returned by this procedure depend on statistics gathered on the segment. Therefore, be sure to obtain statistics shortly before executing this procedure. In the absence of recent statistics, the procedure does not issue an error, but it may return inappropriate results. The procedure returns the following values:

- USED_BYTES: The number of bytes representing the actual index data.
- ALLOC_BYTES: The amount of space allocated for the index in the tablespace.

Obtaining Object Growth Trends

The OBJECT_GROWTH_TREND function of the DBMS_SPACE package produces a table of one or more rows, where each row describes the space use of the object at a specific time. The function retrieves the space use totals from the Automatic Workload Repository or computes current space use and combines it with historic space use changes retrieved from Automatic Workload Repository. See *Oracle Database PL/SQL Packages and Types Reference* for detailed information on the parameters of this function.

The function requires as input the following values:

- OBJECT_OWNER: The owner of the object.
- OBJECT_NAME: The name of the object.
- PARTITION_NAME: The name of the table or index partition, is relevant. Specify NULL otherwise.
- OBJECT_TYPE: The type of the object.
- START_TIME: A TIMESTAMP value indicating the beginning of the growth trend analysis.
- END_TIME: A TIMESTAMP value indicating the end of the growth trend analysis. The default is "NOW".
- INTERVAL: The length in minutes of the reporting interval during which the function should retrieve space use information.
- SKIP_INTERPOLATED: Determines whether the function should omit values based on recorded statistics before and after the INTERVAL ('YES') or not ('NO'). This setting is useful when the result table will be displayed as a table rather than a chart, because you can see more clearly how the actual recording interval relates to the requested reporting interval.

The function returns a table, each of row of which provides space use information on the object for one interval. If the return table is very large, the results are pipelined so that another application can consume the information as it is being produced. The output table has the following columns:

- TIMEPOINT: A TIMESTAMP value indicating the time of the reporting interval.

  Records are not produced for values of TIME that precede the oldest recorded statistics for the object.
- SPACE_USAGE: The number of bytes actually being used by the object data.
- SPACE_ALLOC: The number of bytes allocated to the object in the tablespace at that time.
- QUALITY: A value indicating how well the requested reporting interval matches the actual recording of statistics. This information is useful because there is no guaranteed reporting interval for object size use statistics, and the actual reporting interval varies over time and from object to object.

  The values of the QUALITY column are:

  - GOOD: The value whenever the value of TIME is based on recorded statistics with a recorded timestamp within 10% of the INTERVAL specified in the input parameters.
  - INTERPOLATED: The value did not meet the criteria for GOOD, but was based on recorded statistics before and after the value of TIME. Current in-memory statistics can be collected across all instances in a cluster and treated as the "recorded" value for the present time.
  - PROJECTION: The value of TIME is in the future as of the time the table was produced. In an Oracle Real Application Clusters environment, the rules for recording statistics allow each instance to choose independently which objects will be selected.

  The output returned by this function is an aggregation of values recorded across all instances in an Oracle RAC environment. Each value can be computed from a combination of GOOD and INTERPOLATED values. The aggregate value returned is marked GOOD if at least 80% of that value was derived from GOOD instance values.

20

Managing Tables

In this chapter:

- About Tables
- Guidelines for Managing Tables
- Creating Tables
- Loading Tables
- Automatically Collecting Statistics on Tables
- Altering Tables
- Redefining Tables Online
- Researching and Reversing Erroneous Table Changes
- Recovering Tables Using Oracle Flashback Table
- Dropping Tables
- Using Flashback Drop and Managing the Recycle Bin
- Managing Index-Organized Tables
- Managing External Tables
- Tables Data Dictionary Views

About Tables

Tables are the basic unit of data storage in an Oracle Database. Data is stored in rows and columns. You define a table with a table name, such as `employees`, and a set of columns. You give each column a column name, such as `employee_id`, `last_name`, and `job_id`; a data type, such as `VARCHAR2`, `DATE`, or `NUMBER`; and a width. The width can be predetermined by the data type, as in `DATE`. If columns are of the `NUMBER` data type, define precision and scale instead of width. A row is a collection of column information corresponding to a single record.

You can specify rules for each column of a table. These rules are called integrity constraints. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

You can invoke transparent data encryption to encrypt data before storing it. If users attempt to circumvent the database access control mechanisms by looking inside Oracle datafiles directly with operating system tools, encryption prevents these users from viewing sensitive data.

Tables can also include virtual columns. A **virtual column** is like any other table column, except that its value is derived by evaluating an expression. The expression can include columns from the same table, constants, SQL functions, and user-defined PL/SQL functions. You cannot explicitly write to a virtual column.

Some column types, such as `LOB`s, varrays, and nested tables, are stored in their own segments. `LOB`s and varrays are stored in `LOB` segments, while nested tables are stored in storage tables. You can specify a `STORAGE` clause for these segments that will override storage parameters specified at the table level.

After you create a table, you insert rows of data using SQL statements or using an Oracle bulk load utility. Table data can then be queried, deleted, or updated using SQL.

See Also:

- *Oracle Database Concepts* for an overview of tables
- *Oracle Database SQL Language Reference* for descriptions of Oracle Database data types
- Chapter 19, "Managing Space for Schema Objects" for guidelines for managing space for tables
- Chapter 18, "Managing Schema Objects" for information on additional aspects of managing tables, such as specifying integrity constraints and analyzing tables
- *Oracle Database Advanced Security Administrator's Guide* for a discussion of transparent data encryption

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 can improve performance when creating the table, as well as when loading, updating, and querying the table data.

The following topics are discussed:

- Design Tables Before Creating Them
- Specify the Type of Table to Create
- Specify the Location of Each Table
- Consider Parallelizing Table Creation
- Consider Using NOLOGGING When Creating Tables
- Consider Using Table Compression
- Consider Encrypting Columns That Contain Sensitive Data
- Understand Deferred Segment Creation
- Estimate Table Size and Plan Accordingly
- Restrictions to Consider When Creating Tables

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

establishing the attributes of the underlying tablespace that will hold the application tables. Either the DBA or the applications developer, or both working jointly, can be responsible for the actual creation of the tables, depending upon the practices for a site.

Working with the application developer, consider the following guidelines when designing tables:

- Use descriptive names for tables, columns, indexes, and clusters.
- Be consistent in abbreviations and in the use of singular and plural forms of table names and columns.
- Document the meaning of each table and its columns with the `COMMENT` command.
- Normalize each table.
- Select the appropriate data type for each column.
- Consider whether your applications would benefit from adding one or more virtual columns to some tables.
- Define columns that allow nulls last, to conserve storage space.
- Cluster tables whenever appropriate, to conserve storage space and optimize performance of SQL statements.

Before creating a table, you should also determine whether to use integrity constraints. Integrity constraints can be defined on the columns of a table to enforce the business rules of your database automatically.

Specify the Type of Table to Create

Here are the types of tables that you can create:

Type of Table	Description
Ordinary (heap-organized) table	This is the basic, general purpose type of table which is the primary subject of this chapter. Its data is stored as an unordered collection (heap).
Clustered table	A clustered table is a table that is part of a cluster. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. Clusters and clustered tables are discussed in Chapter 22, "Managing Clusters".
Index-organized table	Unlike an ordinary (heap-organized) table, 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. Index-organized tables are discussed in "Managing Index-Organized Tables" on page 20-55.

Type of Table	Description
Partitioned table	Partitioned tables enable your data to be broken down into smaller, more manageable pieces called partitions, or even subpartitions. Each partition can have separate physical attributes, such as compression enabled or disabled, type of compression, physical storage settings, and tablespace, thus providing a structure that can be better tuned for availability and performance. In addition, each partition can be managed individually, which can simplify and reduce the time required for backup and administration. Partitioned tables are discussed in *Oracle Database VLDB and Partitioning Guide*.

Specify the Location of Each Table

It is advisable to specify the `TABLESPACE` clause in a `CREATE TABLE` statement to identify the tablespace that is to store the new table. For partitioned tables, you can optionally identify the tablespace that is to store each partition. Ensure that you have the appropriate privileges and quota on any tablespaces that you use. 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, ensure 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 and decrease the time needed for database administration.

The following situations illustrate how not specifying a tablespace, or specifying an inappropriate one, can affect performance:

- If users' objects are created in the `SYSTEM` tablespace, the performance of the database can suffer, since both data dictionary objects and user objects must contend for the same datafiles. Users' objects should not be stored in the `SYSTEM` tablespace. To avoid this, ensure that all users are assigned default tablespaces when they are created in the database.
- If application-associated tables are arbitrarily stored in various tablespaces, the time necessary to complete administrative operations (such as backup and recovery) for the data of that application can be increased.

Consider Parallelizing Table Creation

You can use 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 20-15.

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 also parallelizing the table creation.

Consider Using Table Compression

As your database grows in size, consider using table compression. Compression saves disk space, reduces memory use in the database buffer cache, and can significantly speed query execution during reads. Compression has a cost in CPU overhead for data loading and DML. However, this cost may be offset by reduced I/O requirements.

Table compression is completely transparent to applications. It is useful in both decision support systems (DSS) and online transaction processing (OLTP) systems.

You can specify compression for a tablespace, a table, or a partition. If specified at the tablespace level, then all tables created in that tablespace are compressed by default.

Compression can occur while data is being inserted, updated, or bulk loaded into a table. Operations that permit compression include:

- Single-row or array inserts and updates
- The following direct-path INSERT methods:
  - Direct path SQL*Loader
  - CREATE TABLE AS SELECT statements
  - Parallel INSERT statements
  - INSERT statements with an APPEND or APPEND_VALUES hint

Oracle Database support two methods of table compression. They are summarized in Table 20–1.

Table 20–1 Table Compression Methods

Table Compression Method	Applications	CREATE/ALTER TABLE Syntax	Direct-Path INSERT	DML
Basic compression	DSS	COMPRESS [BASIC][1]	Yes	Yes[2]
OLTP compression	OLTP, DSS	COMPRESS FOR OLTP	Yes	Yes

[1] COMPRESS and COMPRESS BASIC are equivalent

[2] Inserted and updated rows are uncompressed

You specify table compression with the COMPRESS clause of the CREATE TABLE statement. You can enable compression for an existing table by using these clauses in an ALTER TABLE statement. In this case, only data that is inserted or updated after compression is enabled is compressed. Similarly, you can disable table compression for an existing compressed table with the ALTER TABLE...NOCOMPRESS statement. In this

case, all data that was already compressed remains compressed, and new data is inserted uncompressed.

To enable OLTP table compression, you must set the COMPATIBLE initialization parameter to 11.1.0 or higher.

See Also:

- The "Table Compression" section of *Oracle Database Concepts* for an overview of table compression
- "Compressed Tablespaces" on page 14-8

Examples

The following example enables OLTP table compression on the table orders:

```
CREATE TABLE orders  ...  COMPRESS FOR OLTP;
```

Data for the orders table is compressed during both direct-path INSERT and conventional DML.

The next two examples, which are equivalent, enable basic table compression on the sales_history table, which is a fact table in a data warehouse. Frequent queries are run against this table, but no DML is expected.

```
CREATE TABLE sales_history  ...  COMPRESS BASIC;

CREATE TABLE sales_history  ...  COMPRESS;
```

The next example demonstrates using the APPEND hint to insert rows into the sales_history table using direct-path INSERT.

```
INSERT /*+ APPEND */ INTO sales_history SELECT * FROM sales WHERE cust_id=8890;
COMMIT;
```

Compression and Partitioned Tables

A table can have both compressed and uncompressed partitions, and different partitions can use different compression methods. If the compression settings for a table and one of its partitions disagree, the partition setting has precedence for the partition.

To change the compression method for a partition, do one of the following:

- To change the compression method for new data only, use ALTER TABLE ... MODIFY PARTITION ... COMPRESS ...
- To change the compression method for both new and existing data, use either ALTER TABLE ... MOVE PARTITION ... COMPRESS ... or online table redefinition.

Determining If a Table Is Compressed

In the *_TABLES data dictionary views, compressed tables have ENABLED in the COMPRESSION column. For partitioned tables, this column is null, and the COMPRESSION column of the *_TAB_PARTITIONS views indicates the partitions that are compressed. In addition, the COMPRESS_FOR column indicates the compression method in use for the table or partition.

```
SELECT table_name, compression, compress_for FROM user_tables;

TABLE_NAME       COMPRESSION COMPRESS_FOR
---------------- ----------- ------------------
T1               DISABLED
```

```
T2                ENABLED     BASIC
T3                ENABLED     OLTP

SELECT table_name, partition_name, compression, compress_for
  FROM user_tab_partitions;

TABLE_NAME  PARTITION_NAME   COMPRESSION COMPRESS_FOR
----------- ---------------- ----------- ------------------------------
SALES       Q4_2008          ENABLED     OLTP
SALES       Q1_2009          ENABLED     OLTP
SALES       Q2_2009          ENABLED     OLTP
```

Adding and Dropping Columns in Compressed Tables

The following restrictions apply when adding columns to compressed tables:

- Basic compression—You cannot specify a default value for an added column.
- OLTP compression—If a default value is specified for an added column, the column must be `NOT NULL`. Added nullable columns with default values are not supported.

The following restrictions apply when dropping columns in compressed tables:

- Basic compression—Dropping a column is not supported.
- OLTP compression—`DROP COLUMN` is supported, but internally the database sets the column `UNUSED` to avoid long-running decompression and recompression operations.

Notes and Other Restrictions for Compressed Tables

- Online segment shrink is not supported for compressed tables.
- The table compression methods described in this section do not apply to SecureFile large objects (LOBs). SecureFile LOBs have their own compression methods. See *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information.
- Compression technology uses CPU. You should ensure that you have enough available CPU to handle the additional load.
- Tables created with basic compression have the `PCT_FREE` parameter automatically set to 0 unless you specify otherwise.

Packing Compressed Tables

If you use conventional DML on a table compressed with basic compression, then all inserted and updated rows are stored uncompressed. To "pack" the compressed table such that these rows are compressed, you can use an `ALTER TABLE MOVE` statement. This operation takes an exclusive lock on the table, and therefore prevents any updates and loads until it completes. If this is not acceptable, you can use online table redefinition.

See Also:

- *Oracle Database SQL Language Reference* for more details on the `CREATE TABLE...COMPRESS`, `ALTER TABLE...COMPRESS`, and `ALTER TABLE...MOVE` statements, including restrictions
- *Oracle Database VLDB and Partitioning Guide* for more information on table partitioning
- "Improving INSERT Performance with Direct-Path INSERT" on page 20-17
- "Redefining Tables Online" on page 20-33

Consider Encrypting Columns That Contain Sensitive Data

You can encrypt individual table columns that contain sensitive data. Examples of sensitive data include social security numbers, credit card numbers, and medical records. Column encryption is transparent to your applications, with some restrictions.

Although encryption is not meant to solve all security problems, it does protect your data from users who try to circumvent the security features of the database and access database files directly through the operating system file system.

Column encryption uses the transparent data encryption feature of Oracle Database, which requires that you create an *Oracle wallet* to store the master encryption key for the database. The wallet must be open before you can create a table with encrypted columns and before you can store or retrieve encrypted data. When you open the wallet, it is available to all sessions, and it remains open until you explicitly close it or until the database is shut down.

Transparent data encryption supports industry-standard encryption algorithms, including the following Advanced Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms:

- AES256
- AES192
- AES128
- 3DES168

You choose the algorithm to use when you create the table. All encrypted columns in the table use the same algorithm. The default is AES192. The encryption key length is implied by the algorithm name. For example, the AES128 algorithm uses 128-bit keys.

If you plan on encrypting many columns in one or more tables, you may want to consider encrypting an entire tablespace instead and storing these tables in that tablespace. Tablespace encryption, which also uses the transparent data encryption feature but encrypts at the physical block level, can perform better than encrypting many columns. Another reason to encrypt at the tablespace level is to address the following limitations of column encryption:

- If the `COMPATIBLE` initialization parameter set to 10.2.0, which is the minimum setting to enable transparent data encryption, data from encrypted columns that is involved in a sort or hash-join and that must be written to a temporary tablespace is written in clear text, and thus exposed to attacks. You must set `COMPATIBLE` to 11.1.0 or higher to ensure that encrypted data written to a temporary tablespace remains encrypted. Note that as long as `COMPATIBLE` is set to 10.2.0 or higher, data

from encrypted columns remains encrypted when written to the undo tablespace or the redo log.

- Certain data types, such as object data types, are not supported for column encryption.
- You cannot use the transportable tablespace feature for a tablespace that includes tables with encrypted columns.
- Other restrictions, which are detailed in *Oracle Database Advanced Security Administrator's Guide*.

See Also:

- "Encrypted Tablespaces" on page 14-8
- "Example: Creating a Table" on page 20-13
- *Oracle Database Advanced Security Administrator's Guide* for more information about transparent data encryption and for instructions for creating and opening wallets
- *Oracle Database SQL Language Reference* for information about the `CREATE TABLE` statement
- *Oracle Real Application Clusters Administration and Deployment Guide* for information on using an Oracle wallet in an Oracle Real Application Clusters environment
- "Transporting Tablespaces Between Databases" on page 14-30

Understand Deferred Segment Creation

Beginning with Oracle Database 11*g* Release 2, when you create heap-organized tables in a locally managed tablespace, the database defers table segment creation until the first row is inserted.

In addition, segment creation is deferred for any LOB columns of the table, any indexes created implicitly as part of table creation, and any indexes subsequently explicitly created on the table.

Note: In Release 11.2.0.1, deferred segment creation is not supported for partitioned tables. This restriction is removed in release 11.2.0.2 and later.

The advantages of this space allocation method are the following:

- It saves a significant amount of disk space in applications that create hundreds or thousands of tables upon installation, many of which might never be populated.
- It reduces application installation time.

There is a small performance penalty when the first row is inserted, because the new segment must be created at that time.

To enable deferred segment creation, compatibility must be set to '11.2.0' or higher.

The new clauses for the `CREATE TABLE` statement are:

- `SEGMENT CREATION DEFERRED`
- `SEGMENT CREATION IMMEDIATE`

These clauses override the default setting of the DEFERRED_SEGMENT_CREATION initialization parameter, TRUE, which defers segment creation. To disable deferred segment creation, set this parameter to FALSE.

Note that when you create a table with deferred segment creation, the new table appears in the *_TABLES views, but no entry for it appears in the *_SEGMENTS views until you insert the first row.

You can verify deferred segment creation by viewing the SEGMENT_CREATED column in *_TABLES, *_INDEXES, and *_LOBS views for nonpartitioned tables, and in *_TAB_PARTITIONS, *_IND_PARTITIONS, and *_LOB_PARTITIONS views for partitioned tables.

Note: With this new allocation method, it is essential that you do proper capacity planning so that the database has enough disk space to handle segment creation when tables are populated. See "Capacity Planning for Database Objects" on page 19-32.

The following example creates two tables to demonstrate deferred segment creation. The first table uses the SEGMENT CREATION DEFERRED clause. No segments are created for it initially. The second table uses the SEGMENT CREATION IMMEDIATE clause and, therefore, segments are created for it immediately.

```
CREATE TABLE part_time_employees (
    empno NUMBER(8),
    name VARCHAR2(30),
    hourly_rate NUMBER (7,2)
    )
    SEGMENT CREATION DEFERRED;

CREATE TABLE hourly_employees (
    empno NUMBER(8),
    name VARCHAR2(30),
    hourly_rate NUMBER (7,2)
    )
   SEGMENT CREATION IMMEDIATE
   PARTITION BY RANGE(empno)
    (PARTITION empno_to_100 VALUES LESS THAN (100),
    PARTITION empno_to_200 VALUES LESS THAN (200));
```

The following query against USER_SEGMENTS returns two rows for HOURLY_EMPLOYEES, one for each partition, but returns no rows for PART_TIME_EMPLOYEES because segment creation for that table was deferred.

```
SELECT segment_name, partition_name FROM user_segments;

SEGMENT_NAME         PARTITION_NAME
-------------------- ------------------------------
HOURLY_EMPLOYEES     EMPNO_TO_100
HOURLY_EMPLOYEES     EMPNO_TO_200
```

The USER_TABLES view shows that PART_TIME_EMPLOYEES has no segments:

```
SELECT table_name, segment_created FROM user_tables;

TABLE_NAME                     SEGMENT_CREATED
------------------------------ ----------------------------------------
PART_TIME_EMPLOYEES            NO
HOURLY_EMPLOYEES               N/A
```

For the HOURLY_EMPLOYEES table, which is partitioned, the segment_created column is N/A because the USER_TABLES view does not provide that information for partitioned tables. It is available from the USER_TAB_PARTITIONS view, shown below.

```
SELECT table_name, segment_created, partition_name
 FROM user_tab_partitions;

TABLE_NAME           SEGMENT_CREATED      PARTITION_NAME
-------------------- -------------------- ------------------------------
HOURLY_EMPLOYEES     YES                  EMPNO_TO_100
HOURLY_EMPLOYEES     YES                  EMPNO_TO_200
```

The following statements add employees to these tables.

```
INSERT INTO hourly_employees VALUES (99, 'FRose', 20.00);
INSERT INTO hourly_employees VALUES (150, 'LRose', 25.00);

INSERT INTO part_time_employees VALUES (50, 'KReilly', 10.00);
```

Repeating the same SELECT statements as before shows that PART_TIME_EMPLOYEES now has a segment, due to the insertion of row data. HOURLY_EMPLOYEES remains as before.

```
SELECT segment_name, partition_name FROM user_segments;

SEGMENT_NAME         PARTITION_NAME
-------------------- ------------------------------
PART_TIME_EMPLOYEES
HOURLY_EMPLOYEES     EMPNO_TO_100
HOURLY_EMPLOYEES     EMPNO_TO_200


SELECT table_name, segment_created FROM user_tables;

TABLE_NAME           SEGMENT_CREATED
-------------------- --------------------
PART_TIME_EMPLOYEES  YES
HOURLY_EMPLOYEES     N/A
```

The USER_TAB_PARTITIONS view does not change.

> **See Also:** *Oracle Database SQL Language Reference* for notes and restrictions on deferred segment creation

Materializing Segments

Beginning with Oracle Database 11*g* release 2 (11.2.0.2), the DBMS_SPACE_ADMIN package includes the MATERIALIZE_DEFERRED_SEGMENTS() procedure, which enables you to materialize segments for tables, table partitions, and dependent objects created with deferred segment creation enabled.

You can add segments as needed, rather than starting with more than you need and using database resources unnecessarily.

The following example materializes segments for the EMPLOYEES table in the HR schema.

```
BEGIN
  DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS(
    schema_name  => 'HR',
    table_name   => 'EMPLOYEES');
END;
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details about this procedure

Estimate Table Size and Plan Accordingly

Estimate the sizes of tables before creating them. Preferably, do this as part of database planning. Knowing the sizes, and uses, for database tables is an important part of database planning.

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.

You can use the estimated size and growth rate of an individual table to better determine the attributes of a tablespace and its underlying datafiles that are best suited for the table. This can enable you to more easily manage the table disk space and improve I/O performance of applications that use the table.

See Also: "Capacity Planning for Database Objects" on page 19-32

Restrictions to Consider When Creating Tables

Here are some restrictions that may affect your table planning and usage:

- Tables containing object types cannot be imported into a pre-Oracle8 database.
- You cannot merge an exported table into a preexisting table having the same name in a different schema.
- You cannot move types and extent tables to a different schema when the original data still exists in the database.
- Oracle Database has a limit on the total number of columns that a table (or attributes that an object type) can have. See *Oracle Database Reference* for this limit.

  Further, when you create a table that contains user-defined type data, the database 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 the database actually creates for the table can be more than those you specify.

  See Also: *Oracle Database Object-Relational Developer's Guide* 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:

- Example: Creating a Table
- Creating a Temporary Table
- Parallelizing Table Creation

> **See Also:** *Oracle Database SQL Language Reference* for exact syntax of the `CREATE TABLE` and other SQL statements discussed in this chapter

Example: 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,
         ssn        NUMBER(9) ENCRYPT USING 'AES256',
         job        VARCHAR2(10),
         mgr        NUMBER(5),
         hiredate   DATE DEFAULT (sysdate),
         photo      BLOB,
         sal        NUMBER(7,2),
         hrly_rate  NUMBER(7,2) GENERATED ALWAYS AS (sal/2080),
         comm       NUMBER(7,2),
         deptno     NUMBER(3) NOT NULL
                     CONSTRAINT admin_dept_fkey REFERENCES hr.departments
                     (department_id))
   TABLESPACE admin_tbs
   STORAGE ( INITIAL 50K);

COMMENT ON TABLE hr.admin_emp IS 'Enhanced employee table';
```

Note the following about this example:

- Integrity constraints are defined on several columns of the table.
- The `STORAGE` clause specifies the size of the first extent. See *Oracle Database SQL Language Reference* for details on this clause.
- Encryption is defined on one column (`ssn`), through the transparent data encryption feature of Oracle Database. The Oracle Wallet must therefore be open for this `CREATE TABLE` statement to succeed.
- The `photo` column is of data type `BLOB`, which is a member of the set of data types called large objects (LOBs). LOBs are used to store semi-structured data (such as an XML tree) and unstructured data (such as the stream of bits in a color image).
- One column is defined as a virtual column (`hrly_rate`). This column computes the employee's hourly rate as the yearly salary divided by 2,080. See *Oracle Database SQL Language Reference* for a discussion of rules for virtual columns.
- A `COMMENT` statement is used to store a comment for the table. You query the `*_TAB_COMMENTS` data dictionary views to retrieve such comments. See *Oracle Database SQL Language Reference* for more information.

See Also:

- *Oracle Database SQL Language Reference* for a description of the data types that you can specify for table columns
- "Managing Integrity Constraints" on page 18-9
- *Oracle Database Advanced Security Administrator's Guide* for information about transparent data encryption and the Oracle Wallet
- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOBs.

Creating a Temporary Table

Temporary tables are useful in applications where a result set is to be buffered (temporarily persisted), perhaps because it is constructed by running multiple DML operations. For example, consider the following:

A Web-based airlines reservations application allows a customer to create several optional itineraries. Each itinerary is represented by a row in a temporary table. The application updates the rows to reflect changes in the itineraries. When the customer decides which itinerary she wants to use, the application moves the row for that itinerary to a persistent table.

During the session, the itinerary data is private. At the end of the session, the optional itineraries are dropped.

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.

Use the `CREATE GLOBAL TEMPORARY TABLE` statement to create a temporary table. The `ON COMMIT` clause indicates if the data in the table is **transaction-specific** (the default) or **session-specific**, the implications of which are as follows:

ON COMMIT Setting	Implications
`DELETE ROWS`	This creates a temporary table that is transaction specific. A session becomes bound to the temporary table with a transactions first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit.
`PRESERVE ROWS`	This creates a temporary table that is session specific. A session gets bound to the temporary table with the first insert into the table in the session. This binding goes away at the end of the session or by issuing a `TRUNCATE` of the table in the session. The database truncates the table when you terminate the session.

This statement 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.

By default, rows in a temporary table are stored in the default temporary tablespace of the user who creates it. However, you can assign a temporary table to another tablespace upon creation of the temporary table by using the TABLESPACE clause of CREATE GLOBAL TEMPORARY TABLE. You can use this feature to conserve space used by temporary tables. For example, if you must perform many small temporary table operations and the default temporary tablespace is configured for sort operations and thus uses a large extent size, these small operations will consume lots of unnecessary disk space. In this case it is better to allocate a second temporary tablespace with a smaller extent size.

The following two statements create a temporary tablespace with a 64 KB extent size, and then a new temporary table in that tablespace.

```
CREATE TEMPORARY TABLESPACE tbs_t1
    TEMPFILE 'tbs_t1.f' SIZE 50m REUSE AUTOEXTEND ON
    MAXSIZE UNLIMITED
    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 64K;

CREATE GLOBAL TEMPORARY TABLE admin_work_area
        (startdate DATE,
         enddate DATE,
         class CHAR(20))
      ON COMMIT DELETE ROWS
      TABLESPACE tbs_t1;
```

See Also: "Temporary Tablespaces" on page 14-10

Unlike permanent tables, temporary tables and their indexes do not automatically allocate a segment when they are created. Instead, segments are allocated when the first INSERT (or CREATE TABLE AS SELECT) is performed. Therefore, if a SELECT, UPDATE, or DELETE is performed before the first INSERT, then the table appears to be empty.

DDL operations (except TRUNCATE) are allowed on an existing temporary table only if no session is currently bound to that temporary table.

If you rollback a transaction, the data you entered is lost, although the table definition persists.

A transaction-specific temporary table allows only one transaction at a time. If there are several autonomous transactions in a single transaction scope, each autonomous transaction can use the table only as soon as the previous one commits.

Because the data in a temporary table is, by definition, temporary, backup and recovery of temporary table data is not available in the event of a system failure. To prepare for such a failure, you should develop alternative methods for preserving temporary table data.

Parallelizing Table Creation

When you specify the AS SELECT clause to create a table and populate it with data from another table, you can use parallel execution. The CREATE TABLE...AS SELECT statement contains two parts: a CREATE part (DDL) and a SELECT part (query). Oracle Database 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 statement parallelizes the creation of a table and stores the result in a compressed format, using table compression:

```
CREATE TABLE hr.admin_emp_dept
     PARALLEL COMPRESS
     AS SELECT * FROM hr.employees
     WHERE department_id = 10;
```

In this case, the PARALLEL clause tells the database to select an optimum number of parallel execution servers when creating the table.

See Also:

- *Oracle Database VLDB and Partitioning Guide* for detailed information on using parallel execution
- "Managing Processes for Parallel SQL Execution" on page 5-20

Loading Tables

This section describes techniques for loading data into tables. In contains the following topics:

- Methods for Loading Tables
- Improving INSERT Performance with Direct-Path INSERT
- Using Conventional Inserts to Load Tables
- Avoiding Bulk INSERT Failures with DML Error Logging

Note: Beginning with release 11.2.0.2 of Oracle Database, the default size of the first extent of any new segment for a partitioned table is 8 MB instead of 64 KB. This helps improve performance of inserts and queries on partitioned tables. Although partitioned tables will start with a larger initial size, once sufficient data is inserted, the space consumption will be the same as in previous releases. You can override this default by setting the INITIAL size in the storage clause for the table. This new default only applies to table partitions and LOB partitions.

Methods for Loading Tables

There are several means of inserting or initially loading data into your tables. Most commonly used are the following:

Method	Description
SQL*Loader	This Oracle utility program loads data from external files into tables of an Oracle Database. For information about SQL*Loader, see *Oracle Database Utilities*.
`CREATE TABLE ... AS SELECT` statement (CTAS)	Using this SQL statement you can create a table and populate it with data selected from another existing table, including an external table.
`INSERT` statement	The `INSERT` statement enables you to add rows to a table, either by specifying the column values or by specifying a subquery that selects data from another existing table, including an external table. One form of the `INSERT` statement enables direct-path `INSERT`, which can improve performance, and is useful for bulk loading. See "Improving INSERT Performance with Direct-Path INSERT" on page 20-17. If you are inserting a lot of data and want to avoid statement termination and rollback if an error is encountered, you can insert with DML error logging. See "Avoiding Bulk INSERT Failures with DML Error Logging" on page 20-22.
`MERGE` statement	The `MERGE` statement enables you to insert rows into or update rows of a table, by selecting rows from another existing table. If a row in the new data corresponds to an item that already exists in the table, then an `UPDATE` is performed, else an `INSERT` is performed.

See *Oracle Database SQL Language Reference* for details on the `CREATE TABLE ... AS SELECT`, `INSERT`, and `MERGE` statements.

Note: Only a few details and examples of inserting data into tables are included in this book. Oracle documentation specific to data warehousing and application development provide more extensive information about inserting and manipulating data in tables. See:

- *Oracle Database Data Warehousing Guide*
- *Oracle Database SecureFiles and Large Objects Developer's Guide*

See Also: "Managing External Tables" on page 20-64

Improving INSERT Performance with Direct-Path INSERT

When loading large amounts of data, you can improve load performance by using direct-path `INSERT`.

This section contains:

- About Direct-Path INSERT
- How Direct-Path INSERT Works
- Loading Data with Direct-Path INSERT
- Specifying the Logging Mode for Direct-Path INSERT
- Additional Considerations for Direct-Path INSERT

About Direct-Path INSERT

Oracle Database inserts data into a table in one of two ways:

- During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints.
- During **direct-path INSERT operations**, the database appends the inserted data after existing data in the table. Data is written directly into datafiles, bypassing the buffer cache. Free space in the table is not reused, and referential integrity constraints are ignored. Direct-path `INSERT` can perform significantly better than conventional insert.

The database can insert data either in serial mode, where one process executes the statement, or in parallel mode, where multiple processes work together simultaneously to run a single SQL statement. The latter is referred to as parallel execution.

The following are benefits of direct-path `INSERT`:

- During direct-path `INSERT`, you can disable the logging of redo and undo entries to reduce load time. Conventional insert operations, in contrast, must always log such entries, because those operations reuse free space and maintain referential integrity.
- Direct-path `INSERT` operations ensure atomicity of the transaction, even when run in parallel mode. Atomicity cannot be guaranteed during parallel direct-path loads (using SQL*Loader).

When performing parallel direct-path loads, one notable difference between SQL*Loader and `INSERT` statements is the following: If errors occur during parallel direct-path loads with SQL*Loader, the load completes, but some indexes could be marked `UNUSABLE` at the end of the load. Parallel direct-path `INSERT`, in contrast, rolls back the statement if errors occur during index update.

How Direct-Path INSERT Works

You can use direct-path `INSERT` on both partitioned and nonpartitioned tables.

Serial Direct-Path INSERT into Partitioned or Nonpartitioned Tables The single process inserts data beyond the current high water mark of the table segment or of each partition segment. (The **high-water mark** is the level at which blocks have never been formatted to receive data.) When a `COMMIT` runs, the high-water mark is updated to the new value, making the data visible to users.

Parallel Direct-Path INSERT into Partitioned Tables This situation is analogous to serial direct-path `INSERT`. Each parallel execution server is assigned one or more partitions, with no more than one process working on a single partition. Each parallel execution server inserts data beyond the current high-water mark of its assigned partition segment(s). When a `COMMIT` runs, the high-water mark of each partition segment is updated to its new value, making the data visible to users.

Parallel Direct-Path INSERT into Nonpartitioned Tables Each parallel execution server allocates a new temporary segment and inserts data into that temporary segment. When a `COMMIT` runs, the parallel execution coordinator merges the new temporary segments into the primary table segment, where it is visible to users.

Loading Data with Direct-Path INSERT

You can load data with direct-path INSERT by using direct-path INSERT SQL statements, inserting data in parallel mode, or by using the Oracle SQL*Loader utility in direct-path mode. A direct-path INSERT can be done in either serial or parallel mode.

Serial Mode Inserts with SQL Statements You can activate direct-path INSERT in serial mode with SQL in the following ways:

- If you are performing an INSERT with a subquery, specify the APPEND hint in each INSERT statement, either immediately after the INSERT keyword, or immediately after the SELECT keyword in the subquery of the INSERT statement.
- If you are performing an INSERT with the VALUES clause, specify the APPEND_VALUES hint in each INSERT statement immediately after the INSERT keyword. Direct-path INSERT with the VALUES clause is best used when there are hundreds of thousands or millions of rows to load. The typical usage scenario is for array inserts using OCI. Another usage scenario might be inserts in a FORALL loop in PL/SQL.

If you specify the APPEND hint (as opposed to the APPEND_VALUES hint) in an INSERT statement with a VALUES clause, the APPEND hint is ignored and a conventional insert is performed.

The following is an example of using the APPEND hint to perform a direct-path INSERT:

```
INSERT /*+ APPEND */ INTO sales_hist SELECT * FROM sales WHERE cust_id=8890;
```

The following PL/SQL code fragment is an example of using the APPEND_VALUES hint:

```
FORALL i IN 1..numrecords
  INSERT /*+ APPEND_VALUES */ INTO orderdata
  VALUES(ordernum(i), custid(i), orderdate(i),shipmode(i), paymentid(i));
COMMIT;
```

Parallel Mode Inserts with SQL Statements When you are inserting in parallel mode, direct-path INSERT is the default. However, you can insert in parallel mode using conventional INSERT by using the NOAPPEND PARALLEL hint.

To run in parallel DML mode, the following requirements must be met:

- You must have Oracle Enterprise Edition installed.
- You must enable parallel DML in your session. To do this, submit the following statement:

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- You must meet at least one of the following requirements:
  - Specify the parallel attribute for the target table, either at create time or subsequently
  - Specify the PARALLEL hint for each insert operation
  - Set the database initialization parameter PARALLEL_DEGREE_POLICY to AUTO

To disable direct-path INSERT, specify the NOAPPEND hint in each INSERT statement. Doing so overrides parallel DML mode.

Note: You cannot query or modify data inserted using direct-path INSERT immediately after the insert is complete. If you attempt to do so, an ORA-12838 error is generated. You must first issue a COMMIT statement before attempting to read or modify the newly-inserted data.

See Also:

- "Using Conventional Inserts to Load Tables" on page 20-22
- *Oracle Database Performance Tuning Guide* for more information on using hints
- *Oracle Database SQL Language Reference* for more information on the subquery syntax of INSERT statements and for additional restrictions on using direct-path INSERT

Specifying the Logging Mode for Direct-Path INSERT

Direct-path INSERT lets you choose whether to log redo and undo information during the insert operation.

- You can specify logging mode for a table, partition, index, or LOB storage at create time (in a CREATE statement) or subsequently (in an ALTER statement).
- If you do not specify either LOGGING or NOLOGGING at these times:
  - The logging attribute of a partition defaults to the logging attribute of its table.
  - The logging attribute of a table or index defaults to the logging attribute of the tablespace in which it resides.
  - The logging attribute of LOB storage defaults to LOGGING if you specify CACHE for LOB storage. If you do not specify CACHE, then the logging attributes defaults to that of the tablespace in which the LOB values resides.
- You set the logging attribute of a tablespace in a CREATE TABLESPACE or ALTER TABLESPACE statements.

Note: If the database or tablespace is in FORCE LOGGING mode, then direct path INSERT always logs, regardless of the logging setting.

Direct-Path INSERT with Logging In this mode, Oracle Database performs full redo logging for instance and media recovery. If the database is in ARCHIVELOG mode, then you can archive redo logs to tape. If the database is in NOARCHIVELOG mode, then you can recover instance crashes but not disk failures.

Direct-Path INSERT without Logging In this mode, Oracle Database inserts data without redo or undo logging. Instead, the database logs a small number of block range invalidation redo records and periodically updates the control file with information about the most recent direct write.

Direct-path INSERT without logging improves performance. However, if you subsequently must perform media recovery, the invalidation redo records mark a range of blocks as logically corrupt, because no redo data was logged for them. Therefore, it is important that you back up the data after such an insert operation.

Beginning with release 11.2.0.2 of Oracle Database, you can significantly improve the performance of unrecoverable direct path inserts by disabling the periodic update of the control files. You do so by setting the initialization parameter `DB_UNRECOVERABLE_SCN_TRACKING` to `FALSE`. However, if you perform an unrecoverable direct path insert with these control file updates disabled, you will no longer be able to accurately query the database to determine if any datafiles are currently unrecoverable.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about unrecoverable datafiles
- The section "Determining If a Backup Is Required After Unrecoverable Operations" in *Oracle Data Guard Concepts and Administration*

Additional Considerations for Direct-Path INSERT

The following are some additional considerations when using direct-path `INSERT`.

Compressed Tables If a table is created with the basic compression, then you must use direct-path `INSERT` to compress table data as it is loaded. If a table is created with OLTP, warehouse, or online archival compression, then best compression ratios are achieved with direct-path `INSERT`.

See "Consider Using Table Compression" on page 20-5 for more information.

Index Maintenance with Direct-Path INSERT Oracle Database performs index maintenance at the end of direct-path `INSERT` operations on tables (partitioned or nonpartitioned) that have indexes. This index maintenance is performed by the parallel execution servers for parallel direct-path `INSERT` or by the single process for serial direct-path `INSERT`. You can avoid the performance impact of index maintenance by making the index unusable before the `INSERT` operation and then rebuilding it afterward.

See Also: "Making an Index Unusable" on page 21-17

Space Considerations with Direct-Path INSERT Direct-path `INSERT` requires more space than conventional-path `INSERT`.

All serial direct-path `INSERT` operations, as well as parallel direct-path `INSERT` into partitioned tables, insert data above the high-water mark of the affected segment. This requires some additional space.

Parallel direct-path `INSERT` into nonpartitioned tables requires even more space, because it creates a temporary segment for each degree of parallelism. If the nonpartitioned table is not in a locally managed tablespace in automatic segment-space management mode, you can modify the values of the `NEXT` and `PCTINCREASE` storage parameter and `MINIMUM EXTENT` tablespace parameter to provide sufficient (but not excess) storage for the temporary segments. Choose values for these parameters so that:

- The size of each extent is not too small (no less than 1 MB). This setting affects the total number of extents in the object.
- The size of each extent is not so large that the parallel `INSERT` results in wasted space on segments that are larger than necessary.

After the direct-path `INSERT` operation is complete, you can reset these parameters to settings more appropriate for serial operations.

Locking Considerations with Direct-Path INSERT During direct-path INSERT, the database obtains exclusive locks on the table (or on all partitions of a partitioned table). As a result, users cannot perform any concurrent insert, update, or delete operations on the table, and concurrent index creation and build operations are not permitted. Concurrent queries, however, are supported, but the query will return only the information before the insert operation.

Using Conventional Inserts to Load Tables

During **conventional INSERT operations**, the database reuses free space in the table, interleaving newly inserted data with existing data. During such operations, the database also maintains referential integrity constraints. Unlike direct-path INSERT operations, conventional INSERT operations do not require an exclusive lock on the table.

Several other restrictions apply to direct-path INSERT operations that do not apply to conventional INSERT operations. See *Oracle Database SQL Language Reference* for information about these restrictions.

You can perform a conventional INSERT operation in serial mode or in parallel mode using the NOAPPEND hint.

The following is an example of using the NOAPPEND hint to perform a conventional INSERT in serial mode:

```
INSERT /*+ NOAPPEND */ INTO sales_hist SELECT * FROM sales WHERE cust_id=8890;
```

The following is an example of using the NOAPPEND hint to perform a conventional INSERT in parallel mode:

```
INSERT /*+ NOAPPEND PARALLEL */ INTO sales_hist
   SELECT * FROM sales;
```

To run in parallel DML mode, the following requirements must be met:

- You must have Oracle Enterprise Edition installed.
- You must enable parallel DML in your session. To do this, submit the following statement:

  ```
  ALTER SESSION { ENABLE | FORCE } PARALLEL DML;
  ```

- You must meet at least one of the following requirements:
  - Specify the parallel attribute for the target table, either at create time or subsequently
  - Specify the PARALLEL hint for each insert operation
  - Set the database initialization parameter PARALLEL_DEGREE_POLICY to AUTO

Avoiding Bulk INSERT Failures with DML Error Logging

When you load a table using an INSERT statement with subquery, if an error occurs, the statement is terminated and rolled back in its entirety. This can be wasteful of time and system resources. For such INSERT statements, you can avoid this situation by using the DML error logging feature.

To use DML error logging, you add a statement clause that specifies the name of an error logging table into which the database records errors encountered during DML operations. When you add this error logging clause to the INSERT statement, certain types of errors no longer terminate and roll back the statement. Instead, each error is

logged and the statement continues. You then take corrective action on the erroneous rows at a later time.

DML error logging works with INSERT, UPDATE, MERGE, and DELETE statements. This section focuses on INSERT statements.

To insert data with DML error logging:

1. Create an error logging table. (Optional)

   You can create the table manually or use the DBMS_ERRLOG package to automatically create it for you. See "Creating an Error Logging Table" on page 20-25 for details.

2. Execute an INSERT statement and include an error logging clause. This clause:

   - Optionally references the error logging table that you created. If you do not provide an error logging table name, the database logs to an error logging table with a default name. The default error logging table name is ERR$_ followed by the first 25 characters of the name of the table that is being inserted into.
   - Optionally includes a **tag** (a numeric or string literal in parentheses) that gets added to the error log to help identify the statement that caused the errors. If the tag is omitted, a NULL value is used.
   - Optionally includes a REJECT LIMIT subclause.

     This subclause indicates the maximum number of errors that can be encountered before the INSERT statement terminates and rolls back. You can also specify UNLIMITED. The default reject limit is zero, which means that upon encountering the first error, the error is logged and the statement rolls back. For parallel DML operations, the reject limit is applied to each parallel server.

     Note: If the statement exceeds the reject limit and rolls back, the error logging table retains the log entries recorded so far.

   See *Oracle Database SQL Language Reference* for error logging clause syntax information.

3. Query the error logging table and take corrective action for the rows that generated errors.

   See "Error Logging Table Format", later in this section, for details on the error logging table structure.

Example The following statement inserts rows into the DW_EMPL table and logs errors to the ERR_EMPL table. The tag 'daily_load' is copied to each log entry. The statement terminates and rolls back if the number of errors exceeds 25.

```
INSERT INTO dw_empl
  SELECT employee_id, first_name, last_name, hire_date, salary, department_id
  FROM employees
  WHERE hire_date > sysdate - 7
  LOG ERRORS INTO err_empl ('daily_load') REJECT LIMIT 25
```

For more examples, see *Oracle Database SQL Language Reference* and *Oracle Database Data Warehousing Guide*.

Error Logging Table Format

The error logging table consists of two parts:

- A mandatory set of columns that describe the error. For example, one column contains the Oracle error number.

  Table 20–2 lists these error description columns.

- An optional set of columns that contain data from the row that caused the error. The column names match the column names from the table being inserted into (the "DML table").

  The number of columns in this part of the error logging table can be zero, one, or more, up to the number of columns in the DML table. If a column exists in the error logging table that has the same name as a column in the DML table, the corresponding data from the offending row being inserted is written to this error logging table column. If a DML table column does not have a corresponding column in the error logging table, the column is not logged. If the error logging table contains a column with a name that does not match a DML table column, the column is ignored.

  Because type conversion errors are one type of error that might occur, the data types of the optional columns in the error logging table must be types that can capture any value without data loss or conversion errors. (If the optional log columns were of the same types as the DML table columns, capturing the problematic data into the log could suffer the same data conversion problem that caused the error.) The database makes a best effort to log a meaningful value for data that causes conversion errors. If a value cannot be derived, `NULL` is logged for the column. An error on insertion into the error logging table causes the statement to terminate.

  Table 20–3 lists the recommended error logging table column data types to use for each data type from the DML table. These recommended data types are used when you create the error logging table automatically with the `DBMS_ERRLOG` package.

Table 20–2 Mandatory Error Description Columns

Column Name	Data Type	Description
`ORA_ERR_NUMBER$`	`NUMBER`	Oracle error number
`ORA_ERR_MESG$`	`VARCHAR2(2000)`	Oracle error message text
`ORA_ERR_ROWID$`	`ROWID`	Rowid of the row in error (for update and delete)
`ORA_ERR_OPTYP$`	`VARCHAR2(2)`	Type of operation: insert (`I`), update (`U`), delete (`D`) Note: Errors from the update clause and insert clause of a `MERGE` operation are distinguished by the `U` and `I` values.
`ORA_ERR_TAG$`	`VARCHAR2(2000)`	Value of the tag supplied by the user in the error logging clause

Table 20–3 Error Logging Table Column Data Types

DML Table Column Type	Error Logging Table Column Type	Notes
`NUMBER`	`VARCHAR2(4000)`	Able to log conversion errors

Table 20–3 (Cont.) Error Logging Table Column Data Types

DML Table Column Type	Error Logging Table Column Type	Notes
CHAR/VARCHAR2(n)	VARCHAR2(4000)	Logs any value without information loss
NCHAR/NVARCHAR2(n)	NVARCHAR2(4000)	Logs any value without information loss
DATE/TIMESTAMP	VARCHAR2(4000)	Logs any value without information loss. Converts to character format with the default date/time format mask
RAW	RAW(2000)	Logs any value without information loss
ROWID	UROWID	Logs any rowid type
LONG/LOB		Not supported
User-defined types		Not supported

Creating an Error Logging Table

You can create an error logging table manually, or you can use a PL/SQL package to automatically create one for you.

Creating an Error Logging Table Automatically You use the DBMS_ERRLOG package to automatically create an error logging table. The CREATE_ERROR_LOG procedure creates an error logging table with all of the mandatory error description columns plus all of the columns from the named DML table, and performs the data type mappings shown in Table 20–3.

The following statement creates the error logging table used in the previous example.

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('DW_EMPL', 'ERR_EMPL');
```

See *Oracle Database PL/SQL Packages and Types Reference* for details on DBMS_ERRLOG.

Creating an Error Logging Table Manually You use standard DDL to manually create the error logging table. See "Error Logging Table Format" on page 20-24 for table structure requirements. You must include all mandatory error description columns. They can be in any order, but must be the first columns in the table.

Error Logging Restrictions and Caveats

Oracle Database logs the following errors during DML operations:

- Column values that are too large
- Constraint violations (NOT NULL, unique, referential, and check constraints)
- Errors raised during trigger execution
- Errors resulting from type conversion between a column in a subquery and the corresponding column of the table
- Partition mapping errors
- Certain MERGE operation errors (ORA-30926: Unable to get a stable set of rows for MERGE operation.)

Some errors are not logged, and cause the DML operation to terminate and roll back. For a list of these errors and for other DML logging restrictions, see the discussion of the error_logging_clause in the INSERT section of *Oracle Database SQL Language Reference*.

Space Considerations Ensure that you consider space requirements before using DML error logging. You require available space not only for the table being inserted into, but also for the error logging table.

Security The user who issues the `INSERT` statement with DML error logging must have `INSERT` privileges on the error logging table.

> **See Also:** *Oracle Database SQL Language Reference* and *Oracle Database Data Warehousing Guide* for DML error logging examples.

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.

Formerly, you enabled `DBMS_STATS` to automatically gather statistics for a table by specifying the `MONITORING` keyword in the `CREATE` (or `ALTER`) `TABLE` statement. Starting with Oracle Database 11*g*, the `MONITORING` and `NOMONITORING` keywords have been deprecated and statistics are collected automatically. If you do specify these keywords, they are ignored.

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. The database uses these views to identify tables with stale statistics.

To disable monitoring of a table, set the `STATISTICS_LEVEL` initialization parameter to `BASIC`. Its default is `TYPICAL`, which enables automatic statistics collection. Automatic statistics collection and the `DBMS_STATS` package enable the optimizer to generate accurate execution plans.

> **See Also:**
>
> - *Oracle Database Reference* for detailed information on the `STATISTICS_LEVEL` initialization parameter
> - *Oracle Database Performance Tuning Guide* for information on managing optimizer statistics
> - *Oracle Database PL/SQL Packages and Types Reference* for information about using the `DBMS_STATS` package
> - "About Automated Maintenance Tasks" on page 26-1 for information on using the Scheduler to collect statistics automatically

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.

Many of the usages of the `ALTER TABLE` statement are presented in the following sections:

- Reasons for Using the ALTER TABLE Statement
- 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 Definition
- Adding Table Columns
- Renaming Table Columns
- Dropping Table Columns
- Placing a Table in Read-Only Mode

Caution: Before altering a table, familiarize yourself with the consequences of doing so. The *Oracle Database SQL Language 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 18-17 for information about how the database manages dependencies.

Reasons for Using the ALTER TABLE Statement

You can use the ALTER TABLE statement to perform any of the following actions that affect a table:

- Modify physical characteristics (`INITRANS` or storage parameters)
- Move the table to a new segment or tablespace
- Explicitly allocate an extent or deallocate unused space
- Add, drop, or rename columns, or modify an existing column definition (data type, length, default value, `NOT NULL` integrity constraint, column expression (for virtual columns), and encryption properties.)
- Modify the logging attributes of the table
- Modify the `CACHE`/`NOCACHE` attributes
- Add, modify or drop integrity constraints associated with the table
- Enable or disable integrity constraints or triggers associated with the table
- Modify the degree of parallelism for the table
- Rename a table
- Put a table in read-only mode and return it to read/write mode
- Add or modify index-organized table characteristics
- Alter the characteristics of an external table
- Add or modify `LOB` columns
- Add or modify object type, nested table, or varray columns

Many of these operations are discussed in succeeding sections.

Altering Physical Attributes of a Table

When altering the transaction entry setting INITRANS of a table, note that a new setting for INITRANS applies only to data blocks subsequently allocated for the table.

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.

> **See Also:** The discussions of the physical attributes clause and the storage clause in *Oracle Database SQL Language Reference*

Moving a Table to a New Segment or Tablespace

The ALTER TABLE...MOVE statement enables you to relocate data of a nonpartitioned table or of a partition of a partitioned table into a new segment, and optionally into a different tablespace for which you have quota. This statement also lets you modify any of the storage attributes of the table or partition, including those which cannot be modified using ALTER TABLE. You can also use the ALTER TABLE...MOVE statement with a COMPRESS clause to store the new segment using table compression.

One important reason to move a table to a new tablespace (with a new datafile) is to eliminate the possibility that old versions of column data—versions left on now unused portions of the disk due to segment shrink, reorganization, or previous table moves—could be viewed by bypassing the access controls of the database (for example with an operating system utility). This is especially important with columns that you intend to modify by adding transparent data encryption.

> **Note:** The ALTER TABLE...MOVE statement does not permit DML against the table while the statement is executing. To leave the table available for DML while moving it, see "Redefining Tables Online" on page 20-33.

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 );
```

Moving a table changes the rowids of the rows in the table. This causes indexes on the table to be marked UNUSABLE, and DML accessing the table using these indexes will receive an ORA-01502 error. The indexes on the table must be dropped or rebuilt. Likewise, any statistics for the table become invalid and new statistics should be collected after moving the table.

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.

> **See Also:** "Consider Encrypting Columns That Contain Sensitive Data" on page 20-8 for more information on transparent data encryption

Manually Allocating Storage for a Table

Oracle Database 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 "Reclaiming Wasted Space" on page 19-12.

Modifying an Existing Column Definition

Use the `ALTER TABLE...MODIFY` statement to modify an existing column definition. You can modify column data type, default value, column constraint, column expression (for virtual columns) and column encryption.

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 non-empty `CHAR` column.

If you are modifying a table to increase the length of a column of data type `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:** *Oracle Database SQL Language 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, the database immediately updates each row with the default value. Note that this can take some time, and that during the update, there is an exclusive DML lock on the table. For some types of tables (for example, tables without LOB columns), if you specify both a `NOT NULL` constraint and a default value, the database can optimize the column add operation and greatly reduce the amount of time that the table is locked for DML.

You can add a column with a `NOT NULL` constraint only if the table does not contain any rows, or you specify a default value.

> **See Also:** *Oracle Database SQL Language Reference* for additional rules and restrictions for adding table columns

Adding a Column to a Compressed Table

If you enable basic compression on a table, you can add columns only if you do not specify default values.

If you enable OLTP compression on a table, you can add columns to that table with or without default values. If a default value is specified, the column must be `NOT NULL`.

> **See Also:** "Consider Using Table Compression" on page 20-5

Adding a Virtual Column

If the new column is a virtual column, its value is determined by its column expression. (Note that a virtual column's value is calculated only when it is queried.)

Renaming Table Columns

Oracle Database lets you 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 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 column can invalidate dependent objects. However, when you rename a column, the database updates associated data dictionary tables to ensure that function-based indexes and check constraints remain valid.

Oracle Database also lets you rename column constraints. This is discussed in "Renaming Constraints" on page 18-13.

> **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:** *Oracle Database SQL Language 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. The first statement drops only the sal column:

```
ALTER TABLE hr.admin_emp DROP COLUMN sal;
```

The next 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
```

For external tables, the SET UNUSED statement is transparently converted into an ALTER TABLE DROP COLUMN statement. Because external tables consist of metadata only in the database, the DROP COLUMN statement performs equivalently to the SET UNUSED statement.

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 ALTER TABLE statement that follows, the optional clause CHECKPOINT is specified. This clause 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;
```

Dropping Columns in Compressed Tables

If you enable OLTP compression on a table, you can drop table columns. If you enable basic compression only, you cannot drop columns.

See Also: "Consider Using Table Compression" on page 20-5

Placing a Table in Read-Only Mode

You can place a table in read-only mode with the `ALTER TABLE...READ ONLY` statement, and return it to read/write mode with the `ALTER TABLE...READ WRITE` statement. An example of a table for which read-only mode makes sense is a configuration table. If your application contains configuration tables that are not modified after installation and that must not be modified by users, your application installation scripts can place these tables in read-only mode.

To place a table in read-only mode, you must have the `ALTER TABLE` privilege on the table or the `ALTER ANY TABLE` privilege. In addition, the `COMPATIBILE` initialization parameter must be set to 11.1.0 or greater.

The following example places the `SALES` table in read-only mode:

```
ALTER TABLE SALES READ ONLY;
```

The following example returns the table to read/write mode:

```
ALTER TABLE SALES READ WRITE;
```

When a table is in read-only mode, operations that attempt to modify table data are disallowed. The following operations are not permitted on a read-only table:

- All DML operations on the table or any of its partitions
- `TRUNCATE TABLE`
- `SELECT FOR UPDATE`
- `ALTER TABLE ADD/MODIFY/RENAME/DROP COLUMN`
- `ALTER TABLE SET COLUMN UNUSED`
- `ALTER TABLE DROP/TRUNCATE/EXCHANGE (SUB)PARTITION`
- `ALTER TABLE UPGRADE INCLUDING DATA` or `ALTER TYPE CASCADE INCLUDING TABLE DATA` for a type with read-only table dependents
- Online redefinition
- `FLASHBACK TABLE`

The following operations are permitted on a read-only table:

- `SELECT`
- `CREATE/ALTER/DROP INDEX`
- `ALTER TABLE ADD/MODIFY/DROP/ENABLE/DISABLE CONSTRAINT`
- `ALTER TABLE` for physical property changes
- `ALTER TABLE DROP UNUSED COLUMNS`
- `ALTER TABLE ADD/COALESCE/MERGE/MODIFY/MOVE/RENAME/SPLIT (SUB)PARTITION`
- `ALTER TABLE MOVE`
- `ALTER TABLE ENABLE ROW MOVEMENT` and `ALTER TABLE SHRINK`
- `RENAME TABLE` and `ALTER TABLE RENAME TO`
- `DROP TABLE`
- `ALTER TABLE DEALLOCATE UNUSED`

- `ALTER TABLE ADD/DROP SUPPLEMENTAL LOG`

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

Redefining Tables Online

In any database system, it is occasionally necessary to modify the logical or physical structure of a table to:

- Improve the performance of queries or DML
- Accommodate application changes
- Manage storage

Oracle Database provides a mechanism to make table structure modifications without significantly affecting the availability of the table. The mechanism is called **online table redefinition**. Redefining tables online provides a substantial increase in availability compared to traditional methods of redefining tables.

When a table is redefined online, it is accessible to both queries and DML during much of the redefinition process. The table is locked in the exclusive mode only during a very small window that is independent of the size of the table and complexity of the redefinition, and that is completely transparent to users.

Online table redefinition requires an amount of free space that is approximately equivalent to the space used by the table being redefined. More space may be required if new columns are added.

You can perform online table redefinition with the Enterprise Manager Reorganize Objects wizard or with the `DBMS_REDEFINITION` package.

> **Note: To invoke the Reorganize Objects wizard:**
>
> 1. On the Tables page of Enterprise Manager, click in the **Select** column to select the table to redefine.
> 2. In the Actions list, select **Reorganize**.
> 3. Click **Go**.

This section describes online redefinition with the `DBMS_REDEFINITION` package. It contains the following topics:

- Features of Online Table Redefinition
- Performing Online Redefinition with DBMS_REDEFINITION
- Results of the Redefinition Process
- Performing Intermediate Synchronization
- Aborting Online Table Redefinition and Cleaning Up After Errors
- Restrictions for Online Redefinition of Tables
- Online Redefinition of a Single Partition
- Online Table Redefinition Examples
- Privileges Required for the DBMS_REDEFINITION Package

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_REDEFINITION` package

Features of Online Table Redefinition

Online table redefinition enables you to:

- Modify the storage parameters of a table or cluster
- Move a table or cluster to a different tablespace

> **Note:** If it is not important to keep a table available for DML when moving it to another tablespace, you can use the simpler `ALTER TABLE MOVE` command. See "Moving a Table to a New Segment or Tablespace" on page 20-28.

- Add, modify, or drop one or more columns in a table or cluster
- Add or drop partitioning support (non-clustered tables only)
- Change partition structure
- Change physical properties of a single table partition, including moving it to a different tablespace in the same schema
- Change physical properties of a materialized view log or an Oracle Streams Advanced Queuing queue table
- Add support for parallel queries
- Re-create a table or cluster to reduce fragmentation

> **Note:** In many cases, online segment shrink is an easier way to reduce fragmentation. See "Reclaiming Wasted Space" on page 19-12.

- Change the organization of a normal table (heap organized) to an index-organized table, or do the reverse.
- Convert a relational table into a table with object columns, or do the reverse.
- Convert an object table into a relational table or a table with object columns, or do the reverse.

Performing Online Redefinition with DBMS_REDEFINITION

You use the `DBMS_REDEFINITION` package to perform online redefinition of a table. See *Oracle Database PL/SQL Packages and Types Reference* for package details.

To redefine a table online:

1. Choose the redefinition method: by key or by rowid

   By key—Select a primary key or pseudo-primary key to use for the redefinition. Pseudo-primary keys are unique keys with all component columns having `NOT NULL` constraints. For this method, the versions of the tables before and after redefinition should have the same primary key columns. This is the preferred and default method of redefinition.

By rowid—Use this method if no key is available. In this method, a hidden column named `M_ROW$$` is added to the post-redefined version of the table. It is recommended that this column be dropped or marked as unused after the redefinition is complete. If `COMPATIBLE` is set to 10.2.0 or higher, the final phase of redefinition automatically sets this column unused. You can then use the `ALTER TABLE ... DROP UNUSED COLUMNS` statement to drop it.

You cannot use this method on index-organized tables.

2. Verify that the table can be redefined online by invoking the `CAN_REDEF_TABLE` procedure. If the table is not a candidate for online redefinition, then this procedure raises an error indicating why the table cannot be redefined online.

3. Create an empty interim table (in the same schema as the table to be redefined) with all of the desired logical and physical 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. If a column is to be modified, create it in the interim table with the properties that you want.

   It is not necessary to create the interim table with all the indexes, constraints, grants, and triggers of the table being redefined, because these will be defined in step 7 when you copy dependent objects.

4. If redefining a partitioned table with the rowid method, enable row movement on the interim table.

   ```
   ALTER TABLE ... ENABLE ROW MOVEMENT;
   ```

5. (Optional) If you are redefining a large table and want to improve the performance of the next step by running it in parallel, issue the following statements:

   ```
   alter session force parallel dml parallel degree-of-parallelism;
   alter session force parallel query parallel degree-of-parallelism;
   ```

6. Start the redefinition process by calling `START_REDEF_TABLE`, providing the following:

   - The schema and table name of the table to be redefined
   - The interim table name
   - A column mapping string that maps the columns of table to be redefined to the columns of the interim table

     See "Constructing a Column Mapping String" on page 20-37 for details.

   - The redefinition method

     Package constants are provided for specifying the redefinition method. `DBMS_REDEFINITION.CONS_USE_PK` is used to indicate that the redefinition should be done using primary keys or pseudo-primary keys. `DBMS_REDEFINITION.CONS_USE_ROWID` is use to indicate that the redefinition should be done using rowids. If this argument is omitted, the default method of redefinition (`CONS_USE_PK`) is assumed.

   - Optionally, the columns to be used in ordering rows
   - If redefining only a single partition of a partitioned table, the partition name

   Because this process involves copying data, it may take a while. The table being redefined remains available for queries and DML during the entire process.

Note: If START_REDEF_TABLE fails for any reason, you must call ABORT_REDEF_TABLE, otherwise subsequent attempts to redefine the table will fail.

---

7. Copy dependent objects (such as triggers, indexes, materialized view logs, grants, and constraints) and statistics from the table being redefined to the interim table, using one of the following two methods. Method 1 is the preferred method because it is more automatic, but there may be times that you would choose to use method 2. Method 1 also enables you to copy table statistics to the interim table.

   - Method 1: Automatically Creating Dependent Objects

     Use the COPY_TABLE_DEPENDENTS procedure to automatically create dependent objects on the interim table. This procedure also **registers** the dependent objects. Registering the dependent objects enables the identities of these objects and their copied counterparts to be automatically swapped later as part of the redefinition completion process. The result is that when the redefinition is completed, the names of the dependent objects will be the same as the names of the original dependent objects.

     For more information, see "Creating Dependent Objects Automatically" on page 20-37.

   - Method 2: Manually Creating Dependent Objects

     You can manually create dependent objects on the interim table and then register them. For more information, see "Creating Dependent Objects Manually" on page 20-38.

     Note: In Oracle Database Release 9i, you were *required* to manually create the triggers, indexes, grants, and constraints on the interim table, and there may still be situations where you want to or must do so. In such cases, 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. When online redefinition completes, the referential constraint is automatically enabled. In addition, until the redefinition process is either completed or aborted, any trigger defined on the interim table does not execute.

     ---

8. Execute the FINISH_REDEF_TABLE procedure to complete the redefinition of the table. During this procedure, the original table is locked in exclusive mode for a very short time, independent of the amount of data in the original table. However, FINISH_REDEF_TABLE will wait for all pending DML to commit before completing the redefinition.

9. If you used rowids for the redefinition and your COMPATIBLE initialization parameter is set to 10.1.0 or lower, drop or set UNUSED the hidden column M_ROW$$ that is now in the redefined table.

   ```
   ALTER TABLE table_name SET UNUSED (M_ROW$$);
   ```

   If COMPATIBLE is 10.2.0 or higher, this hidden column is automatically set UNUSED when redefinition completes. You can then drop the column with the ALTER TABLE ... DROP UNUSED COLUMNS statement.

10. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

    If you drop the interim table while there are active queries running against it, you may encounter an `ORA-08103` error ("object no longer exists").

    See Also: "Online Table Redefinition Examples" on page 20-42

Constructing a Column Mapping String

The column mapping string that you pass as an argument to `START_REDEF_TABLE` contains a comma-delimited list of column mapping pairs, where each pair has the following syntax:

```
[expression]  column_name
```

The *`column_name`* term indicates a column in the interim table. The optional *`expression`* can include columns from the table being redefined, constants, operators, function or method calls, and so on, in accordance with the rules for expressions in a SQL `SELECT` statement. However, only simple deterministic subexpressions—that is, subexpressions whose results do not vary between one evaluation and the next—plus sequences and `SYSDATE` can be used. No subqueries are permitted. In the simplest case, the expression consists of just a column name from the table being redefined.

If an expression is present, its value is placed in the designated interim table column during redefinition. If the expression is omitted, it is assumed that both the table being redefined and the interim table have a column named *`column_name`*, and the value of that column in the table being redefined is placed in the same column in the interim table.

For example, if the `override` column in the table being redefined is to be renamed to `override_commission`, and every override commission is to be raised by 2%, the correct column mapping pair is:

```
override*1.02  override_commission
```

If you supply `'*'` or `NULL` as the column mapping string, it is assumed that all the columns (with their names unchanged) are to be included in the interim table. Otherwise, only those columns specified explicitly in the string are considered. The order of the column mapping pairs is unimportant.

For examples of column mapping strings, see "Online Table Redefinition Examples" on page 20-42.

Data Conversions When mapping columns, you can convert data types, with some restrictions.

If you provide `'*'` or `NULL` as the column mapping string, only the implicit conversions permitted by SQL are supported. For example, you can convert from `CHAR` to `VARCHAR2`, from `INTEGER` to `NUMBER`, and so on.

To perform other data type conversions, including converting from one object type to another or one collection type to another, you must provide a column mapping pair with an expression that performs the conversion. The expression can include the `CAST` function, built-in functions like `TO_NUMBER`, conversion functions that you create, and so on.

Creating Dependent Objects Automatically

You use the `COPY_TABLE_DEPENDENTS` procedure to automatically create dependent objects on the interim table.

You can discover if errors occurred while copying dependent objects by checking the num_errors output argument. If the ignore_errors argument is set to TRUE, the COPY_TABLE_DEPENDENTS procedure continues copying dependent objects even if an error is encountered when creating an object. You can view these errors by querying the DBA_REDEFINITION_ERRORS view.

Reasons for errors include:

- A lack of system resources
- A change in the logical structure of the table that would require recoding the dependent object.

  See Example 3 in "Online Table Redefinition Examples" on page 20-42 for a discussion of this type of error.

If ignore_errors is set to FALSE, the COPY_TABLE_DEPENDENTS procedure stops copying objects as soon as any error is encountered.

After you correct any errors you can again attempt to copy the dependent objects by reexecuting the COPY_TABLE_DEPENDENTS procedure. Optionally you can create the objects manually and then register them as explained in "Creating Dependent Objects Manually". The COPY_TABLE_DEPENDENTS procedure can be used multiple times as necessary. If an object has already been successfully copied, it is not copied again.

Creating Dependent Objects Manually

If you manually create dependent objects on the interim table with SQL*Plus or Enterprise Manager, you must then use the REGISTER_DEPENDENT_OBJECT procedure to register the dependent objects. Registering dependent objects enables the redefinition completion process to restore dependent object names to what they were before redefinition.

You would also use the REGISTER_DEPENDENT_OBJECT procedure if the COPY_TABLE_DEPENDENTS procedure failed to copy a dependent object and manual intervention is required.

You can query the DBA_REDEFINITION_OBJECTS view to determine which dependent objects are registered. This view shows dependent objects that were registered explicitly with the REGISTER_DEPENDENT_OBJECT procedure or implicitly with the COPY_TABLE_DEPENDENTS procedure. Only current information is shown in the view.

The UNREGISTER_DEPENDENT_OBJECT procedure can be used to unregister a dependent object on the table being redefined and on the interim table.

Note: Manually created dependent objects do not have to be identical to their corresponding original dependent objects. For example, when manually creating a materialized view log on the interim table, you can log different columns. In addition, the interim table can have more or fewer dependent objects.

Results of the Redefinition Process

The following are the end results of the redefinition process:

- The original table is redefined with the columns, indexes, constraints, grants, triggers, and statistics of the interim table.
- Dependent objects that were registered, either explicitly using REGISTER_DEPENDENT_OBJECT or implicitly using COPY_TABLE_DEPENDENTS, are renamed

automatically so that dependent object names on the redefined table are the same as before redefinition.

> **Note:** If no registration is done or no automatic copying is done, then you must manually rename the dependent objects.

- The referential constraints involving the interim table now involve the redefined table and are enabled.
- Any indexes, triggers, materialized view logs, 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.
- Some PL/SQL objects, views, synonyms, and other table-dependent objects may become invalidated. Only those objects that depend on elements of the table that were changed are invalidated. For example, if a PL/SQL procedure queries only columns of the redefined table that were unchanged by the redefinition, the procedure remains valid. See "Managing Object Dependencies" on page 18-17 for more information about schema object dependencies.

Performing Intermediate Synchronization

After the redefinition process has been started by calling `START_REDEF_TABLE` and before `FINISH_REDEF_TABLE` has been called, a large number of DML statements might have been executed on the original table. If you know that this is the case, it is recommended that you periodically synchronize the interim table with the original table. This is done by calling the `SYNC_INTERIM_TABLE` procedure. Calling this procedure reduces the time taken by `FINISH_REDEF_TABLE` to complete the redefinition process. There is no limit to the number of times that you can call `SYNC_INTERIM_TABLE`.

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.

Aborting Online Table Redefinition and Cleaning Up After Errors

In the event that an error is raised during the redefinition process, or if you choose to terminate the redefinition process, call `ABORT_REDEF_TABLE`. This procedure drops temporary logs and tables associated with the redefinition process. After this procedure is called, you can drop the interim table and its dependent objects.

If the online redefinition process must be restarted, if you do not first call `ABORT_REDEF_TABLE`, subsequent attempts to redefine the table will fail.

Restrictions for Online Redefinition of Tables

The following restrictions apply to the online redefinition of tables:

- If the table is to be redefined using primary key or pseudo-primary keys (unique keys or constraints with all component columns having *not null* constraints), then the post-redefinition table must have the same primary key or pseudo-primary key columns. If the table is to be redefined using rowids, then the table must not be an index-organized table.

- After redefining a table that has a materialized view log, the subsequent refresh of any dependent materialized view must be a complete refresh.
- Tables that are replicated in an n-way master configuration can be redefined, but horizontal subsetting (subset of rows in the table), vertical subsetting (subset of columns in the table), and column transformations are not allowed.
- The overflow table of an index-organized table cannot be redefined online independently.
- Tables with fine-grained access control (row-level security) cannot be redefined online.
- Tables for which Flashback Data Archive is enabled cannot be redefined online. You cannot enable Flashback Data Archive for the interim table.
- Tables with `BFILE` columns cannot be redefined online.
- Tables with `LONG` columns can be redefined online, but those columns must be converted to `CLOBS`. Also, `LONG RAW` columns must be converted to `BLOBS`. Tables with `LOB` columns are acceptable.
- On a system with sufficient resources for parallel execution, and in the case where the interim table is not partitioned, redefinition of a `LONG` column to a `LOB` column can be executed in parallel, provided that:
  - The segment used to store the `LOB` column in the interim table belongs to a locally managed tablespace with Automatic Segment Space Management (ASSM) enabled.
  - There is a simple mapping from one `LONG` column to one `LOB` column, and the interim table has only one `LOB` column.

  In the case where the interim table is partitioned, the normal methods for parallel execution for partitioning apply.
- Tables in the `SYS` and `SYSTEM` schema cannot be redefined online.
- Temporary tables cannot be redefined.
- A subset of rows in the table cannot be redefined.
- Only simple deterministic expressions, sequences, and `SYSDATE` 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 are being added as part of the redefinition and there are no column mappings for these columns, 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`.
- For materialized view logs and queue tables, online redefinition is restricted to changes in physical properties. No horizontal or vertical subsetting is permitted, nor are any column transformations. The only valid value for the column mapping string is `NULL`.
- You can convert a `VARRAY` to a nested table with the `CAST` operator in the column mapping. However, you cannot convert a nested table to a `VARRAY`.

Online Redefinition of a Single Partition

Beginning with Oracle Database 10g Release 2, you can redefine online a single partition of a table. This is useful if, for example, you want to move a partition to a different tablespace and keep the partition available for DML during the operation.

Another use for this capability is redefining an entire table, but doing it one partition at a time to reduce resource requirements. For example, to move a very large table to a different tablespace, you can move it one partition at a time to minimize the free space and undo space required to complete the move. Be aware, however, that when you redefine a single partition, if a global index is present, it is marked as `UNUSABLE` when redefinition is complete.

Redefining a single partition differs from redefining a table in the following ways:

- There is no need to copy dependent objects. It is not valid to use the `COPY_TABLE_DEPENDENTS` procedure when redefining a single partition.
- You must manually create any local indexes on the interim table.
- The column mapping string for `START_REDEF_TABLE` must be `NULL`.
- When using the by-rowid method, the final phase of redefinition drops the hidden column `M_ROW$$` instead of setting it unused.

Note: If it is not important to keep a partition available for DML when moving it to another tablespace, you can use the simpler `ALTER TABLE...MOVE PARTITION` statement.

See also:

- The section "Moving Partitions" in *Oracle Database VLDB and Partitioning Guide*
- *Oracle Database SQL Language Reference* for details on the `ALTER TABLE...MOVE PARTITION` statement

---

Rules for Online Redefinition of a Single Partition

The underlying mechanism for redefinition of a single partition is the **exchange partition** capability of the database (`ALTER TABLE...EXCHANGE PARTITION`). Rules and restrictions for online redefinition of a single partition are therefore governed by this mechanism. Here are some general restrictions:

- No logical changes (such as adding or dropping a column) are permitted.
- No changes to the partitioning method (such as changing from range partitioning to hash partitioning) are permitted.
- If a global index is present, it is marked as `UNUSABLE` when redefinition of any table partition is complete.

Here are the rules for defining the interim table:

- If the partition being redefined is a range, hash, or list partition, the interim table must be nonpartitioned.
- If the partition being redefined is a range partition of a composite range-hash partitioned table, the interim table must be a hash partitioned table. In addition, the partitioning key of the interim table must be identical to the subpartitioning key of the range-hash partitioned table, and the number of partitions in the interim table must be identical to the number of subpartitions in the range partition being redefined.

- If the partition being redefined is a hash partition that uses the rowid redefinition method, then row movement must be enabled on the interim table before redefinition starts.
- If the partition being redefined is a range partition of a composite range-list partitioned table, the interim table must be a list partitioned table. In addition, the partitioning key of the interim table must be identical to the subpartitioning key of the range-list partitioned table, and the values lists of the interim table's list partitions must exactly match the values lists of the list subpartitions in the range partition being redefined.
- If you define the interim table as compressed, then you must use the by-key method of redefinition, not the by-rowid method.

These additional rules apply if the table being redefined is a partitioned index-organized table:

- The interim table must also be index-organized.
- The original and interim tables must have primary keys on the same columns, in the same order.
- If key compression is enabled, it must be enabled for both the original and interim tables, with the same prefix length.
- Both the original and interim tables must have overflow segments, or neither can have them. Likewise for mapping tables.
- Both the original and interim tables must have identical storage attributes for any LOB columns.

> **See Also:** The section "Exchanging Partitions" in *Oracle Database VLDB and Partitioning Guide*

Online Table Redefinition Examples

For the following examples, see *Oracle Database PL/SQL Packages and Types Reference* for descriptions of all `DBMS_REDEFINITION` subprograms.

Example	Description
Example 1	Redefines a table by adding new columns and adding partitioning.
Example 2	Demonstrates redefinition with object data types.
Example 3	Demonstrates redefinition with manually registered dependent objects.
Example 4	Redefines a single table partition, moving it to a different tablespace.

Example 1

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` are added. (These existed in the original table but were dropped in previous examples.)
- 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. In this case you specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```
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. Copy dependent objects. (Automatically create any triggers, indexes, materialized view logs, grants, and constraints on `hr.int_admin_emp`.)

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('hr', 'admin_emp','int_admin_emp',
   DBMS_REDEFINITION.CONS_ORIG_PARAMS, TRUE, TRUE, TRUE, TRUE, num_errors);
END;
/
```

   Note that the `ignore_errors` argument is set to `TRUE` for this call. The reason is that the interim table was created with a primary key constraint, and when `COPY_TABLE_DEPENDENTS` attempts to copy the primary key constraint and index from the original table, errors occurs. You can ignore these errors, but you must run the query shown in the next step to see if there are other errors.

5. Query the `DBA_REDEFINITION_ERRORS` view to check for errors.

```
SQL> select object_name, base_table_name, ddl_txt from
         DBA_REDEFINITION_ERRORS;

OBJECT_NAME   BASE_TABLE_NAME  DDL_TXT
------------- --------------- ------------------------------
SYS_C005836   ADMIN_EMP       CREATE UNIQUE INDEX "HR"."TMP$
                              $_SYS_C0058360" ON "HR"."INT_A
                              DMIN_EMP" ("EMPNO")
```

```
SYS_C005836   ADMIN_EMP        ALTER TABLE "HR"."INT_ADMIN_EM
                               P" ADD CONSTRAINT "TMP$$_SYS_C
                               0058360" PRIMARY KEY
```

These errors are caused by the existing primary key constraint on the interim table and can be ignored. Note that with this approach, the names of the primary key constraint and index on the post-redefined table are changed. An alternate approach, one that avoids errors and name changes, would be to define the interim table without a primary key constraint. In this case, the primary key constraint and index are copied from the original table.

> **Note:** The best approach is to define the interim table with a primary key constraint, use `REGISTER_DEPENDENT_OBJECT` to register the primary key constraint and index, and then copy the remaining dependent objects with `COPY_TABLE_DEPENDENTS`. This approach avoids errors and ensures that the redefined table always has a primary key and that the dependent object names do not change.

6. Optionally, synchronize the interim table `hr.int_admin_emp`.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE('hr', 'admin_emp', 'int_admin_emp');
END;
/
```

7. 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.

8. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 2

This example redefines a table to change columns into object attributes. The redefined table gets a new column that is an object type.

The original table, named `CUSTOMER`, is defined as follows:

```
Name         Type
------------ -------------
CID          NUMBER            <- Primary key
NAME         VARCHAR2(30)
STREET       VARCHAR2(100)
CITY         VARCHAR2(30)
STATE        VARCHAR2(2)
ZIP          NUMBER(5)
```

The type definition for the new object is:

```
CREATE TYPE ADDR_T AS OBJECT (
   street VARCHAR2(100),
```

```
city VARCHAR2(30),
state VARCHAR2(2),
zip NUMBER(5, 0) );
```

Here are the steps for this redefinition:

1. Verify that the table is a candidate for online redefinition. Specify that the redefinition is to be done using primary keys or pseudo-primary keys.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE('STEVE','CUSTOMER',
        DBMS_REDEFINITION.CONS_USE_PK);
END;
/
```

2. Create the interim table `int_customer`.

```
CREATE TABLE INT_CUSTOMER(
  CID NUMBER,
  NAME  VARCHAR2(30),
  ADDR  ADDR_T);
```

   Note that no primary key is defined on the interim table. When dependent objects are copied in step 5, the primary key constraint and index are copied.

3. Because `CUSTOMER` is a very large table, specify parallel operations for the next step.

```
alter session force parallel dml parallel 4;
alter session force parallel query parallel 4;
```

4. Start the redefinition process using primary keys.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
   uname       => 'STEVE',
   orig_table  => 'CUSTOMER',
   int_table   => 'INT_CUSTOMER',
   col_mapping => 'cid cid,  name name,
      addr_t(street, city, state, zip) addr');
END;
/
```

   Note that `addr_t(street, city, state, zip)` is a call to the object constructor.

5. Copy dependent objects.

```
DECLARE
num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS(
   'STEVE','CUSTOMER','INT_CUSTOMER',DBMS_REDEFINITION.CONS_ORIG_PARAMS,
    TRUE, TRUE, TRUE, FALSE, num_errors, TRUE);
END;
/
```

   Note that for this call, the final argument indicates that table statistics are to be copied to the interim table.

6. Optionally synchronize the interim table.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

7. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE('STEVE', 'CUSTOMER', 'INT_CUSTOMER');
END;
/
```

8. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

Example 3

This example addresses the situation where a dependent object must be manually created and registered.

Consider the case where a table `T1` has a column named `C1`, and where this column becomes `C2` after the redefinition. Assume that there is an index `Index1` on `C1`. In this case, `COPY_TABLE_DEPENDENTS` tries to create an index on the interim table corresponding to `Index1`, and tries to create it on a column `C1`, which does not exist on the interim table. This results in an error. You must therefore manually create the index on column `C2` and register it. Here are the steps:

1. Create the interim table `INT_T1` and create an index `Int_Index1` on column `C2`.
2. Ensure that `T1` is a candidate for online redefinition with `CAN_REDEF_TABLE`, and then begin the redefinition process with `START_REDEF_TABLE`.
3. Register the original (`Index1`) and interim (`Int_Index1`) dependent objects.

```
BEGIN
 DBMS_REDEFINITION.REGISTER_DEPENDENT_OBJECT(
   uname         => 'STEVE',
   orig_table    => 'T1',
   int_table     => 'INT_T1',
   dep_type      => DBMS_REDEFINITION.CONS_INDEX,
   dep_owner     => 'STEVE',
   dep_orig_name => 'Index1',
   dep_int_name  => 'Int_Index1');
END;
/
```

4. Use `COPY_TABLE_DEPENDENTS` to copy the remaining dependent objects.
5. Optionally synchronize the interim table.
6. Complete the redefinition and drop the interim table.

Example 4

This example demonstrates redefining a single partition. It moves the oldest partition of a range-partitioned sales table to a tablespace named `TBS_LOW_FREQ`. The table containing the partition to be redefined is defined as follows:

```
CREATE TABLE salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE users
PARTITION BY RANGE(s_saledate)
(PARTITION sal03q1 VALUES LESS THAN (TO_DATE('01-APR-2003', 'DD-MON-YYYY')),
PARTITION sal03q2 VALUES LESS THAN (TO_DATE('01-JUL-2003', 'DD-MON-YYYY')),
```

```
PARTITION sal03q3 VALUES LESS THAN (TO_DATE('01-OCT-2003', 'DD-MON-YYYY')),
PARTITION sal03q4 VALUES LESS THAN (TO_DATE('01-JAN-2004', 'DD-MON-YYYY')));
```

The table has a local partitioned index that is defined as follows:

```
CREATE INDEX sales_index ON salestable
   (s_saledate, s_productid, s_custid) LOCAL;
```

Here are the steps. In the following procedure calls, note the extra argument: partition name (`part_name`).

1. Ensure that `salestable` is a candidate for redefinition.

```
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE(
   uname        => 'STEVE',
   tname        => 'SALESTABLE',
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal03q1');
END;
/
```

2. Create the interim table in the `TBS_LOW_FREQ` tablespace. Because this is a redefinition of a range partition, the interim table is nonpartitioned.

```
CREATE TABLE int_salestable
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
TABLESPACE tbs_low_freq;
```

3. Start the redefinition process using rowid.

```
BEGIN
  DBMS_REDEFINITION.START_REDEF_TABLE(
   uname        => 'STEVE',
   orig_table   => 'salestable',
   int_table    => 'int_salestable',
   col_mapping  => NULL,
   options_flag => DBMS_REDEFINITION.CONS_USE_ROWID,
   part_name    => 'sal03q1');
END;
/
```

4. Manually create any local indexes on the interim table.

```
CREATE INDEX int_sales_index ON int_salestable
(s_saledate, s_productid, s_custid)
TABLESPACE tbs_low_freq;
```

5. Optionally synchronize the interim table.

```
BEGIN
  DBMS_REDEFINITION.SYNC_INTERIM_TABLE(
   uname      => 'STEVE',
   orig_table => 'salestable',
   int_table  => 'int_salestable',
   part_name  => 'sal03q1');
END;
/
```

6. Complete the redefinition.

```
BEGIN
  DBMS_REDEFINITION.FINISH_REDEF_TABLE(
   uname      => 'STEVE',
   orig_table => 'salestable',
   int_table  => 'int_salestable',
   part_name  => 'sal03q1');
END;
/
```

7. Wait for any long-running queries against the interim table to complete, and then drop the interim table.

The following query shows that the oldest partition has been moved to the new tablespace:

```
select partition_name, tablespace_name from user_tab_partitions
 where table_name = 'SALESTABLE';

PARTITION_NAME                 TABLESPACE_NAME
------------------------------ ------------------------------
SAL03Q1                        TBS_LOW_FREQ
SAL03Q2                        USERS
SAL03Q3                        USERS
SAL03Q4                        USERS

4 rows selected.
```

Privileges Required for the DBMS_REDEFINITION Package

Execute privileges on the `DBMS_REDEFINITION` package are 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`

The following additional privileges are required to execute `COPY_TABLE_DEPENDENTS`:

- `CREATE ANY TRIGGER`
- `CREATE ANY INDEX`

Researching and Reversing Erroneous Table Changes

To enable you to research and reverse erroneous changes to tables, Oracle Database provides a a group of features that you can use to view past states of database objects or to return database objects to a previous state without using point-in-time media recovery. These features are known as **Oracle Flashback features**, and are described in *Oracle Database Advanced Application Developer's Guide*.

To research an erroneous change, you can use multiple Oracle Flashback queries to view row data at specific points in time. A more efficient approach would be to use Oracle Flashback Version Query to view all changes to a row over a period of time. With this feature, you append a `VERSIONS` clause to a `SELECT` statement that specifies a system change number (SCN) or timestamp range between which you want to view

changes to row values. The query also can return associated metadata, such as the transaction responsible for the change.

After you identify an erroneous transaction, you can use Oracle Flashback Transaction Query to identify other changes that were made by the transaction. You can then use Oracle Flashback Transaction to reverse the erroneous transaction. (Note that Oracle Flashback Transaction must also reverse all dependent transactions—subsequent transactions involving the same rows as the erroneous transaction.) You also have the option of using Oracle Flashback Table, described in "Recovering Tables Using Oracle Flashback Table" on page 20-49.

Note: You must be using automatic undo management to use Oracle Flashback features. See "Introduction to Automatic Undo Management" on page 16-2.

See Also: *Oracle Database Advanced Application Developer's Guide* for information about Oracle Flashback features.

Recovering Tables Using Oracle Flashback Table

Oracle Flashback Table enables you to restore a table to its state as of a previous point in time. It provides a fast, online solution for recovering a table that has been accidentally modified or deleted by a user or application. In many cases, Oracle Flashback Table eliminates the need for you to perform more complicated point-in-time recovery operations.

Oracle Flashback Table:

- Restores all data in a specified table to a previous point in time described by a timestamp or SCN.
- Performs the restore operation online.
- Automatically maintains all of the table attributes, such as indexes, triggers, and constraints that are necessary for an application to function with the flashed-back table.
- Maintains any remote state in a distributed environment. For example, all of the table modifications required by replication if a replicated table is flashed back.
- Maintains data integrity as specified by constraints. Tables are flashed back provided none of the table constraints are violated. This includes any referential integrity constraints specified between a table included in the `FLASHBACK TABLE` statement and another table that is not included in the `FLASHBACK TABLE` statement.
- Even after a flashback operation, the data in the original table is not lost. You can later revert to the original state.

Note: You must be using automatic undo management to use Oracle Flashback Table. See "Introduction to Automatic Undo Management" on page 16-2.

See Also: *Oracle Database Backup and Recovery User's Guide* for more information about the `FLASHBACK TABLE` statement.

Dropping Tables

To drop a table that you no longer need, use the `DROP TABLE` statement. The table must be contained in your schema or you must have the `DROP ANY TABLE` system privilege.

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 18-17 for information about how the database 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 22, "Managing Clusters".

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, then include the `CASCADE` clause in the `DROP TABLE` statement, as shown below:

```
DROP TABLE hr.admin_emp CASCADE CONSTRAINTS;
```

When you drop a table, normally the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the `FLASHBACK TABLE` statement if you find that you dropped the table in error. If you should want to immediately release the space associated with the table at the time you issue the `DROP TABLE` statement, include the `PURGE` clause as shown in the following statement:

```
DROP TABLE hr.admin_emp PURGE;
```

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 18-5.

Using Flashback Drop and Managing the Recycle Bin

When you drop a table, the database does not immediately remove the space associated with the table. The database renames the table and places it and any associated objects in a recycle bin, where, in case the table was dropped in error, it can

be recovered at a later time. This feature is called Flashback Drop, and the `FLASHBACK TABLE` statement is used to restore the table. Before discussing the use of the `FLASHBACK TABLE` statement for this purpose, it is important to understand how the recycle bin works, and how you manage its contents.

This section contains the following topics:

- What Is the Recycle Bin?
- Viewing and Querying Objects in the Recycle Bin
- Purging Objects in the Recycle Bin
- Restoring Tables from the Recycle Bin

What Is the Recycle Bin?

The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects such as indexes, constraints, nested tables, and the likes are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

Each user can be thought of as having his own recycle bin, because, unless a user has the `SYSDBA` privilege, the only objects that the user has access to in the recycle bin are those that the user owns. A user can view his objects in the recycle bin using the following statement:

```
SELECT * FROM RECYCLEBIN;
```

When you drop a tablespace including its contents, the objects in the tablespace are not placed in the recycle bin and the database purges any entries in the recycle bin for objects located in the tablespace. The database also purges any recycle bin entries for objects in a tablespace when you drop the tablespace, not including contents, and the tablespace is otherwise empty. Likewise:

- When you drop a user, any objects belonging to the user are not placed in the recycle bin and any objects in the recycle bin are purged.
- When you drop a cluster, its member tables are not placed in the recycle bin and any former member tables in the recycle bin are purged.
- When you drop a type, any dependent objects such as subtypes are not placed in the recycle bin and any former dependent objects in the recycle bin are purged.

Object Naming in the Recycle Bin

When a dropped table is moved to the recycle bin, the table and its associated objects are given system-generated names. This is necessary to avoid name conflicts that may arise if multiple tables have the same name. This could occur under the following circumstances:

- A user drops a table, re-creates it with the same name, then drops it again.
- Two users have tables with the same name, and both users drop their tables.

The renaming convention is as follows:

```
BIN$unique_id$version
```

where:

- *unique_id* is a 26-character globally unique identifier for this object, which makes the recycle bin name unique across all databases
- *version* is a version number assigned by the database

Enabling and Disabling the Recycle Bin

When the recycle bin is enabled, dropped tables and their dependent objects are placed in the recycle bin. When the recycle bin is disabled, dropped tables and their dependent objects are *not* placed in the recycle bin; they are just dropped, and you must use other means to recover them (such as recovering from backup).

Disabling the recycle bin does not purge or otherwise affect objects already in the recycle bin. The recycle bin is enabled by default.

You enable and disable the recycle bin by changing the `recyclebin` initialization parameter. This parameter is not dynamic, so a database restart is required when you change it with an `ALTER SYSTEM` statement.

To disable the recycle bin:

1. Issue one of the following statements:

```
ALTER SESSION SET recyclebin = OFF;

ALTER SYSTEM SET recyclebin = OFF SCOPE = SPFILE;
```

2. If you used `ALTER SYSTEM`, restart the database.

To enable the recycle bin:

1. Issue one of the following statements:

```
ALTER SESSION SET recyclebin = ON;

ALTER SYSTEM SET recyclebin = ON SCOPE = SPFILE;
```

2. If you used `ALTER SYSTEM`, restart the database.

See Also:

- "About Initialization Parameters and Initialization Parameter Files" on page 2-24 for more information on initialization parameters
- "Changing Initialization Parameter Values" on page 2-35 for a description of dynamic and static initialization parameters

Viewing and Querying Objects in the Recycle Bin

Oracle Database provides two views for obtaining information about objects in the recycle bin:

View	Description
`USER_RECYCLEBIN`	This view can be used by users to see their own dropped objects in the recycle bin. It has a synonym `RECYCLEBIN`, for ease of use.
`DBA_RECYCLEBIN`	This view gives administrators visibility to all dropped objects in the recycle bin

One use for these views is to identify the name that the database has assigned to a dropped object, as shown in the following example:

```
SELECT object_name, original_name FROM dba_recyclebin
   WHERE owner = 'HR';

OBJECT_NAME                    ORIGINAL_NAME
------------------------------ --------------------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 EMPLOYEES
```

You can also view the contents of the recycle bin using the SQL*Plus command `SHOW RECYCLEBIN`.

```
SQL> show recyclebin

ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
EMPLOYEES        BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TABLE        2003-10-27:14:00:19
```

You can query objects that are in the recycle bin, just as you can query other objects. However, you must specify the name of the object as it is identified in the recycle bin. For example:

```
SELECT * FROM "BIN$yrMKlZaVMhfgNAgAIMenRA==$0";
```

Purging Objects in the Recycle Bin

If you decide that you are never going to restore an item from the recycle bin, you can use the `PURGE` statement to remove the items and their associated objects from the recycle bin and release their storage space. You need the same privileges as if you were dropping the item.

When you use the `PURGE` statement to purge a table, you can use the name that the table is known by in the recycle bin or the original name of the table. The recycle bin name can be obtained from either the `DBA_` or `USER_RECYCLEBIN` view as shown in "Viewing and Querying Objects in the Recycle Bin" on page 20-52. The following hypothetical example purges the table `hr.int_admin_emp`, which was renamed to `BIN$jsleilx392mk2=293$0` when it was placed in the recycle bin:

```
PURGE TABLE BIN$jsleilx392mk2=293$0;
```

You can achieve the same result with the following statement:

```
PURGE TABLE int_admin_emp;
```

You can use the `PURGE` statement to purge all the objects in the recycle bin that are from a specified tablespace or only the tablespace objects belonging to a specified user, as shown in the following examples:

```
PURGE TABLESPACE example;
PURGE TABLESPACE example USER oe;
```

Users can purge the recycle bin of their own objects, and release space for objects, by using the following statement:

```
PURGE RECYCLEBIN;
```

If you have the `SYSDBA` privilege, then you can purge the entire recycle bin by specifying `DBA_RECYCLEBIN`, instead of `RECYCLEBIN` in the previous statement.

You can also use the `PURGE` statement to purge an index from the recycle bin or to purge from the recycle bin all objects in a specified tablespace.

> **See Also:** *Oracle Database SQL Language Reference* for more information on the PURGE statement

Restoring Tables from the Recycle Bin

Use the FLASHBACK TABLE ... TO BEFORE DROP statement to recover objects from the recycle bin. You can specify either the name of the table in the recycle bin or the original table name. An optional RENAME TO clause lets you rename the table as you recover it. The recycle bin name can be obtained from either the DBA_ or USER_RECYCLEBIN view as shown in "Viewing and Querying Objects in the Recycle Bin" on page 20-52. To use the FLASHBACK TABLE ... TO BEFORE DROP statement, you need the same privileges required to drop the table.

The following example restores int_admin_emp table and assigns to it a new name:

```
FLASHBACK TABLE int_admin_emp TO BEFORE DROP
   RENAME TO int2_admin_emp;
```

The system-generated recycle bin name is very useful if you have dropped a table multiple times. For example, suppose you have three versions of the int2_admin_emp table in the recycle bin and you want to recover the second version. You can do this by issuing two FLASHBACK TABLE statements, or you can query the recycle bin and then flashback to the appropriate system-generated name, as shown in the following example. Including the create time in the query can help you verify that you are restoring the correct table.

```
SELECT object_name, original_name, createtime FROM recyclebin;

OBJECT_NAME                    ORIGINAL_NAME   CREATETIME
------------------------------ --------------- -------------------
BIN$yrMKlZaLMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:05:52
BIN$yrMKlZaVMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:21:25:13
BIN$yrMKlZaQMhfgNAgAIMenRA==$0 INT2_ADMIN_EMP  2006-02-05:22:05:53

FLASHBACK TABLE BIN$yrMKlZaVMhfgNAgAIMenRA==$0 TO BEFORE DROP;
```

Restoring Dependent Objects

When you restore a table from the recycle bin, dependent objects such as indexes do not get their original names back; they retain their system-generated recycle bin names. You must manually rename dependent objects to restore their original names. If you plan to manually restore original names for dependent objects, ensure that you make note of each dependent object's system-generated recycle bin name *before* you restore the table.

The following is an example of restoring the original names of some of the indexes of the dropped table JOB_HISTORY, from the HR sample schema. The example assumes that you are logged in as the HR user.

1. After dropping JOB_HISTORY and before restoring it from the recycle bin, run the following query:

   ```
   SELECT OBJECT_NAME, ORIGINAL_NAME, TYPE FROM RECYCLEBIN;

   OBJECT_NAME                    ORIGINAL_NAME             TYPE
   ------------------------------ ------------------------- --------
   BIN$DBo9UChtZSbgQFeMiAdCcQ==$0 JHIST_JOB_IX              INDEX
   BIN$DBo9UChuZSbgQFeMiAdCcQ==$0 JHIST_EMPLOYEE_IX         INDEX
   BIN$DBo9UChvZSbgQFeMiAdCcQ==$0 JHIST_DEPARTMENT_IX       INDEX
   BIN$DBo9UChwZSbgQFeMiAdCcQ==$0 JHIST_EMP_ID_ST_DATE_PK   INDEX
   BIN$DBo9UChxZSbgQFeMiAdCcQ==$0 JOB_HISTORY               TABLE
   ```

2. Restore the table with the following command:

```
FLASHBACK TABLE JOB_HISTORY TO BEFORE DROP;
```

3. Run the following query to verify that all JOB_HISTORY indexes retained their system-generated recycle bin names:

```
SELECT INDEX_NAME FROM USER_INDEXES WHERE TABLE_NAME = 'JOB_HISTORY';

INDEX_NAME
------------------------------
BIN$DBo9UChwZSbgQFeMiAdCcQ==$0
BIN$DBo9UChtZSbgQFeMiAdCcQ==$0
BIN$DBo9UChuZSbgQFeMiAdCcQ==$0
BIN$DBo9UChvZSbgQFeMiAdCcQ==$0
```

4. Restore the original names of the first two indexes as follows:

```
ALTER INDEX "BIN$DBo9UChtZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_JOB_IX;
ALTER INDEX "BIN$DBo9UChuZSbgQFeMiAdCcQ==$0" RENAME TO JHIST_EMPLOYEE_IX;
```

   Note that double quotes are required around the system-generated names.

Managing Index-Organized Tables

This section describes aspects of managing index-organized tables, and contains the following topics:

- What Are Index-Organized Tables?
- Creating Index-Organized Tables
- Maintaining Index-Organized Tables
- Creating Secondary Indexes on 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. Each leaf block in the index structure stores both the key and nonkey columns.

The structure of an index-organized table provides the following benefits:

- Fast random access on the primary key because an index-only scan is sufficient. And, because there is no separate table storage area, changes to the table data (such as adding new rows, updating rows, or deleting rows) result only in updating the index structure.
- Fast range access on the primary key because the rows are clustered in primary key order.

- Lower storage requirements because duplication of primary keys is avoided. They are not stored both in the index and underlying table, as is true with heap-organized tables.

Index-organized tables have full table functionality. They support features such as constraints, triggers, LOB and object columns, partitioning, parallel operations, online reorganization, and replication. And, they offer these additional features:

- Key compression
- Overflow storage area and specific column placement
- Secondary indexes, including bitmap indexes.

Index-organized tables are ideal for OLTP applications, which require fast primary key access and high availability. Queries and DML on an orders table used in electronic order processing are predominantly primary-key based and heavy volume causes fragmentation resulting in a frequent need to reorganize. Because an index-organized table can be reorganized online and without invalidating its secondary indexes, the window of unavailability is greatly reduced or eliminated.

Index-organized tables are 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. A fundamental component of an internet search engine is an inverted index that can be modeled using index-organized tables.

These are but a few of the applications for index-organized tables.

See Also:

- *Oracle Database Concepts* for a more thorough description of index-organized tables
- *Oracle Database VLDB and Partitioning Guide* for information about partitioning index-organized tables

Creating Index-Organized Tables

You use the `CREATE TABLE` statement to create index-organized tables, but you must provide 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).

Optionally, you can specify the following:

- An `OVERFLOW` clause, which preserves dense clustering of the B-tree index by enabling the storage of some of the nonkey columns in a separate overflow data segment.
- A `PCTTHRESHOLD` value, which, when an overflow segment is being used, defines the maximum size of the portion of the row that is stored in the index block, as a percentage of block size. Rows columns that would cause the row size to exceed this maximum are stored in the overflow segment. 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.

- An INCLUDING clause, which can be used to specify the nonkey columns that are to be stored in the index block with the primary key.

Example: Creating an Index-Organized Table

The following statement creates an index-organized table:

```
CREATE TABLE admin_docindex(
        token char(20),
        doc_id NUMBER,
        token_frequency NUMBER,
        token_offsets VARCHAR2(2000),
        CONSTRAINT pk_admin_docindex PRIMARY KEY (token, doc_id))
    ORGANIZATION INDEX
    TABLESPACE admin_tbs
    PCTTHRESHOLD 20
    OVERFLOW TABLESPACE admin_tbs2;
```

This example creates an index-organized table named admin_docindex, with a primary key composed of the columns token and doc_id. The OVERFLOW and PCTTHRESHOLD clauses specify that if the length of a row exceeds 20% of the index block size, then the column that exceeded that threshold and all columns after it are moved to the overflow segment. The overflow segment is stored in the admin_tbs2 tablespace.

> **See Also:** *Oracle Database SQL Language Reference* for more information about the syntax to create an index-organized table

Restrictions for Index-Organized Tables

The following are restrictions on creating index-organized tables.

- The maximum number of columns is 1000.
- The maximum number of columns in the index portion of a row is 255, including both key and nonkey columns. If more than 255 columns are required, you must use an overflow segment.
- The maximum number of columns that you can include in the primary key is 32.
- PCTTHRESHOLD must be in the range of 1–50. The default is 50.
- All key columns must fit within the specified threshold.
- If the maximum size of a row exceeds 50% of the index block size and you do not specify an overflow segment, the CREATE TABLE statement fails.
- Index-organized tables cannot have virtual columns.

Creating Index-Organized Tables that Contain Object Types

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;
```

Another example, that follows, shows that index-organized tables store nested tables efficiently. For a nested table column, the database internally creates a storage table to hold all the nested table rows.

```
CREATE TYPE project_t AS OBJECT(pno NUMBER, pname VARCHAR2(80));
/
CREATE TYPE project_set AS TABLE OF project_t;
/
CREATE TABLE proj_tab (eno NUMBER, projects PROJECT_SET)
    NESTED TABLE projects STORE AS emp_project_tab
                ((PRIMARY KEY(nested_table_id, pno))
    ORGANIZATION INDEX)
    RETURN AS LOCATOR;
```

The rows belonging to a single nested table instance are identified by a `nested_table_id` column. If an ordinary table is used to store nested table columns, the nested table rows typically get de-clustered. But when you use an index-organized table, the nested table rows can be clustered based on the `nested_table_id` column.

See Also:

- *Oracle Database SQL Language Reference* for details of the syntax used for creating index-organized tables
- *Oracle Database VLDB and Partitioning Guide* for information about creating partitioned index-organized tables
- *Oracle Database Object-Relational Developer's Guide* for information about object types

Choosing and Monitoring a Threshold Value

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:

- "Listing Chained Rows of Tables and Clusters" on page 18-4 for more information about chained rows
- *Oracle Database SQL Language Reference* for syntax 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. The database accommodates all nonkey columns up to and including 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 segment. If the `INCLUDING` and `PCTTHRESHOLD` clauses conflict, `PCTTHRESHOLD` takes precedence.

Note: Oracle Database moves all primary key columns of an indexed-organized table to the beginning of the table (in their key 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 following `CREATE TABLE` statement is similar to the one shown earlier in "Example: Creating an Index-Organized Table" on page 20-57 but is 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(2000),
        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.

Parallelizing Index-Organized Table Creation

The `CREATE TABLE...AS SELECT` statement enables you to create an index-organized table and load data from an existing table into it. By including the `PARALLEL` clause, the load can be done in parallel.

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
     AS SELECT * FROM hr.jobs;
```

This statement provides an alternative to parallel bulk-load using SQL*Loader.

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;
```

One application of key compression is in a time-series application that uses a set of time-stamped rows belonging to a single item, such as a stock price. Index-organized tables are attractive for such applications because of the ability to cluster rows based on the primary key. By defining an index-organized table with primary key (stock symbol, time stamp), you can store and manipulate time-series data efficiently. You can achieve more storage savings by compressing repeated occurrences of the item identifier (for example, the stock symbol) in a time series by using an index-organized table with key compression.

> **See Also:** *Oracle Database Concepts* for more information about key compression

Maintaining Index-Organized Tables

Index-organized tables differ from ordinary tables only in physical organization. Logically, they are manipulated in the same manner as ordinary tables. You can specify an index-organized table just as you would specify a regular table in INSERT, SELECT, DELETE, and UPDATE statements.

Altering Index-Organized Tables

All of the alter options available for ordinary tables are available for index-organized tables. This includes ADD, MODIFY, and DROP COLUMNS and CONSTRAINTS. However, the primary key constraint for an index-organized table cannot be dropped, deferred, or disabled

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 INITRANS of the primary key index segment to 4 and the overflow of the data segment INITRANS to 6 as follows:

```
ALTER TABLE admin_docindex INITRANS 4 OVERFLOW INITRANS 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 PCTHRESHOLD 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 operation 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 statement, an index-organized table with a LOB column (CLOB) is created. Later, the table is moved with the LOB index and data segment being 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);
```

> **See Also:** *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about LOBs in index-organized tables

Creating Secondary Indexes on Index-Organized Tables

You can create secondary indexes on an index-organized tables to provide multiple access paths. Secondary indexes on index-organized tables differ from indexes on ordinary tables in two ways:

- They store logical rowids instead of physical rowids. This is necessary because the inherent movability of rows in a B-tree index results in the rows having no permanent physical addresses. If the physical location of a row changes, its logical rowid remains valid. One effect of this is that a table maintenance operation, such as `ALTER TABLE ... MOVE`, does not make the secondary index unusable.
- The logical rowid also includes a physical guess which identifies the database block address at which the row is likely to be found. If the physical guess is correct, a secondary index scan would incur a single additional I/O once the secondary key is found. The performance would be similar to that of a secondary index-scan on an ordinary table.

Unique and non-unique secondary indexes, function-based secondary indexes, and bitmap indexes are supported as secondary indexes on index-organized tables.

Syntax for Creating the Secondary Index

The following statement shows the creation of a secondary index on the `docindex` index-organized table where `doc_id` and `token` are the key columns:

```
CREATE INDEX Doc_id_index on Docindex(Doc_id, Token);
```

This secondary index allows the database to efficiently process a query, such as the following, the involves a predicate on `doc_id`:

```
SELECT Token FROM Docindex WHERE Doc_id = 1;
```

Maintaining Physical Guesses in Logical Rowids

A logical rowid can include a guess, which identifies the block location of a row at the time the guess is made. Instead of doing a full key search, the database uses the guess to search the block directly. However, as new rows are inserted, guesses can become stale. The indexes are still usable through the primary key-component of the logical rowid, but access to rows is slower.

Collect index statistics with the `DBMS_STATS` package to monitor the staleness of guesses. The database checks whether the existing guesses are still valid and records the percentage of rows with valid guesses in the data dictionary. This statistic is stored in the `PCT_DIRECT_ACCESS` column of the `DBA_INDEXES` view (and related views).

To obtain fresh guesses, you can rebuild the secondary index. Note that rebuilding a secondary index on an index-organized table involves reading the base table, unlike rebuilding an index on an ordinary table. A quicker, more light weight means of fixing the guesses is to use the `ALTER INDEX ... UPDATE BLOCK REFERENCES` statement. This statement is performed online, while DML is still allowed on the underlying index-organized table.

After you rebuild a secondary index, or otherwise update the block references in the guesses, collect index statistics again.

Bitmap Indexes

Bitmap indexes on index-organized tables are supported, provided the index-organized table is created with a mapping table. This is done by specifying the `MAPPING TABLE` clause in the `CREATE TABLE` statement that you use to create the index-organized table, or in an `ALTER TABLE` statement to add the mapping table later.

> **See Also:** *Oracle Database Concepts* for a description of mapping tables

Analyzing Index-Organized Tables

Just like ordinary tables, index-organized tables are analyzed using the `DBMS_STATS` package, or the `ANALYZE` statement.

Collecting Optimizer Statistics for Index-Organized Tables

To collect optimizer statistics, use the `DBMS_STATS` package.

For example, the following statement gathers statistics for the index-organized `countries` table in the `hr` schema:

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('HR','COUNTRIES');
```

The `DBMS_STATS` package 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 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';
  ```

> **See Also:**
>
> - *Oracle Database Performance Tuning Guide* for more information about collecting optimizer statistics
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about of the `DBMS_STATS` package

Validating the Structure of Index-Organized Tables

Use the `ANALYZE` statement to validate the structure of your index-organized table or to list any chained rows. These operations are discussed in the following sections located elsewhere in this book:

- "Validating Tables, Indexes, Clusters, and Materialized Views" on page 18-3

- "Listing Chained Rows of Tables and Clusters" on page 18-4

> **Note:** There are special considerations when listing chained rows for index-organized tables. These are discussed in the *Oracle Database SQL Language Reference*.

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 (heap organized) 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:** *Oracle Database Utilities* for more details about using the original `IMP` and `EXP` utilities and the Data Pump import and export utilities

Managing External Tables

This section contains:

- About External Tables
- Creating External Tables
- Altering External Tables
- Preprocessing External Tables

- Dropping External Tables
- System and Object Privileges for External Tables

About External Tables

Oracle Database 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 the database with metadata describing an external table, the database 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.

External tables provide a framework to unload the result of an arbitrary `SELECT` statement into a platform-independent Oracle-proprietary format that can be used by Oracle Data Pump. External tables provide a valuable means for performing basic extraction, transformation, and loading (ETL) tasks that are common for data warehousing.

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. When you use external tables to unload data, the metadata is automatically created based on the data types in the `SELECT` statement.

Oracle Database provides two access drivers for external tables. The default access driver is `ORACLE_LOADER`, which 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. The second access driver, `ORACLE_DATAPUMP`, lets you unload data—that is, read data from the database and insert it into an external table, represented by one or more external files—and then reload it into an Oracle Database.

External Table Restrictions

The following are restrictions on external tables:

- The `ANALYZE` statement is not supported for gathering statistics for external tables. Use the `DBMS_STATS` package instead.
- Virtual columns are not supported

See Also:

- *Oracle Database Utilities* for information about access drivers
- *Oracle Database Data Warehousing Guide* for information about using external tables for ETL in a data warehousing environment
- *Oracle Database Performance Tuning Guide* for information about using the `DBMS_STATS` package

Creating External Tables

You create external tables using the `CREATE TABLE` statement with an `ORGANIZATION EXTERNAL` clause. This statement creates only metadata in the data dictionary.

> **Note:** External tables cannot have virtual columns.

The following example creates an external table and then uploads the data to a database table. Alternatively, you can unload data through the external table framework by specifying the `AS` *`subquery`* clause of the `CREATE TABLE` statement. External table data pump unload can use only the `ORACLE_DATAPUMP` access driver.

EXAMPLE: Creating an External Table and Loading Data

In this example, the data for the external table resides in the two text files `empxt1.dat` and `empxt2.dat`.

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 named `admin_ext_employees` in the `hr` schema and load data from the external table into the `hr.employees` table.

```
CONNECT  /  AS SYSDBA;
-- Set up directories and grant access to hr
CREATE OR REPLACE DIRECTORY admin_dat_dir
    AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY admin_log_dir
    AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY admin_bad_dir
    AS '/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. Provide the user password (hr) when prompted.
CONNECT 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 Database directory object must be an existing OS directory path.
> - The operating system directory path named in the directory object should not contain any symbolic links in its components.

The `TYPE` specification indicates the access driver of the external table. The access driver is the API that interprets the external data for the database. If you omit the `TYPE` specification, `ORACLE_LOADER` is the default access driver. You must specify the `ORACLE_DATAPUMP` access driver if you specify the `AS` *`subquery`* clause to unload data from one Oracle Database and reload it into the same or a different Oracle Database.

The access parameters, specified in the `ACCESS PARAMETERS` clause, are opaque to the database. These access parameters are defined by the access driver, and are provided to the access driver by the database when the external table is accessed. See *Oracle 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 multiple parallel execution servers 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 datafiles 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 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 Database 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 Database internal data type.

> **See Also:** *Oracle Database SQL Language 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 ALTER TABLE clauses shown in Table 20–4 to change the characteristics of an external table. No other clauses are permitted.

Table 20–4 ALTER TABLE Clauses for External Tables

ALTER TABLE Clause	Description	Example
REJECT LIMIT	Changes the reject limit	`ALTER TABLE admin_ext_employees REJECT LIMIT 100;`
PROJECT COLUMN	Determines how the access driver validates rows in subsequent queries: ■ PROJECT COLUMN REFERENCED: the access driver processes only the columns in the select list of the query. This setting may not provide a consistent set of rows when querying a different column list from the same external table. ■ PROJECT COLUMN ALL: the access driver processes all of the columns defined on the external table. This setting always provides a consistent set of rows when querying an external table. This is the default.	`ALTER TABLE admin_ext_employees PROJECT COLUMN REFERENCED;` `ALTER TABLE admin_ext_employees PROJECT COLUMN ALL;`
DEFAULT DIRECTORY	Changes the default directory specification	`ALTER TABLE admin_ext_employees DEFAULT DIRECTORY admin_dat2_dir;`

Preprocessing External Tables

Caution: There are security implications to consider when using the PREPROCESSOR clause. See *Oracle Database Security Guide* for more information.

External tables can be preprocessed by user-supplied preprocessor programs. By using a preprocessing program, users can use data from a file that is not in a format supported by the driver. For example, a user may want to access data stored in a compressed format. Specifying a decompression program for the ORACLE_LOADER access driver allows the data to be decompressed as the access driver processes the data.

To use the preprocessing feature, you must specify the PREPROCESSOR clause in the access parameters of the ORACLE_LOADER access driver. The preprocessor must be a directory object, and the user accessing the external table must have EXECUTE privileges for the directory object. The following example includes the PREPROCESSOR clause and specifies the directory and preprocessor program.

```
CREATE TABLE sales_transactions_ext
(PROD_ID NUMBER,
 CUST_ID NUMBER,
 TIME_ID DATE,
 CHANNEL_ID CHAR,
 PROMO_ID NUMBER,
 QUANTITY_SOLD NUMBER,
 AMOUNT_SOLD NUMBER(10,2),
 UNIT_COST NUMBER(10,2),
 UNIT_PRICE NUMBER(10,2))
ORGANIZATION external
(TYPE oracle_loader
 DEFAULT DIRECTORY data_file_dir
 ACCESS PARAMETERS
```

```
  (RECORDS DELIMITED BY NEWLINE
   CHARACTERSET US7ASCII
   PREPROCESSOR exec_file_dir:'zcat'
   BADFILE log_file_dir:'sh_sales.bad_xt'
   LOGFILE log_file_dir:'sh_sales.log_xt'
   FIELDS TERMINATED BY "|" LDRTRIM
  ( PROD_ID,
    CUST_ID,
    TIME_ID,
    CHANNEL_ID,
    PROMO_ID,
    QUANTITY_SOLD,
    AMOUNT_SOLD,
    UNIT_COST,
    UNIT_PRICE))
 location ('sh_sales.dat.gz')
)REJECT LIMIT UNLIMITED;
```

The PREPROCESSOR clause is not available for databases that use Oracle Database Vault.

See Also:

- *Oracle Database Utilities* provides information more information about the PREPROCESSOR clause
- *Oracle Database Security Guide* for more information about the security implications of the PREPROCESSOR clause

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.

Tables Data Dictionary Views

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 object tables 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_STATISTICS ALL_TAB_STATISTICS USER_TAB_STATISTICS	These views contain optimizer statistics for tables.
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.
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. They are not populated immediately, but after a time lapse (usually 3 hours).
DBA_ENCRYPTED_COLUMNS USER_ENCRYPTED_COLUMNS ALL_ENCRYPTED_COLUMNS	These views list table columns that are encrypted, and for each column, lists the encryption algorithm in use.

View	Description
DBA_UNUSED_COL_TABS ALL_UNUSED_COL_TABS USER_UNUSED_COL_TABS	These views list tables with unused columns, as marked by the ALTER TABLE ... SET UNUSED statement.
DBA_PARTIAL_DROP_TABS ALL_PARTIAL_DROP_TABS USER_PARTIAL_DROP_TABS	These views list tables that have partially completed DROP COLUMN operations. These operations could be incomplete because the operation was interrupted by the user or a system failure.

Example: Displaying Column Information

Column information, such as name, data type, 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_TYPE, DATA_LENGTH, LAST_ANALYZED
    FROM DBA_TAB_COLUMNS
    WHERE OWNER = 'HR'
    ORDER BY TABLE_NAME;
```

The following is the output from the query:

```
TABLE_NAME           COLUMN_NAME          DATA_TYPE   DATA_LENGTH LAST_ANALYZED
-------------------- -------------------- ---------- ------------ -------------
COUNTRIES            COUNTRY_ID           CHAR                  2 05-FEB-03
COUNTRIES            COUNTRY_NAME         VARCHAR2             40 05-FEB-03
COUNTRIES            REGION_ID            NUMBER               22 05-FEB-03
DEPARTMENTS          DEPARTMENT_ID        NUMBER               22 05-FEB-03
DEPARTMENTS          DEPARTMENT_NAME      VARCHAR2             30 05-FEB-03
DEPARTMENTS          MANAGER_ID           NUMBER               22 05-FEB-03
DEPARTMENTS          LOCATION_ID          NUMBER               22 05-FEB-03
EMPLOYEES            EMPLOYEE_ID          NUMBER               22 05-FEB-03
EMPLOYEES            FIRST_NAME           VARCHAR2             20 05-FEB-03
EMPLOYEES            LAST_NAME            VARCHAR2             25 05-FEB-03
EMPLOYEES            EMAIL                VARCHAR2             25 05-FEB-03
.
.
.
LOCATIONS            COUNTRY_ID           CHAR                  2 05-FEB-03
REGIONS              REGION_ID            NUMBER               22 05-FEB-03
REGIONS              REGION_NAME          VARCHAR2             25 05-FEB-03

51 rows selected.
```

See Also:

- *Oracle Database Reference* for complete descriptions of these views
- *Oracle Database Object-Relational Developer's Guide* for information about object tables
- *Oracle Database Performance Tuning Guide* for information about histograms and generating statistics for tables
- "Analyzing Tables, Indexes, and Clusters" on page 18-2

21

Managing Indexes

In this chapter:

- About Indexes
- Guidelines for Managing Indexes
- Creating Indexes
- Altering Indexes
- Monitoring Space Use of Indexes
- Dropping Indexes
- Indexes Data Dictionary Views

About Indexes

Indexes are optional structures associated with tables and clusters that allow SQL queries 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 Database 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 Database 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 Clusters 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. The database 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.

See Also:

- *Oracle Database Concepts* for an overview of indexes
- Chapter 19, "Managing Space for Schema Objects"

Guidelines for Managing Indexes

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
- Indexes and Deferred Segment Creation
- Estimate Index Size and Set Storage Parameters
- Specify the Tablespace for Each Index
- Consider Parallelizing Index Creation
- Consider Creating Indexes with NOLOGGING
- Understand When to Use Unusable or Invisible Indexes
- Consider Costs and Benefits of Coalescing or Rebuilding Indexes
- Consider Cost Before Disabling or Dropping Constraints

See Also:

- *Oracle Database Concepts* for conceptual information about indexes and indexing, including descriptions of the various indexing schemes offered by Oracle
- *Oracle Database Performance Tuning Guide* and *Oracle Database Data Warehousing Guide* for information about bitmap indexes
- *Oracle Database 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 an 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, the database 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 creator. The amount for each user is determined by the initialization parameter `SORT_AREA_SIZE`. The database 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 direct-path load and an index can be created as data is loaded.

See Also: *Oracle 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 the distribution of the row data in relation to 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.

Columns That Are Suitable for Indexing

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 preceding 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 That Are Not Suitable for Indexing

Columns with the following characteristics are less suitable for indexing:

- There are many nulls in the column and you do not search on the not null values.

`LONG` and `LONG RAW` columns cannot be indexed.

Virtual Columns

You can create unique or non-unique indexes on virtual columns.

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 21-19

Indexes and Deferred Segment Creation

Index segment creation is deferred when the associated table defers segment creation. This is because index segment creation reflects the behavior of the table it is associated with.

See Also: "Understand Deferred Segment Creation" on page 20-9 for further information

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, the undo tablespace, 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 data segment, and all of the index 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.

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

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, the database 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:** *Oracle Database VLDB and Partitioning Guide* for information about using parallel execution

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.

Understand When to Use Unusable or Invisible Indexes

Use unusable or invisible indexes when you want to improve the performance of bulk loads, test the effects of removing an index before dropping it, or otherwise suspend the use of an index by the optimizer.

Unusable indexes

An **unusable index** is ignored by the optimizer and is not maintained by DML. One reason to make an index unusable is to improve bulk load performance. (Bulk loads go more quickly if the database does not need to maintain indexes when inserting rows.) Instead of dropping the index and later re-creating it, which requires you to recall the exact parameters of the `CREATE INDEX` statement, you can make the index unusable, and then rebuild it.

You can create an index in the unusable state, or you can mark an existing index or index partition unusable. In some cases the database may mark an index unusable, such as when a failure occurs while building the index. When one partition of a partitioned index is marked unusable, the other partitions of the index remain valid.

An unusable index or index partition must be rebuilt, or dropped and re-created, before it can be used. Truncating a table makes an unusable index valid.

Beginning with Oracle Database 11*g* Release 2, when you make an existing index unusable, its index segment is dropped.

The functionality of unusable indexes depends on the setting of the `SKIP_UNUSABLE_INDEXES` initialization parameter. When `SKIP_UNUSABLE_INDEXES` is `TRUE` (the default), then:

- DML statements against the table proceed, but unusable indexes are not maintained.
- DML statements terminate with an error if there are any unusable indexes that are used to enforce the `UNIQUE` constraint.
- For nonpartitioned indexes, the optimizer does not consider any unusable indexes when creating an access plan for `SELECT` statements. The only exception is when an index is explicitly specified with the `INDEX()` hint.
- For a partitioned index where one or more of the partitions are unusable, the optimizer does not consider the index if it cannot determine at query compilation time if any of the index partitions can be pruned. This is true for both partitioned and nonpartitioned tables. The only exception is when an index is explicitly specified with the `INDEX()` hint.

When `SKIP_UNUSABLE_INDEXES` is `FALSE`, then:

- If any unusable indexes or index partitions are present, any DML statements that would cause those indexes or index partitions to be updated are terminated with an error.
- For `SELECT` statements, if an unusable index or unusable index partition is present but the optimizer does not choose to use it for the access plan, the statement proceeds. However, if the optimizer does choose to use the unusable index or unusable index partition, the statement terminates with an error.

Invisible Indexes

Beginning with Oracle Database 11*g* Release 1, you can create invisible indexes or make an existing index invisible. An **invisible index** is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization

parameter to TRUE at the session or system level. Unlike unusable indexes, an invisible index is maintained during DML statements. Although you can make a partitioned index invisible, you cannot make an individual index partition invisible while leaving the other partitions visible.

Using invisible indexes, you can do the following:

- Test the removal of an index before dropping it.
- Use temporary index structures for certain operations or modules of an application without affecting the overall application.

See Also:

- "Creating an Unusable Index" on page 21-13
- "Creating an Invisible Index" on page 21-14
- "Making an Index Unusable" on page 21-17
- "Making an Index Invisible" on page 21-18

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 21–1 is a comparison of the costs and benefits associated with rebuilding and coalescing indexes.

Table 21–1 Costs and Benefits of Coalescing or Rebuilding Indexes

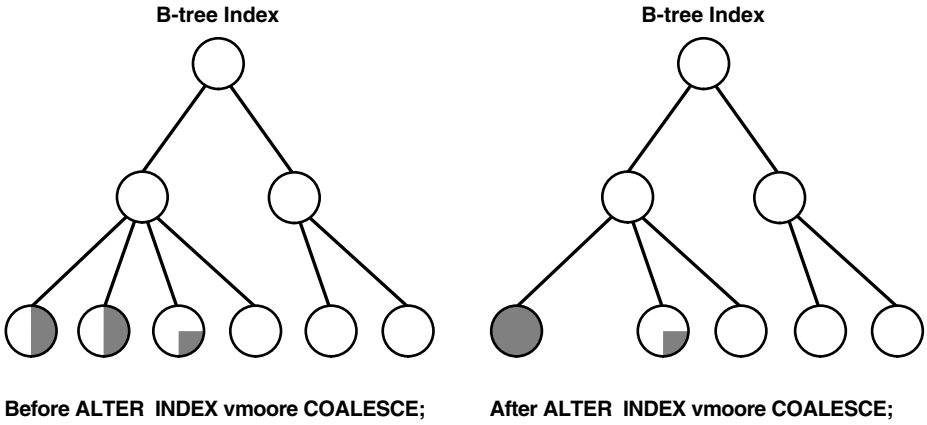
Rebuild Index	Coalesce Index
Quickly moves index to another tablespace	Cannot move index to another tablespace
Higher costs: requires more disk space	Lower costs: does not require more disk space
Creates new tree, shrinks height if applicable	Coalesces leaf blocks within same branch of tree
Enables you 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 21–1 illustrates the effect of an ALTER INDEX COALESCE on the index vmoore. Before performing the operation, the first two leaf blocks are 50% full. Therefore, you have an opportunity to reduce fragmentation and completely fill the first block, while freeing up the second.

Figure 21–1 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 18-9

Creating Indexes

This section describes how to create indexes.

Note: These operations also collect index statistics.

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
- Creating a Large Index

- Creating an Index Online
- Creating a Function-Based Index
- Creating a Key-Compressed Index
- Creating an Unusable Index
- Creating an Invisible Index

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);
```

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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and restrictions on the use of the `CREATE INDEX` statement

Creating a Unique Index Explicitly

Indexes can be unique or non-unique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns). Non-unique 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. The database 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:** *Oracle Database Performance Tuning Guide* for more information about creating an index for performance

Creating an Index Associated with a Constraint

Oracle Database 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 the database 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. The index associated with a constraint always takes the name of the constraint, unless you optionally specify otherwise.

Note: An efficient procedure for enabling a constraint that can make use of parallelism is described in "Efficient Use of Integrity Constraints: A Procedure" on page 18-11.

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 storage options of the associated index:

```
CREATE TABLE emp (
     empno NUMBER(5) PRIMARY KEY, age INTEGER)
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE users;
```

Specifying the Index Associated with a Constraint

If you require more explicit control over the indexes associated with UNIQUE and PRIMARY KEY constraints, the database lets you:

- Specify an existing index that the database is to use to enforce the constraint
- Specify a CREATE INDEX statement that the database 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 18-9

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. Therefore, you can 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: Keep in mind that the time that it takes on online index build to complete is proportional to the size of the table and the number of concurrently executing DML statements. Therefore, it is best to start online index builds when DML activity is low.

See Also: "Rebuilding an Existing Index" on page 21-16

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.

In addition to the prerequisites for creating a conventional index, if the index is based on user-defined functions, then those functions must be marked `DETERMINISTIC`. Also, you just have the `EXECUTE` object privilege on any user-defined function(s) used in the function-based index if those functions are owned by another user.

> **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, 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 lets you disable the use of a function-based index. Consider doing this if you are working on the body of the function.

> **Note:** An alternative to creating a function-based index is to add a virtual column to the target table and index the virtual column. See "About Tables" on page 20-1 for more information.

See Also:

- *Oracle Database Concepts* for more information about function-based indexes
- *Oracle Database Advanced Application Developer's Guide* for information about using function-based indexes in applications and examples of their use

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 multicolumn 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:** *Oracle Database Concepts* for a more detailed discussion of key compression

Creating an Unusable Index

When you create an index in the UNUSABLE state, it is ignored by the optimizer and is not maintained by DML. An unusable index must be rebuilt, or dropped and re-created, before it can be used.

If the index is partitioned, then all index partitions are marked UNUSABLE.

Beginning with Oracle Database 11*g* Release 2, the database does not create an index segment when creating an unusable index.

The following procedure illustrates how to create unusable indexes and query the database for details about the index.

To create an unusable index:

1. If necessary, create the table to be indexed.

   For example, create a hash-partitioned table called hr.employees_part as follows:

   ```
   sh@PROD> CONNECT hr
   Enter password: **
   Connected.

   hr@PROD> CREATE TABLE employees_part
     2    PARTITION BY HASH (employee_id) PARTITIONS 2
     3    AS SELECT * FROM employees;

   Table created.

   hr@PROD> SELECT COUNT(*) FROM employees_part;

     COUNT(*)
   ----------
         107
   ```

2. Create an index with the keyword UNUSABLE.

   The following example creates a locally partitioned index on employees_part, naming the index partitions p1_i_emp_ename and p2_i_emp_ename, and making p1_i_emp_ename unusable:

```
hr@PROD> CREATE INDEX i_emp_ename ON employees_part (employee_id)
  2    LOCAL (PARTITION p1_i_emp_ename UNUSABLE, PARTITION p2_i_emp_ename);

Index created.
```

3. Optionally, verify that the index is unusable by querying the data dictionary.

   The following example queries the status of index `i_emp_ename` and its two partitions, showing that only partition `p2_i_emp_ename` is unusable:

```
hr@PROD> SELECT INDEX_NAME AS "INDEX OR PARTITION NAME", STATUS
  2  FROM   USER_INDEXES
  3  WHERE  INDEX_NAME = 'I_EMP_ENAME'
  4  UNION ALL
  5  SELECT PARTITION_NAME AS "INDEX OR PARTITION NAME", STATUS
  6  FROM   USER_IND_PARTITIONS
  7  WHERE  PARTITION_NAME LIKE '%I_EMP_ENAME%';

INDEX OR PARTITION NAME        STATUS
------------------------------ --------
I_EMP_ENAME                    N/A
P1_I_EMP_ENAME                 UNUSABLE
P2_I_EMP_ENAME                 USABLE
```

4. Optionally, query the data dictionary to determine whether storage exists for the partitions.

   For example, the following query shows that only index partition `p2_i_emp_ename` occupies a segment. Because you created `p1_i_emp_ename` as unusable, the database did not allocate a segment for it.

```
hr@PROD> COL PARTITION_NAME FORMAT a14
hr@PROD> COL SEG_CREATED FORMAT a11
hr@PROD> SELECT p.PARTITION_NAME, p.STATUS AS "PART_STATUS",
  2         p.SEGMENT_CREATED AS "SEG_CREATED",
  3  FROM   USER_IND_PARTITIONS p, USER_SEGMENTS s
  4  WHERE  s.SEGMENT_NAME = 'I_EMP_ENAME';

PARTITION_NAME PART_STA SEG_CREATED
-------------- -------- -----------
P2_I_EMP_ENAME USABLE   YES
P1_I_EMP_ENAME UNUSABLE NO
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes" on page 21-6
- "Making an Index Unusable" on page 21-17
- *Oracle Database SQL Language Reference* for more information on creating unusable indexes, including restrictions.

Creating an Invisible Index

An invisible index is an index that is ignored by the optimizer unless you explicitly set the `OPTIMIZER_USE_INVISIBLE_INDEXES` initialization parameter to `TRUE` at the session or system level.

To create an invisible index:

- Use the `CREATE INDEX` statement with the `INVISIBLE` keyword.

The following statement creates an invisible 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)
      INVISIBLE;
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes" on page 21-6
- "Making an Index Invisible" on page 21-18
- *Oracle Database SQL Language Reference* for more information on creating invisible indexes

Altering Indexes

To alter an index, your schema must contain the index or you must have the ALTER ANY INDEX system privilege. With the ALTER INDEX statement, you can:

- 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
- Make the index invisible
- Rename the index
- Start or stop the monitoring of index usage

You cannot alter index 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
- Making an Index Unusable
- Making an Index Invisible
- Monitoring Index Usage

See Also:

- *Oracle Database SQL Language Reference* for details on the ALTER INDEX statement

Altering Storage Characteristics of an Index

Alter the storage parameters of any index, including those created by the database 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 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 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;
```

> **See Also:** *Oracle Database SQL Language Reference* for syntax and restrictions on the use of the `ALTER INDEX` statement

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 21–1 on page 21-7.

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 with the `DEALLOCATE UNUSED` clause.

You have the option of rebuilding the index online. Rebuilding online enables you to update base tables at the same time that you are rebuilding. The following statement rebuilds the `emp_name` index online:

```
ALTER INDEX emp_name REBUILD ONLINE;
```

To rebuild an index in a different user's schema online, the following additional system privileges are required:

- `CREATE ANY TABLE`
- `CREATE ANY INDEX`

> **Note:** Online index rebuilding has stricter limitations on the maximum key length that can be handled, compared to other methods of rebuilding an index. If an ORA-1450 (maximum key length exceeded) error occurs when rebuilding online, try rebuilding offline, coalescing, or dropping and re-creating the index.

If you do not have the space required to rebuild an index, you can choose instead to coalesce the index. Coalescing an index is an online operation.

See Also:

- "Creating an Index Online" on page 21-11
- "Monitoring Space Use of Indexes" on page 21-20

Making an Index Unusable

When you make an index unusable, it is ignored by the optimizer and is not maintained by DML. When you make one partition of a partitioned index unusable, the other partitions of the index remain valid.

You must rebuild or drop and re-create an unusable index or index partition before using it.

The following procedure illustrates how to make an index and index partition unusable, and how to query the object status.

To make an index unusable:

1. Query the data dictionary to determine whether an existing index or index partition is usable or unusable.

   For example, issue the following query (output truncated to save space):

```
hr@PROD> SELECT INDEX_NAME AS "INDEX OR PART NAME", STATUS, SEGMENT_CREATED
  2  FROM   USER_INDEXES
  3  UNION ALL
  4  SELECT PARTITION_NAME AS "INDEX OR PART NAME", STATUS, SEGMENT_CREATED
  5  FROM   USER_IND_PARTITIONS;

INDEX OR PART NAME             STATUS   SEG
------------------------------ -------- ---
I_EMP_ENAME                    N/A      N/A
JHIST_EMP_ID_ST_DATE_PK        VALID    YES
JHIST_JOB_IX                   VALID    YES
JHIST_EMPLOYEE_IX              VALID    YES
JHIST_DEPARTMENT_IX            VALID    YES
EMP_EMAIL_UK                   VALID    NO
.
.
.
COUNTRY_C_ID_PK                VALID    YES
REG_ID_PK                      VALID    YES
P2_I_EMP_ENAME                 USABLE   YES
P1_I_EMP_ENAME                 UNUSABLE NO

22 rows selected.
```

   The preceding output shows that only index partition `p1_i_emp_ename` is unusable.

2. Make an index or index partition unusable by specifying the `UNUSABLE` keyword.

   The following example makes index `emp_email_uk` unusable:

```
hr@PROD> ALTER INDEX emp_email_uk UNUSABLE;

Index altered.
```

The following example makes index partition p2_i_emp_ename unusable:

```
hr@PROD> ALTER INDEX i_emp_ename MODIFY PARTITION p2_i_emp_ename UNUSABLE;

Index altered.
```

3. Optionally, query the data dictionary to verify the status change.

   For example, issue the following query (output truncated to save space):

```
hr@PROD> SELECT INDEX_NAME AS "INDEX OR PARTITION NAME", STATUS,
  2  SEGMENT_CREATED
  3  FROM   USER_INDEXES
  4  UNION ALL
  5  SELECT PARTITION_NAME AS "INDEX OR PARTITION NAME", STATUS,
  6  SEGMENT_CREATED
  7  FROM   USER_IND_PARTITIONS;

INDEX OR PARTITION NAME        STATUS   SEG
------------------------------ -------- ---
I_EMP_ENAME                    N/A      N/A
JHIST_EMP_ID_ST_DATE_PK        VALID    YES
JHIST_JOB_IX                   VALID    YES
JHIST_EMPLOYEE_IX              VALID    YES
JHIST_DEPARTMENT_IX            VALID    YES
EMP_EMAIL_UK                   UNUSABLE NO
.
.
.
COUNTRY_C_ID_PK                VALID    YES
REG_ID_PK                      VALID    YES
P2_I_EMP_ENAME                 UNUSABLE NO
P1_I_EMP_ENAME                 UNUSABLE NO

22 rows selected.
```

   A query of space consumed by the i_emp_ename and emp_email_uk segments shows that the segments no longer exist:

```
hr@PROD> SELECT SEGMENT_NAME, BYTES
  2  FROM   USER_SEGMENTS
  3  WHERE  SEGMENT_NAME IN ('I_EMP_ENAME', 'EMP_EMAIL_UK');

no rows selected
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes" on page 21-6
- "Creating an Unusable Index" on page 21-13
- *Oracle Database SQL Language Reference* for more information about the UNUSABLE keyword, including restrictions

Making an Index Invisible

An invisible index is ignored by the optimizer unless you explicitly set the OPTIMIZER_USE_INVISIBLE_INDEXES initialization parameter to TRUE at the session or system level. Making an index invisible is an alternative to making it unusable or dropping it. You cannot make an individual index partition invisible. Attempting to do so produces an error.

To make an index invisible:

- Submit the following SQL statement:

```
ALTER INDEX index INVISIBLE;
```

To make an invisible index visible again:

- Submit the following SQL statement:

```
ALTER INDEX index VISIBLE;
```

To determine whether an index is visible or invisible:

- Query the dictionary views USER_INDEXES, ALL_INDEXES, or DBA_INDEXES.

  For example, to determine if the index ind1 is invisible, issue the following query:

```
SELECT INDEX_NAME, VISIBILITY FROM USER_INDEXES
   WHERE INDEX_NAME = 'IND1';

INDEX_NAME   VISIBILITY
----------   ----------
IND1         VISIBLE
```

See Also:

- "Understand When to Use Unusable or Invisible Indexes" on page 21-6
- "Creating an Invisible Index" on page 21-14

Renaming an Index

To rename an index, issue this statement:

```
ALTER INDEX index_name RENAME TO new_name;
```

Monitoring Index Usage

Oracle Database provides a means of monitoring indexes to determine whether they are being used. If an index is not being used, then it can be dropped, eliminating unnecessary statement overhead.

To start monitoring the usage of an index, 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 index efficiency of space usage at regular intervals by first analyzing the index 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 index space usage varies according to how often index keys are inserted, updated, or deleted. Develop a history of average efficiency of space usage for an index by performing the following sequence of operations several times:

- Analyzing statistics
- Validating the index
- Checking `PCT_USED`
- Dropping and rebuilding (or coalescing) the index

When you find that index space usage drops below its average, you can condense the index space by dropping the index and rebuilding it, or coalescing it.

See Also: "Analyzing Tables, Indexes, and Clusters" on page 18-2

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 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 constraints associated index, you must disable or drop the constraint itself.

Note: If a table is dropped, all associated indexes are dropped automatically.

See Also:

- *Oracle Database SQL Language Reference* for syntax and restrictions on the use of the `DROP INDEX` statement
- "Managing Integrity Constraints" on page 18-9
- "Making an Index Invisible" on page 21-18 for an alternative to dropping indexes

Indexes Data Dictionary Views

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.
DBA_IND_STATISTICS ALL_IND_STATISTICS USER_IND_STATISTICS	These views contain optimizer statistics for indexes.
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: *Oracle Database Reference* for a complete description of these views

22

Managing Clusters

In this chapter:

- About Clusters
- Guidelines for Managing Clusters
- Creating Clusters
- Altering Clusters
- Dropping Clusters
- Clusters Data Dictionary Views

About 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 22–1), Oracle Database 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:

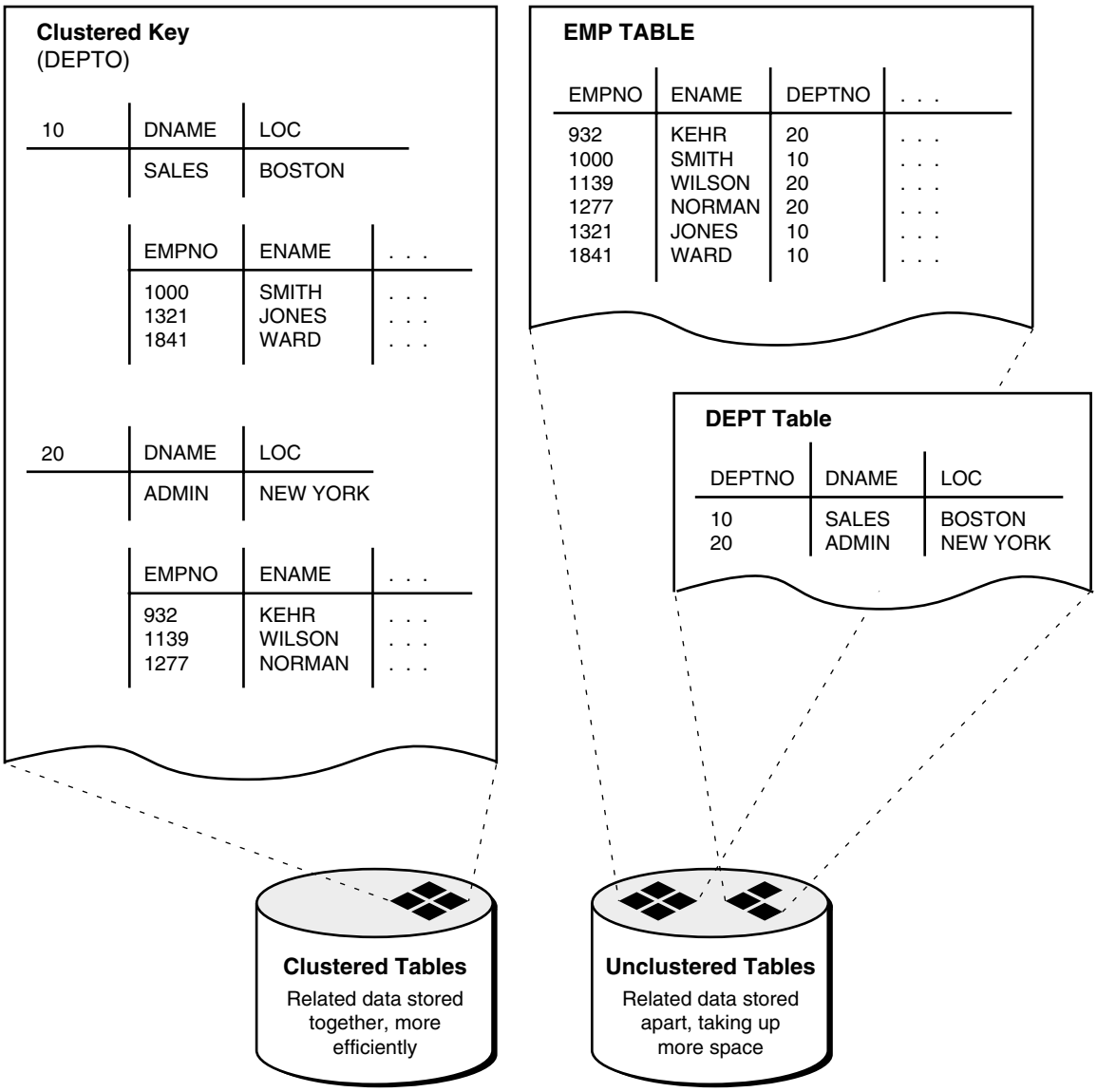
- Disk I/O is reduced and access time improves for joins of clustered tables.
- The **cluster key** is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating 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 22–1, 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 22–1 Clustered Table Data



See Also:

- Chapter 23, "Managing Hash Clusters" for a description of another type of cluster: a hash cluster
- Chapter 19, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter

Guidelines for Managing Clusters

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 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:

- *Oracle Database Concepts* for more information about clusters
- *Oracle Database Performance Tuning Guide* 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 21-2.

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 22-3).

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 the Space Required by an Average Cluster Key and Its Associated Rows

The `CREATE CLUSTER` statement has an optional clause, `SIZE`, which is the estimated number of bytes required by an average cluster key and its associated rows. The database 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, the database stores only one cluster key and its associated rows in each data block of the cluster 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 systems.

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 multiple keys fit in a block, then blocks can belong to multiple chains.

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 clause 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 cluster size before creating the cluster:

- You can use the combined estimated size of clusters, along with estimates for indexes 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.

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 into the cluster. Storage parameters specified when creating or altering a clustered table are ignored. The storage parameters set for the cluster override the table storage parameters.

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))
   SIZE 600
   TABLESPACE users
   STORAGE (INITIAL 200K
      NEXT 300K
      MINEXTENTS 2
      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 23, "Managing Hash Clusters".

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 clause. 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.

See Also: *Oracle Database SQL Language Reference* for syntax of the CREATE TABLE statement for creating cluster tables

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
   TABLESPACE users
   STORAGE (INITIAL 50K
      NEXT 50K
      MINEXTENTS 2
      MAXEXTENTS 10
      PCTINCREASE 33);
```

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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax of the CREATE INDEX statement for creating cluster indexes

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 (INITRANS 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. The database 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 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 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 setting INITRANS of a cluster, the new setting for INITRANS applies only to data blocks subsequently allocated for the 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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax of the ALTER CLUSTER statement

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). The database 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 20-26.

> **See Also:** *Oracle Database SQL Language Reference* for syntax of the ALTER TABLE statement

Altering Cluster Indexes

You alter cluster indexes exactly as you do other indexes. See "Altering Indexes" on page 21-15.

> **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 needed. When a cluster is dropped, so are the tables within the cluster and the corresponding cluster index. All extents belonging to both the cluster 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 clause of the DROP CLUSTER statement, as follows:

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

If the INCLUDING TABLES clause 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 clause of the DROP CLUSTER statement, as shown in the following example:

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

The database returns an error if you do not use the CASCADE CONSTRAINTS clause and constraints exist.

> **See Also:** *Oracle Database SQL Language Reference* for syntax of the DROP CLUSTER statement

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 cluster, other clustered tables, or the cluster index. A clustered table is dropped just as a nonclustered table is dropped, with the `DROP TABLE` statement. See "Dropping Table Columns" on page 20-30.

Note: When you drop a single table from a cluster, the database 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` clause. 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 21-20.

Clusters Data Dictionary Views

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: *Oracle Database Reference* for complete descriptions of these views

23

Managing Hash Clusters

In this chapter:

- About Hash Clusters
- When to Use Hash Clusters
- Creating Hash Clusters
- Altering Hash Clusters
- Dropping Hash Clusters
- Hash Clusters Data Dictionary Views

About 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 non-clustered table with an index or an index cluster. With an indexed table or index cluster, Oracle Database locates the rows in a table using key values that the database stores in a separate index. To use hashing, you create a hash cluster and load tables into it. The database physically stores the rows of a table in a hash cluster and retrieves them according to the results of a **hash function**.

Oracle Database 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, the database applies the hash function to the cluster key value of the row. The resulting hash value corresponds to a data block in the cluster, which the database 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, the database 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.

> **See Also:** Chapter 19, "Managing Space for Schema Objects" is recommended reading before attempting tasks described in this chapter.

When to Use Hash Clusters

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.

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

You create a hash cluster using a `CREATE CLUSTER` statement, but you specify a `HASHKEYS` clause. The following statement creates a cluster named `trial_cluster`, clustered by the `trialno` column (the cluster key):

```
CREATE CLUSTER trial_cluster ( trialno NUMBER(5,0) )
```

```
    TABLESPACE users
    STORAGE ( INITIAL 250K
              NEXT 50K
              MINEXTENTS 1
              MAXEXTENTS 3
              PCTINCREASE 0 )
    HASH IS trialno
    HASHKEYS 150;
```

The following statement creates the `trial` table in the `trial_cluster` hash cluster:

```
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 the preceding example, the key is the `trialno` column.

The `HASHKEYS` value, in this case `150`, specifies and limits the number of unique hash values that the hash function can generate. The database rounds the number specified to the nearest prime number.

If no `HASH IS` clause is specified, then the database uses an internal hash function. If the cluster key is already a unique identifier that is uniformly distributed over its range, then you can bypass the internal hash function and specify the cluster key as the hash value, as in the preceding 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.

The following sections explain and provide guidelines for setting the parameters of the `CREATE CLUSTER` statement specific to hash clusters:

- Creating a Sorted Hash Cluster
- Creating Single-Table Hash Clusters
- Controlling Space Use Within a Hash Cluster
- Estimating Size Required by Hash Clusters

> **See Also:** Chapter 22, "Managing Clusters" 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

Creating a Sorted Hash Cluster

A **sorted hash cluster** stores the rows corresponding to each value of the hash function in such a way that the database can efficiently return them in sorted order. For applications that always consume data in sorted order, sorted hash clusters can retrieve data faster by minimizing logical I/Os.

Assume that a telecommunications company stores detailed call records for a fixed number of originating telephone numbers through a telecommunications switch. From each originating telephone number there can be an unlimited number of calls.

The application stores calls records as calls are made. Each call has a detailed call record identified by a timestamp. For example, the application stores a call record with timestamp 0, then a call record with timestamp 1, and so on.

When generating bills for each originating phone number, the application processes them in first-in, first-out (FIFO) order. The following table shows sample details for three originating phone numbers:

telephone_number	call_timestamp
6505551212	0, 1, 2, 3, 4, ...
6505551213	0, 1, 2, 3, 4, ...
6505551214	0, 1, 2, 3, 4, ...

In the following SQL statements, the `telephone_number` column is the hash key. The hash cluster is sorted on the `call_timestamp` and `call_duration` columns. The example uses the same names for the clustering and sorting columns in the table definition as in the cluster definition, but this is not required. The number of hash keys is based on 10-digit telephone numbers.

```
CREATE CLUSTER call_detail_cluster (
   telephone_number NUMBER,
   call_timestamp   NUMBER SORT,
   call_duration    NUMBER SORT )
  HASHKEYS 10000
  HASH IS telephone_number
  SIZE 256;

CREATE TABLE call_detail (
   telephone_number     NUMBER,
   call_timestamp       NUMBER   SORT,
   call_duration        NUMBER   SORT,
   other_info           VARCHAR2(30) )
  CLUSTER call_detail_cluster (
   telephone_number, call_timestamp, call_duration );
```

Suppose that you seed the `call_detail` table with the rows in FIFO order as shown in Example 23–1.

Example 23–1 Data Inserted in Sequential Order

```
INSERT INTO call_detail VALUES (6505551212, 0, 9, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 1, 17, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 2, 5, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 3, 90, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 0, 35, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 1, 6, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 2, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 3, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 0, 15, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 1, 20, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 2, 1, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 3, 25, 'misc info');
COMMIT;
```

In Example 23–2, you `SET AUTOTRACE ON`, and then query the `call_detail` table for the call details for the phone number `6505551212`.

Example 23–2 Querying call_detail

```
SQL> SET AUTOTRACE ON;
SQL> SELECT * FROM call_detail WHERE telephone_number = 6505551212;
```

```
TELEPHONE_NUMBER CALL_TIMESTAMP CALL_DURATION OTHER_INFO
---------------- -------------- ------------- ------------------------------
      6505551212              0             9 misc info
      6505551212              1            17 misc info
      6505551212              2             5 misc info
      6505551212              3            90 misc info

Execution Plan
----------------------------------------------------------
Plan hash value: 2118876266

---------------------------------------------------------------------
| Id  | Operation         | Name        | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------
|   0 | SELECT STATEMENT  |             |     1 |    56 |     0   (0)|
|*  1 |  TABLE ACCESS HASH| CALL_DETAIL |     1 |    56 |            |
---------------------------------------------------------------------
```

Example 23–2 shows that the query retrieves the rows ordered by timestamp even though no sort appears in the query plan.

Suppose you then delete the existing rows and insert the same rows out of sequence:

```
DELETE FROM call_detail;
INSERT INTO call_detail VALUES (6505551213, 3, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 0, 15, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 0, 9, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 1, 20, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 2, 1, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 1, 6, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 2, 4, 'misc info');
INSERT INTO call_detail VALUES (6505551214, 3, 25, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 1, 17, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 2, 5, 'misc info');
INSERT INTO call_detail VALUES (6505551212, 3, 90, 'misc info');
INSERT INTO call_detail VALUES (6505551213, 0, 35, 'misc info');
COMMIT;
```

If you rerun the same query of `call_detail`, the database again retrieves the rows in sorted order even though no `ORDER BY` clause is specified. No `SORT ORDER BY` operation appears in the query plan because the database performs an internal sort.

Now assume that you create a nonclustered table `call_detail_nonclustered` and then load it with the same sample values in Example 23–1. To retrieve the data in sorted order, you must use an `ORDER BY` clause as follows:

```
SQL> SELECT * FROM call_detail_nonclustered WHERE telephone_number = 6505551212
  2  ORDER BY call_timestamp, call_duration;

TELEPHONE_NUMBER CALL_TIMESTAMP CALL_DURATION OTHER_INFO
---------------- -------------- ------------- ------------------------------
      6505551212              0             9 misc info
      6505551212              1            17 misc info
      6505551212              2             5 misc info
      6505551212              3            90 misc info

Execution Plan
----------------------------------------------------------
Plan hash value: 2555750302
```

```
-------------------------------------------------------------------------------
|Id| Operation          | Name                     |Rows|Bytes|Cost (%CPU)|Time   |
-------------------------------------------------------------------------------
| 0| SELECT STATEMENT   |                          | 4  | 224 | 4 (25)| 00:00:01 |
| 1|  SORT ORDER BY     |                          | 4  | 224 | 4 (25)| 00:00:01 |
|*2|   TABLE ACCESS FULL| CALL_DETAIL_NONCLUSTERED | 4  | 224 | 3  (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

The preceding plan shows that in the nonclustered case the sort is more expensive than in the clustered case. The rows, bytes, cost, and time are all greater in the case of the table that is not stored in a sorted hash cluster.

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;
```

The database rounds the `HASHKEYS` value up to the nearest prime number, so this cluster has a maximum of 503 hash key values, each of size 512 bytes. The `SINGLE TABLE` clause is valid only for hash clusters. `HASHKEYS` must also be specified.

> **See Also:** *Oracle Database SQL Language Reference* for the syntax of the `CREATE CLUSTER` statement

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 the internal hash function of the database.

Setting HASH IS

Specify the `HASH IS` parameter only if the cluster key is a single column of the `NUMBER` data type, and contains uniformly distributed integers. If these 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 clause 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 adjustments shown in the preceding table and use the original value for SIZE.

Setting HASHKEYS

For maximum distribution of rows in a hash cluster, the database rounds the HASHKEYS value up to the nearest prime number.

Controlling Space in Hash Clusters

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).

Controlling Space in Hash Clusters: 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. Because 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. The database rounds this value up to the next highest prime number: 10007.

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```

Controlling Space in Hash Clusters: 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 (see "Setting SIZE" on page 23-7), there is more space for rows from expected collisions.

HASHKEYS can be set to the number of unique department numbers, 1000. The database 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 Database 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 22-6. 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 22-7

Hash Clusters Data Dictionary Views

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: *Oracle Database Reference* for complete descriptions of these views

24

Managing Views, Sequences, and Synonyms

In this chapter:

- Managing Views
- Managing Sequences
- Managing Synonyms
- Views, Synonyms, and Sequences Data Dictionary Views

Managing Views

This section describes aspects of managing views, and contains the following topics:

- About Views
- Creating Views
- Replacing Views
- Using Views in Queries
- Updating a Join View
- Altering Views
- Dropping Views

About Views

A **view** is a logical representation of a table or combination of tables. In essence, a view is a stored query. A view derives its data from the tables on which it is based. These tables are called **base tables**. Base tables might in turn be actual tables or might be views themselves. All operations performed on a view actually affect the base table of the view. You can use views in almost the same way as tables. You can query, update, insert into, and delete from views, just as you can standard tables.

Views can provide a different representation (such as subsets or supersets) of the data that resides within other tables and views. Views are very powerful because they allow you to tailor the presentation of data to different types of users.

Note: One special type of view is the editioning view, which is used only to support online upgrade of applications using edition-based redefinition. The remainder of this section on managing views describes all views except editioning views. See *Oracle Database Advanced Application Developer's Guide* for a discussion of editioning views and edition-based redefinition.

See Also: *Oracle Database Concepts* for an overview of 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 depends on the privileges of the view owner. For example, if the owner of the view has only the `INSERT` privilege for Scott's `emp` table, then the view can be used only 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 with 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 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:** *Oracle Database SQL Language Reference* for syntax and semantics of the CREATE VIEW statement

Join Views

You can also create views that specify multiple base tables or views 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 24-6 for further discussion.

Expansion of Defining Queries at View Creation Time

When a view is created, Oracle Database 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 quotation 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;
```

The database 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, the database 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 clause of the CREATE VIEW statement.

```
CREATE FORCE VIEW AS ...;
```

By default, views with errors are created as INVALID. When you try to create such a view, the database returns a message indicating the view was created with errors. 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 18-17.

Replacing Views

To replace a view, you must have all of the privileges required to drop and create a view. If the definition of a view must change, the view must be replaced; you cannot use an ALTER VIEW statement to 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 regranted.

- You can redefine the view with a CREATE VIEW statement that contains the OR REPLACE clause. The OR REPLACE clause 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 must 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 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 dependent on a replaced view become invalid (not usable). In addition, dependent PL/SQL program units may become invalid, depending on what was changed in the new version of the view. For example, if only the WHERE clause of the view changes, dependent PL/SQL program units remain valid. However, if any changes are made to the number of view columns or to the view column names or data types, dependent PL/SQL program units are invalidated. See "Managing Object Dependencies" on page 18-17 for more information on how the database manages such dependencies.

Using Views in Queries

To issue a query or an INSERT, UPDATE, or DELETE statement against a view, you must have the SELECT, INSERT, UPDATE, or DELETE object privilege for the view, respectively, either explicitly or through a role.

Views can be queried in the same manner as tables. For example, to query the Division1_staff view, enter a valid SELECT statement that references the view:

```
SELECT * FROM Division1_staff;

ENAME        EMPNO       JOB             DNAME
-----------------------------------------------------
CLARK         7782       MANAGER         ACCOUNTING
```

```
KING          7839       PRESIDENT       ACCOUNTING
MILLER        7934       CLERK           ACCOUNTING
ALLEN         7499       SALESMAN        SALES
WARD          7521       SALESMAN        SALES
JAMES         7900       CLERK           SALES
TURNER        7844       SALESMAN        SALES
MARTIN        7654       SALESMAN        SALES
BLAKE         7698       MANAGER         SALES
```

With some restrictions, rows can be inserted into, updated in, or deleted from a base table using a view. The following statement inserts a new row into the `emp` table using the `sales_staff` view:

```
INSERT INTO sales_staff
    VALUES (7954, 'OSTER', 30);
```

Restrictions on DML operations for views use the following criteria in the order listed:

1. If a view is defined by a query that contains `SET` or `DISTINCT` operators, a `GROUP BY` clause, or a group function, then rows cannot be inserted into, updated in, or deleted from the base tables using the view.
2. If a view is defined with `WITH CHECK OPTION`, a row cannot be inserted into, or updated in, the base table (using the view), if the view cannot select the row from the base table.
3. If a `NOT NULL` column that does not have a `DEFAULT` clause is omitted from the view, then a row cannot be inserted into the base table using the view.
4. If the view was created by using an expression, such as `DECODE(deptno, 10, "SALES", ...)`, then rows cannot be inserted into or updated in the base table using the view.

The constraint created by `WITH CHECK OPTION` of the `sales_staff` view only allows rows that have a department number of 30 to be inserted into, or updated in, the `emp` table. Alternatively, assume that the `sales_staff` view is defined by the following statement (that is, excluding the `deptno` column):

```
CREATE VIEW sales_staff AS
    SELECT empno, ename
    FROM emp
    WHERE deptno = 10
    WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

Considering this view definition, you can update the `empno` or `ename` fields of existing records, but you cannot insert rows into the `emp` table through the `sales_staff` view because the view does not let you alter the `deptno` field. If you had defined a `DEFAULT` value of 10 on the `deptno` field, then you could perform inserts.

When a user attempts to reference an invalid view, the database returns an error message to the user:

```
ORA-04063: view 'view_name' has errors
```

This error message is returned when a view exists but is unusable due to errors in its query (whether it had errors when originally created or it was created successfully but became unusable later because underlying objects were altered or dropped).

Updating a Join View

An updatable join view (also referred to as a **modifiable join view**) is a view that contains multiple tables in the top-level FROM clause of the SELECT statement, and is not restricted by the WITH READ ONLY clause.

The rules for updatable join views are shown in the following table. Views that meet these criteria are said to be inherently updatable.

Rule	Description
General Rule	Any INSERT, UPDATE, or DELETE operation on a join view can modify only one underlying base table at a time.
UPDATE Rule	All updatable columns of a join view must map to columns of a **key-preserved table**. See "Key-Preserved Tables" on page 24-7 for a discussion of key-preserved tables. If the view is defined with the WITH CHECK OPTION clause, then all join columns and all columns of repeated tables are not updatable.
DELETE Rule	Rows from a join view can be deleted as long as there is exactly one key-preserved table in the join. The key preserved table can be repeated in the FROM clause. If the view is defined with the WITH CHECK OPTION clause and the key preserved table is repeated, then the rows cannot be deleted from the view.
INSERT Rule	An INSERT statement must not explicitly or implicitly refer to the columns of a **non-key-preserved table**. If the join view is defined with the WITH CHECK OPTION clause, INSERT statements are not permitted.

There are data dictionary views that indicate whether the columns in a join view are inherently updatable. See "Using the UPDATABLE_ COLUMNS Views" on page 24-11 for descriptions of these views.

> **Note:** There are some additional restrictions and conditions that can affect whether a join view is inherently updatable. Specifics are listed in the description of the CREATE VIEW statement in the *Oracle Database SQL Language Reference.*
>
> If a view is not inherently updatable, it can be made updatable by creating an INSTEAD OF trigger on it. See *Oracle Database PL/SQL Language Reference* for information about triggers.
>
> 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 that reference views, see the *Oracle Database Performance Tuning Guide.*

Examples illustrating the rules for inherently updatable join views, and a discussion of key-preserved tables, are presented in following sections. The examples in these sections work only if you explicitly define the primary and foreign keys in the tables, or define unique indexes. The following statements create 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 in the preceding example, 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 keys 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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the UPDATE statement

DELETE Statements You can delete from a join view provided there is *one and only one* key-preserved table in the join. The key-preserved table can be repeated in the FROM clause.

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.

In the following view, a DELETE operation is permitted, because although there are two key-preserved tables, they are the same table. That is, the key-preserved table is repeated. In this case, the delete statement operates on the first table in the FROM list (e1, in this example):

```
CREATE VIEW emp_emp AS
     SELECT e1.ename, e2.empno, e2.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, 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;
```

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the DELETE statement

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 join view):

```
INSERT INTO emp_dept (empno, ename, loc)
   VALUES (9010, 'KURODA', 'BOSTON');
```

An INSERT cannot implicitly or explicitly refer to columns of a non-key-preserved table. If the join view is defined using the WITH CHECK OPTION clause, then you cannot perform an INSERT to it.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the INSERT statement

Updating Views That Involve Outer Joins

Views that involve outer joins are modifiable in some cases. For example:

```
CREATE VIEW emp_dept_oj1 AS
    SELECT empno, ename, e.deptno, dname, loc
    FROM emp e, dept d
    WHERE e.deptno = d.deptno (+);
```

The statement:

```
SELECT * FROM emp_dept_oj1;
```

Results in:

```
EMPNO   ENAME      DEPTNO  DNAME          LOC
------- ---------- ------- -------------- -------------
7369    SMITH      40      OPERATIONS     BOSTON
7499    ALLEN      30      SALES          CHICAGO
7566    JONES      20      RESEARCH       DALLAS
```

```
7654    MARTIN     30      SALES          CHICAGO
7698    BLAKE      30      SALES          CHICAGO
7782    CLARK      10      ACCOUNTING     NEW YORK
7788    SCOTT      20      RESEARCH       DALLAS
7839    KING       10      ACCOUNTING     NEW YORK
7844    TURNER     30      SALES          CHICAGO
7876    ADAMS      20      RESEARCH       DALLAS
7900    JAMES      30      SALES          CHICAGO
7902    FORD       20      RESEARCH       DALLAS
7934    MILLER     10      ACCOUNTING     NEW YORK
7521    WARD       30      SALES          CHICAGO
14 rows selected.
```

Columns in the base `emp` table of `emp_dept_oj1` are modifiable through the view, because `emp` is a key-preserved table in the join.

The following view also contains an outer join:

```
CREATE VIEW emp_dept_oj2 AS
SELECT e.empno, e.ename, e.deptno, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno (+) = d.deptno;
```

The following statement:

```
SELECT * FROM emp_dept_oj2;
```

Results in:

```
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
7499       ALLEN      30        SALES          CHICAGO
7698       BLAKE      30        SALES          CHICAGO
7654       MARTIN     30        SALES          CHICAGO
7900       JAMES      30        SALES          CHICAGO
7844       TURNER     30        SALES          CHICAGO
7521       WARD       30        SALES          CHICAGO
                                OPERATIONS     BOSTON
15 rows selected.
```

In this view, `emp` is no longer a key-preserved table, because the `empno` column in the result of the join can have nulls (the last row in the preceding `SELECT` statement). So, `UPDATE`, `DELETE`, and `INSERT` operations cannot be performed on this view.

In the case of views containing an outer join on other nested views, a table is key preserved if the view or views containing the table are merged into their outer views, all the way to the top. A view which is being outer-joined is currently merged only if it is "simple." For example:

```
SELECT col1, col2, ... FROM T;
```

The select list of the view has no expressions, and there is no `WHERE` clause.

Consider the following set of views:

```
CREATE VIEW emp_v AS
    SELECT empno, ename, deptno
        FROM emp;
CREATE VIEW emp_dept_oj1 AS
    SELECT e.*, Loc, d.dname
        FROM emp_v e, dept d
            WHERE e.deptno = d.deptno (+);
```

In these examples, emp_v is merged into emp_dept_oj1 because emp_v is a simple view, and so emp is a key-preserved table. But if emp_v is changed as follows:

```
CREATE VIEW emp_v_2 AS
    SELECT empno, ename, deptno
        FROM emp
            WHERE sal > 1000;
```

Then, because of the presence of the WHERE clause, emp_v_2 cannot be merged into emp_dept_oj1, and hence emp is no longer a key-preserved table.

If you are in doubt whether a view is modifiable, then you can select from the USER_UPDATABLE_COLUMNS view to see if it is. For example:

```
SELECT owner, table_name, column_name, updatable FROM USER_UPDATABLE_COLUMNS
     WHERE TABLE_NAME = 'EMP_DEPT_VIEW';
```

This returns output similar to the following:

```
OWNER       TABLE_NAME      COLUMN_NAM      UPD
----------  ----------      ----------      ---
SCOTT       EMP_DEPT_V      EMPNO           NO
SCOTT       EMP_DEPT_V      ENAME           NO
SCOTT       EMP_DEPT_V      DEPTNO          NO
SCOTT       EMP_DEPT_V      DNAME           NO
SCOTT       EMP_DEPT_V      LOC             NO
5 rows selected.
```

Using the UPDATABLE_ COLUMNS Views

The views described in the following table can assist you to identify inherently updatable join views.

View	Description
DBA_UPDATABLE_COLUMNS	Shows all columns in all tables and views that are modifiable.
ALL_UPDATABLE_COLUMNS	Shows all columns in all tables and views accessible to the user that are modifiable.
USER_UPDATABLE_ COLUMNS	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.
```

> **See Also:** *Oracle Database Reference* for complete descriptions of the updatable column views

Altering Views

You use the `ALTER VIEW` statement only to explicitly recompile a view that is invalid. To change the definition of a view, see "Replacing Views" on page 24-4.

The `ALTER VIEW` statement lets you 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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the `ALTER VIEW` statement

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;
```

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the `DROP VIEW` statement

Managing Sequences

This section describes aspects of managing sequences, and contains the following topics:

- About Sequences
- Creating Sequences
- Altering Sequences
- Using Sequences
- Dropping Sequences

About Sequences

Sequences are database objects from which multiple users can generate unique integers. The sequence generator generates sequential numbers, which can be used to generate unique primary keys automatically, and to coordinate keys across multiple rows or tables.

Without sequences, sequential values can only be produced programmatically. A new primary key value can be obtained by selecting the most recently produced value and incrementing it. This method requires a lock during the transaction and causes multiple users to wait for the next value of the primary key; this waiting is known as

serialization. If developers have such constructs in applications, then you should encourage the developers to replace them with access to sequences. Sequences eliminate serialization and improve the concurrency of an application.

> **See Also:** *Oracle Database Concepts* for an overview of sequences

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;
```

Notice that several parameters can be specified to control the function of sequences. You can use these parameters to indicate whether the sequence is ascending or descending, the starting point of the sequence, the minimum and maximum values, and the interval between sequence values. The `NOCYCLE` option indicates that the sequence cannot generate more values after reaching its maximum or minimum value.

The `CACHE` clause preallocates 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, the database reads another set of numbers into the cache.

The database 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. The database might also skip cached sequence numbers after an export and import. See *Oracle Database Utilities* for details.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the `CREATE SEQUENCE` statement syntax
> - *Oracle Real Application Clusters Administration and Deployment Guide* for information about using sequences in an Oracle Real Application Clusters environment

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 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;
```

See Also: *Oracle Database SQL Language Reference* for syntax and additional information about the `ALTER SEQUENCE` statement

Using Sequences

To use a sequence, your schema must contain the sequence or you must have been granted the `SELECT` object privilege for another user's sequence. Once a sequence is defined, it can be accessed and incremented by multiple users (who have `SELECT` object privilege for the sequence containing the sequence) with no waiting. The database does not wait for a transaction that has incremented a sequence to complete before that sequence can be incremented again.

The examples outlined in the following sections show how sequences can be used in master/detail table relationships. Assume an order entry system is partially comprised of two tables, `orders_tab` (master table) and `line_items_tab` (detail table), that hold information about customer orders. A sequence named `order_seq` is defined by the following statement:

```
CREATE SEQUENCE Order_seq
    START WITH 1
    INCREMENT BY 1
    NOMAXVALUE
    NOCYCLE
    CACHE 20;
```

Referencing a Sequence

A sequence is referenced in SQL statements with the `NEXTVAL` and `CURRVAL` pseudocolumns; each new sequence number is generated by a reference to the sequence pseudocolumn `NEXTVAL`, while the current sequence number can be repeatedly referenced using the pseudo-column `CURRVAL`.

`NEXTVAL` and `CURRVAL` are not reserved words or keywords and can be used as pseudocolumn names in SQL statements such as `SELECT`, `INSERT`, or `UPDATE`.

Generating Sequence Numbers with NEXTVAL To generate and use a sequence number, reference *seq_name*.`NEXTVAL`. For example, assume a customer places an order. The sequence number can be referenced in a values list. For example:

```
INSERT INTO Orders_tab (Orderno, Custno)
    VALUES (Order_seq.NEXTVAL, 1032);
```

Or, the sequence number can be referenced in the `SET` clause of an `UPDATE` statement. For example:

```
UPDATE Orders_tab
    SET Orderno = Order_seq.NEXTVAL
    WHERE Orderno = 10112;
```

The sequence number can also be referenced outermost `SELECT` of a query or subquery. For example:

```
SELECT Order_seq.NEXTVAL FROM dual;
```

As defined, the first reference to `order_seq.NEXTVAL` returns the value 1. Each subsequent statement that references `order_seq.NEXTVAL` generates the next sequence number (2, 3, 4,. . .). The pseudo-column `NEXTVAL` can be used to generate as many new sequence numbers as necessary. However, only a single sequence number can be generated for each row. In other words, if `NEXTVAL` is referenced more than once in a single statement, then the first reference generates the next number, and all subsequent references in the statement return the same number.

Once a sequence number is generated, the sequence number is available only to the session that generated the number. Independent of transactions committing or rolling back, other users referencing `order_seq.NEXTVAL` obtain unique values. If two users are accessing the same sequence concurrently, then the sequence numbers each user receives might have gaps because sequence numbers are also being generated by the other user.

Using Sequence Numbers with CURRVAL To use or refer to the current sequence value of your session, reference *seq_name*.`CURRVAL`. `CURRVAL` can only be used if *seq_name*.`NEXTVAL` has been referenced in the current user session (in the current or a previous transaction). `CURRVAL` can be referenced as many times as necessary, including multiple times within the same statement. The next sequence number is not generated until `NEXTVAL` is referenced. Continuing with the previous example, you would finish placing the customer's order by inserting the line items for the order:

```
INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 20321, 3);

INSERT INTO Line_items_tab (Orderno, Partno, Quantity)
    VALUES (Order_seq.CURRVAL, 29374, 1);
```

Assuming the `INSERT` statement given in the previous section generated a new sequence number of 347, both rows inserted by the statements in this section insert rows with order numbers of 347.

Uses and Restrictions of NEXTVAL and CURRVAL `CURRVAL` and `NEXTVAL` can be used in the following places:

- `VALUES` clause of `INSERT` statements
- The `SELECT` list of a `SELECT` statement
- The `SET` clause of an `UPDATE` statement

`CURRVAL` and `NEXTVAL` cannot be used in these places:

- A subquery
- A view query or materialized view query
- A `SELECT` statement with the `DISTINCT` operator
- A `SELECT` statement with a `GROUP BY` or `ORDER BY` clause
- A `SELECT` statement that is combined with another `SELECT` statement with the `UNION, INTERSECT`, or `MINUS` set operator
- The `WHERE` clause of a `SELECT` statement
- `DEFAULT` value of a column in a `CREATE TABLE` or `ALTER TABLE` statement
- The condition of a `CHECK` constraint

Caching Sequence Numbers

Sequence numbers can be kept in the sequence cache in the System Global Area (SGA). Sequence numbers can be accessed more quickly in the sequence cache than they can be read from disk.

The sequence cache consists of entries. Each entry can hold many sequence numbers for a single sequence.

Follow these guidelines for fast access to all sequence numbers:

- Be sure the sequence cache can hold all the sequences used concurrently by your applications.
- Increase the number of values for each sequence held in the sequence cache.

The Number of Entries in the Sequence Cache When an application accesses a sequence in the sequence cache, the sequence numbers are read quickly. However, if an application accesses a sequence that is not in the cache, then the sequence must be read from disk to the cache before the sequence numbers are used.

If your applications use many sequences concurrently, then your sequence cache might not be large enough to hold all the sequences. In this case, access to sequence numbers might often require disk reads. For fast access to all sequences, be sure your cache has enough entries to hold all the sequences used concurrently by your applications.

The Number of Values in Each Sequence Cache Entry When a sequence is read into the sequence cache, sequence values are generated and stored in a cache entry. These values can then be accessed quickly. The number of sequence values stored in the cache is determined by the `CACHE` parameter in the `CREATE SEQUENCE` statement. The default value for this parameter is 20.

This `CREATE SEQUENCE` statement creates the `seq2` sequence so that 50 values of the sequence are stored in the `SEQUENCE` cache:

```
CREATE SEQUENCE seq2
    CACHE 50;
```

The first 50 values of `seq2` can then be read from the cache. When the 51st value is accessed, the next 50 values will be read from disk.

Choosing a high value for `CACHE` lets you access more successive sequence numbers with fewer reads from disk to the sequence cache. However, if there is an instance failure, then all sequence values in the cache are lost. Cached sequence numbers also could be skipped after an export and import if transactions continue to access the sequence numbers while the export is running.

If you use the `NOCACHE` option in the `CREATE SEQUENCE` statement, then the values of the sequence are not stored in the sequence cache. In this case, every access to the sequence requires a disk read. Such disk reads slow access to the sequence. This `CREATE SEQUENCE` statement creates the `SEQ3` sequence so that its values are never stored in the cache:

```
CREATE SEQUENCE seq3
    NOCACHE;
```

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.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the DROP SEQUENCE statement

Managing Synonyms

This section describes aspects of managing synonyms, and contains the following topics:

- About Synonyms
- Creating Synonyms
- Using Synonyms in DML Statements
- Dropping Synonyms

About 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 must 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 to grantees for the underlying object.

Synonyms themselves are not securable. When you grant object privileges on a synonym, you are really granting privileges on the underlying object, and the synonym is acting only as an alias for the object in the GRANT statement.

> **See Also:** *Oracle Database Concepts* for a more complete description of synonyms

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 for the CREATE SYNONYM statement to succeed. 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
```

When you create a synonym for a remote procedure or function, you must qualify the remote object with its schema name. Alternatively, you can create a local public synonym on the database where the remote object resides, in which case the database link must be included in all subsequent calls to the procedure or function.

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the `CREATE SYNONYM` statement

Using Synonyms in DML Statements

You can successfully use any private synonym contained in your schema or any public synonym, assuming that you have the necessary privileges to access the underlying object, either explicitly, from an enabled role, or from `PUBLIC`. You can also reference any private synonym contained in another schema if you have been granted the necessary object privileges for the underlying object.

You can reference another user's synonym using only the object privileges that you have been granted. For example, if you have only the `SELECT` privilege on the `jward.emp` table, and the synonym `jward.employee` is created for `jward.emp`, you can query the `jward.employee` synonym, but you cannot insert rows using the `jward.employee` synonym.

A synonym can be referenced in a DML statement the same way that the underlying object of the synonym can be referenced. For example, if a synonym named `employee` refers to a table or view, then the following statement is valid:

```
INSERT INTO employee (empno, ename, job)
    VALUES (emp_sequence.NEXTVAL, 'SMITH', 'CLERK');
```

If the synonym named `fire_emp` refers to a standalone procedure or package procedure, then you could execute it with the command

```
EXECUTE Fire_emp(7344);
```

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".

> **See Also:** *Oracle Database SQL Language Reference* for syntax and additional information about the `DROP SYNONYM` statement

Views, Synonyms, and Sequences Data Dictionary Views

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: *Oracle Database Reference* for complete descriptions of these views

25

Repairing Corrupted Data

In this chapter:

- 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, then 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 Database 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, then another option is to rebuild the table by selecting all data except for the corrupt rows.

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. You can 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. Depending on the nature of the repair, you might lose data, and logical inconsistencies can be introduced. You must determine whether the repair approach provided by this package is the appropriate tool for each specific corruption problem.

About the DBMS_REPAIR Package

This section describes the procedures contained in the `DBMS_REPAIR` package and notes some limitations and restrictions on their use.

> **See Also:** *Oracle Database 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
ADMIN_TABLES	Provides administrative functions (create, drop, purge) for repair or orphan key tables. **Note:** These tables are always created in the SYS schema.
CHECK_OBJECT	Detects and reports corruptions in a table or index
DUMP_ORPHAN_KEYS	Reports on index entries that point to rows in corrupt data blocks
FIX_CORRUPT_BLOCKS	Marks blocks as software corrupt that have been previously identified as corrupt by the CHECK_OBJECT procedure
REBUILD_FREELISTS	Rebuilds the free lists of the object
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-01578 when encountering blocks marked corrupt.

These procedures are further described, with examples of their use, in "DBMS_REPAIR Examples" on page 25-5.

Limitations and Restrictions

DBMS_REPAIR procedures have the following limitations:

- Tables with LOB data types, 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 no more than 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

Task 1: Detect and Report Corruptions

The first task is the detection and reporting of corruptions. Reporting not only indicates what is wrong with a block, but also identifies the associated repair directive. There are several ways to detect corruptions. Table 25–1 describes the different detection methodologies.

Table 25–1 Comparison of Corruption Detection Methods

Detection Method	Description
DBMS_REPAIR PL/SQL package	Performs block checking for a specified table, partition, or index. It populates a repair table with results.
DB_VERIFY utility	Performs block checking on an offline database
ANALYZE TABLE SQL statement	Used with the VALIDATE STRUCTURE option, the ANALYZE TABLE statement verifies the integrity of the structure of an index, table, or cluster; checks or verifies that tables and indexes are synchronized.
DB_BLOCK_CHECKING initialization parameter	When DB_BLOCK_CHECKING=TRUE, corrupt blocks are identified before they 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 reported problems.

DB_VERIFY: Performing an Offline Database Check

Use DB_VERIFY as an offline diagnostic utility when you encounter data corruption.

> **See Also:** *Oracle Database Utilities* for more information about DB_VERIFY

ANALYZE: Reporting Corruption

The ANALYZE TABLE...VALIDATE STRUCTURE statement validates the structure of the analyzed object. If the database encounters corruption in the structure of the object, then an error message is returned. In this case, drop and re-create the object.

You can use the CASCADE clause of the ANALYZE TABLE statement to check the structure of the table and all of its indexes in one operation. Because this operation can consume significant resources, there is a FAST option that performs a lightweight check. See "Validating Tables, Indexes, Clusters, and Materialized Views" on page 18-3 for details.

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

DB_BLOCK_CHECKING Initialization Parameter

You can enable database block checking by setting the DB_BLOCK_CHECKING initialization parameter to TRUE. This checks data and index blocks for internal consistency 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:** *Oracle 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. Begin by answering the following questions:

- 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.

- What other options are available for addressing block corruptions? Consider the following:
  - If the data is available from another source, then drop, re-create, and repopulate 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.

- 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?

  You might not have access to rows in blocks marked corrupt. However, a block can be marked corrupt even if there are rows that you can validly access.

  It is also possible that referential integrity constraints are broken when blocks are marked corrupt. If this occurs, then disable and reenable the constraint; any inconsistencies are reported. After fixing all problems, you should be able to reenable the constraint.

  Logical corruption can occur when there are triggers defined on the table. For example, if rows are reinserted, should insert triggers be fired or not? You can address these issues only if you understand triggers and their use in your installation.

  If indexes and tables are not synchronized, then execute 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 synchronize the table with its indexes.

- 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.

Task 3: Make Objects Usable

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 can make a corrupt object usable by establishing an environment that skips corruptions that remain outside the scope of DBMS_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 that are marked as corrupt. When the SKIP_FLAG parameter in the procedure 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 not synchronized, then a SET TRANSACTION READ ONLY transaction can be inconsistent in situations where one query probes only the index, and 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 not to skip corruptions in a SET TRANSACTION READ ONLY transaction.

A similar issue occurs when selecting rows that are chained. A query of the same row may or may not access the corruption, producing different results.

Task 4: Repair Corruptions and Rebuild Lost Data

After making an object usable, 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.

Fix Segment Bitmaps Using the SEGMENT_FIX_STATUS Procedure

Use this procedure if free space in segments is being managed by using bitmaps (SEGMENT SPACE MANAGEMENT AUTO).

This procedure recalculates the state of a bitmap entry based on the current contents of the corresponding block. Alternatively, 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

This section includes the following topics:

- Examples: Building a Repair Table or Orphan Key Table
- Example: Detecting Corruption
- Example: Fixing Corrupt Blocks

- Example: Finding Index Entries Pointing to Corrupt Data Blocks
- Example: Skipping Corrupt Blocks

Examples: Building a Repair Table or Orphan Key Table

The ADMIN_TABLE procedure is used to create, purge, or drop a repair table or an orphan key table.

A repair table provides information about the corruptions that 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.

Example: 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 and is prefixed by DBA_ (for example, DBA_REPAIR_TABLE or DBA_ORPHAN_KEY_TABLE).

The following query describes the repair table that was created for the users tablespace.

```
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
```

Example: 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:

```
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
```

Example: Detecting Corruption

The `CHECK_OBJECT` procedure checks the specified object, 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 want 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.

The following 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   ktbbhitc=1
mark block software corrupt
```

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.

Example: Fixing Corrupt Blocks

Use the `FIX_CORRUPT_BLOCKS` procedure to fix the corrupt blocks in specified objects based on information in the repair table that was generated by the `CHECK_OBJECT` procedure. Before changing a block, the block is checked to ensure that 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 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
```

Example: Finding Index Entries Pointing to Corrupt Data Blocks

The DUMP_ORPHAN_KEYS procedure reports on index entries that point to rows in corrupt data blocks. For each index entry, 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 output indicates that 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.

Example: Skipping 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, skipping 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 Resource Management and Task Scheduling

Part IV discusses automated database maintenance tasks, database resource management, and task scheduling. It contains the following chapters:

- Chapter 26, "Managing Automated Database Maintenance Tasks"
- Chapter 27, "Managing Resources with Oracle Database Resource Manager"
- Chapter 28, "Oracle Scheduler Concepts"
- Chapter 29, "Scheduling Jobs with Oracle Scheduler"
- Chapter 30, "Administering Oracle Scheduler"

26

Managing Automated Database Maintenance Tasks

Oracle Database has automated several common maintenance tasks typically performed by database administrators. These automated maintenance tasks are performed when the system load is expected to be light. You can enable and disable individual maintenance tasks, and can configure when these tasks run and what resource allocations they are allotted.

In this chapter:

- About Automated Maintenance Tasks
- About Maintenance Windows
- Configuring Automated Maintenance Tasks
- Configuring Maintenance Windows
- Configuring Resource Allocations for Automated Maintenance Tasks
- Automated Maintenance Tasks Reference

Note: This chapter explains how to administer automated maintenance tasks using PL/SQL packages. An easier way is to use the graphical interface of Enterprise Manager.

To manage automatic maintenance tasks with Enterprise Manager:

1. Access the Database Home Page.

   See *Oracle Database 2 Day DBA* or the Oracle Enterprise Manager Grid Control online help for instructions.

2. On the Database Home Page, click **Server** to display the Server page.
3. Under the Oracle Scheduler section, click **Automated Maintenance Tasks** to configure maintenance tasks, or **Windows** to configure maintenance windows.

About Automated Maintenance Tasks

Automated maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. An example is a task that gathers statistics on schema objects for the query optimizer. Automated maintenance tasks run in *maintenance windows*, which are predefined time intervals that are intended to occur during a period of low system load. You can customize maintenance windows based on the resource usage patterns of your database, or

disable certain default windows from running. You can also create your own maintenance windows.

Oracle Database has three predefined automated maintenance tasks:

- **Automatic Optimizer Statistics Collection—**Collects optimizer statistics for all schema objects in the database for which there are no statistics or only stale statistics. The statistics gathered by this task are used by the SQL query optimizer to improve the performance of SQL execution.

  See Also: *Oracle Database Performance Tuning Guide* for more information on automatic statistics collection

- **Automatic Segment Advisor**—Identifies segments that have space available for reclamation, and makes recommendations on how to defragment those segments.

  You can also run the Segment Advisor manually to obtain more up-to-the-minute recommendations or to obtain recommendations on segments that the Automatic Segment Advisor did not examine for possible space reclamation.

  See Also: "Using the Segment Advisor" on page 19-12 for more information.

- **Automatic SQL Tuning Advisor—**Examines the performance of high-load SQL statements, and makes recommendations on how to tune those statements. You can configure this advisor to automatically implement SQL profile recommendations.

  See Also: *Oracle Database Performance Tuning Guide* for more information on SQL Tuning Advisor

By default, all three automated maintenance tasks are configured to run in all maintenance windows.

About Maintenance Windows

A **maintenance window** is a contiguous time interval during which automated maintenance tasks are run. Maintenance windows are Oracle Scheduler windows that belong to the window group named `MAINTENANCE_WINDOW_GROUP`. A Scheduler window can be a simple repeating interval (such as "between midnight and 6 a.m., every Saturday"), or a more complex interval (such as "between midnight and 6 a.m., on the last workday of every month, excluding company holidays").

When a maintenance window opens, Oracle Database creates an Oracle Scheduler job for each maintenance task that is scheduled to run in that window. Each job is assigned a job name that is generated at run time. All automated maintenance task job names begin with `ORA$AT`. For example, the job for the Automatic Segment Advisor might be called `ORA$AT_SA_SPC_SY_26`. When an automated maintenance task job finishes, it is deleted from the Oracle Scheduler job system. However, the job can still be found in the Scheduler job history.

Note: To view job history, you must log in as the `SYS` user.

In the case of a very long maintenance window, all automated maintenance tasks except Automatic SQL Tuning Advisor are restarted every four hours. This feature ensures that maintenance tasks are run regularly, regardless of window size.

The framework of automated maintenance tasks relies on maintenance windows being defined in the database. Table 26–1 on page 26-7 lists the maintenance windows that are automatically defined with each new Oracle Database installation.

See Also:

- "About Jobs and Supporting Scheduler Objects" on page 28-3 for more information on windows and groups.

Configuring Automated Maintenance Tasks

To enable or disable specific maintenance tasks in any subset of maintenance windows, you can use the `DBMS_AUTO_TASK_ADMIN` PL/SQL package.

This section contains the following topics:

- Enabling and Disabling Maintenance Tasks for all Maintenance Windows
- Enabling and Disabling Maintenance Tasks for Specific Maintenance Windows

Enabling and Disabling Maintenance Tasks for all Maintenance Windows

You can disable a particular automated maintenance tasks for all maintenance windows with a single operation. You do so by calling the `DISABLE` procedure of the `DBMS_AUTO_TASK_ADMIN` PL/SQL package without supplying the `window_name` argument. For example, you can completely disable the Automatic SQL Tuning Advisor task as follows:

```
BEGIN
  dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
/
```

To enable this maintenance task again, use the `ENABLE` procedure, as follows:

```
BEGIN
  dbms_auto_task_admin.enable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => NULL);
END;
/
```

The task names to use for the `client_name` argument are listed in the `DBA_AUTOTASK_CLIENT` database dictionary view.

To enable or disable all automated maintenance tasks for all windows, call the `ENABLE` or `DISABLE` procedure with no arguments.

```
EXECUTE DBMS_AUTO_TASK_ADMIN.DISABLE;
```

See Also:

- "Automated Maintenance Tasks Database Dictionary Views" on page 26-7
- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_AUTO_TASK_ADMIN` PL/SQL package.

Enabling and Disabling Maintenance Tasks for Specific Maintenance Windows

By default, all maintenance tasks run in all predefined maintenance windows. You can disable a maintenance task for a specific window. The following example disables the Automatic SQL Tuning Advisor from running in the window MONDAY_WINDOW:

```
BEGIN
  dbms_auto_task_admin.disable(
    client_name => 'sql tuning advisor',
    operation   => NULL,
    window_name => 'MONDAY_WINDOW');
END;
/
```

Configuring Maintenance Windows

You may want to adjust the predefined maintenance windows to a time suitable to your database environment or create a new maintenance window. You can customize maintenance windows using the DBMS_SCHEDULER PL/SQL package.

This section contains the following topics:

- Modifying a Maintenance Window
- Creating a New Maintenance Window
- Removing a Maintenance Window

Modifying a Maintenance Window

The DBMS_SCHEDULER PL/SQL package includes a SET_ATTRIBUTE procedure for modifying the attributes of a window. For example, the following script changes the duration of the maintenance window SATURDAY_WINDOW to 4 hours:

```
BEGIN
  dbms_scheduler.disable(
    name  => 'SATURDAY_WINDOW');
  dbms_scheduler.set_attribute(
    name      => 'SATURDAY_WINDOW',
    attribute => 'DURATION',
    value     => numtodsinterval(4, 'hour'));
  dbms_scheduler.enable(
    name => 'SATURDAY_WINDOW');
END;
/
```

Note that you must use the DBMS_SCHEDULER.DISABLE subprogram to disable the window before making changes to it, and then re-enable the window with DBMS_SCHEDULER.ENABLE when you are finished. If you change a window when it is currently open, the change does not take effect until the next time the window opens.

> **See Also:** "Managing Job Scheduling and Job Priorities with Windows" on page 29-55 for more information about modifying windows.

Creating a New Maintenance Window

To create a new maintenance window, you must create an Oracle Scheduler window object and then add it to the window group MAINTENANCE_WINDOW_GROUP. You use the DBMS_SCHEDULER.CREATE_WINDOW package procedure to create the window, and the

DBMS_SCHEDULER.ADD_GROUP_MEMBER procedure to add the new window to the window group.

The following example creates a maintenance window named EARLY_MORNING_WINDOW. This window runs for one hour daily between 5 a.m. and 6 a.m.

```
BEGIN
  dbms_scheduler.create_window(
    window_name     => 'EARLY_MORNING_WINDOW',
    duration        =>  numtodsinterval(1, 'hour'),
    resource_plan   => 'DEFAULT_MAINTENANCE_PLAN',
    repeat_interval => 'FREQ=DAILY;BYHOUR=5;BYMINUTE=0;BYSECOND=0');
  dbms_scheduler.add_group_member(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    member      => 'EARLY_MORNING_WINDOW');
END;
/
```

See Also:

- "Creating Windows" on page 29-57
- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_SCHEDULER package

Removing a Maintenance Window

To remove an existing maintenance window, remove it from the MAINTENANCE_WINDOW_GROUP window group. The window continues to exist but no longer runs automated maintenance tasks. Any other Oracle Scheduler jobs assigned to this window continue to run as usual.

The following example removes EARLY_MORNING_WINDOW from the window group:

```
BEGIN
  DBMS_SCHEDULER.REMOVE_GROUP_MEMBER(
    group_name  => 'MAINTENANCE_WINDOW_GROUP',
    member      => 'EARLY_MORNING_WINDOW');
END;
/
```

See Also:

- "Removing a Member from a Window Group" on page 29-62
- "Dropping Windows" on page 29-59
- *Oracle Database PL/SQL Packages and Types Reference* for information on the DBMS_SCHEDULER package

Configuring Resource Allocations for Automated Maintenance Tasks

This section contains the following topics on resource allocation for maintenance windows:

- About Resource Allocations for Automated Maintenance Tasks
- Changing Resource Allocations for Automated Maintenance Tasks

See Also: Chapter 27, "Managing Resources with Oracle Database Resource Manager"

About Resource Allocations for Automated Maintenance Tasks

By default, all predefined maintenance windows use the resource plan DEFAULT_MAINTENANCE_PLAN. Automated maintenance tasks run under its subplan ORA$AUTOTASK_SUB_PLAN. This subplan divides its portion of total resource allocation equally among the maintenance tasks.

DEFAULT_MAINTENANCE_PLAN defines the following resource allocations:

Consumer Group/subplan	Level 1	Level 2	Maximum Utilization Limit
ORA$AUTOTASK_SUB_PLAN	-	25%	90
ORA$DIAGNOSTICS	-	5%	90
OTHER_GROUPS	-	70%	
SYS_GROUP	75%	-	

In this plan, any sessions in the SYS_GROUP consumer group get priority. (Sessions in this group are sessions created by user accounts SYS and SYSTEM.) Any resource allocation that is unused by sessions in SYS_GROUP is then shared by sessions belonging to the other consumer groups and subplans in the plan. Of that allocation, 25% goes to maintenance tasks, 5% goes to background processes performing diagnostic operations, and 70% goes to user sessions. The maximum utilization limit for ORA$AUTOTASK_SUB_PLAN and ORA$DIAGNOSTICS is 90. Therefore, even if the CPU is idle, this group/plan cannot be allocated more than 90% of the CPU resources.

To reduce or increase resource allocation to the automated maintenance tasks, you make adjustments to DEFAULT_MAINTENANCE_PLAN. See "Changing Resource Allocations for Automated Maintenance Tasks" on page 26-6 for more information.

Note that as with any resource plan, the portion of an allocation that is not used by a consumer group or subplan is available for other consumer groups or subplans. Note also that the Database Resource Manager does not begin to limit resource allocations according to resource plans until 100% of CPU is being used.

Note: Although DEFAULT_MAINTENANCE_PLAN is the default, you can assign any resource plan to any maintenance window. If you do change a maintenance window resource plan, ensure that you include the subplan ORA$AUTOTASK_SUB_PLAN and the consumer group ORA$DIAGNOSTICS in the new plan.

See Also: Chapter 27, "Managing Resources with Oracle Database Resource Manager" for more information on resource plans.

Changing Resource Allocations for Automated Maintenance Tasks

To change the resource allocation for automated maintenance tasks within a maintenance window, you must change the percentage of resources allocated to the subplan ORA$AUTOTASK_SUB_PLAN in the resource plan for that window. (By default, the resource plan for each predefined maintenance window is DEFAULT_MAINTENANCE_PLAN.) You must also adjust the resource allocation for one or more other subplans or consumer groups in the window's resource plan such that the resource allocation at the top level of the plan adds up to 100%. For information on changing resource allocations, see Chapter 27, "Managing Resources with Oracle Database Resource Manager".

Automated Maintenance Tasks Reference

This section contains the following reference topics for automated maintenance tasks:

- Predefined Maintenance Windows
- Automated Maintenance Tasks Database Dictionary Views

Predefined Maintenance Windows

By default there are seven predefined maintenance windows, each one representing a day of the week. The weekend maintenance windows, SATURDAY_WINDOW and SUNDAY_WINDOW, are longer in duration than the weekday maintenance windows. The window group MAINTENANCE_WINDOW_GROUP consists of these seven windows. The list of predefined maintenance windows is given in Table 26–1.

Table 26–1 Predefined Maintenance Windows

Window Name	Description
MONDAY_WINDOW	Starts at 10 p.m. on Monday and ends at 2 a.m.
TUESDAY_WINDOW	Starts at 10 p.m. on Tuesday and ends at 2 a.m.
WEDNESDAY_WINDOW	Starts at 10 p.m. on Wednesday and ends at 2 a.m.
THURSDAY_WINDOW	Starts at 10 p.m. on Thursday and ends at 2 a.m.
FRIDAY_WINDOW	Starts at 10 p.m. on Friday and ends at 2 a.m.
SATURDAY_WINDOW	Starts at 6 a.m. on Saturday and is 20 hours long.
SUNDAY_WINDOW	Starts at 6 a.m. on Sunday and is 20 hours long.

Automated Maintenance Tasks Database Dictionary Views

Table 26–2 displays information about database dictionary views for automated maintenance tasks:

Table 26–2 Automated Maintenance Tasks Database Dictionary Views

View Name	Description
DBA_AUTOTASK_CLIENT_JOB	Contains information about currently running Scheduler jobs created for automated maintenance tasks. It provides information about some objects targeted by those jobs, as well as some additional statistics from previous instantiations of the same task. Some of this additional data is taken from generic Scheduler views.
DBA_AUTOTASK_CLIENT	Provides statistical data for each automated maintenance task over 7-day and 30-day periods.
DBA_AUTOTASK_JOB_HISTORY	Lists the history of automated maintenance task job runs. Jobs are added to this view after they finish executing.
DBA_AUTOTASK_WINDOW_CLIENTS	Lists the windows that belong to MAINTENANCE_WINDOW_GROUP, along with the Enabled or Disabled status for the window for each maintenance task. Primarily used by Enterprise Manager.
DBA_AUTOTASK_CLIENT_HISTORY	Provides per-window history of job execution counts for each automated maintenance task. This information is viewable in the Job History page of Enterprise Manager.

See Also: "Resource Manager Data Dictionary Views" on page 27-60 for column descriptions for views.

27

Managing Resources with Oracle Database Resource Manager

In this chapter:

- About Oracle Database Resource Manager
- Assigning Sessions to Resource Consumer Groups
- The Types of Resources Managed by the Resource Manager
- Creating a Simple Resource Plan
- Creating a Complex Resource Plan
- Enabling Oracle Database Resource Manager and Switching Plans
- Putting It All Together: Oracle Database Resource Manager Examples
- Managing Multiple Database Instances on a Single Server
- Maintaining Consumer Groups, Plans, and Directives
- Viewing Database Resource Manager Configuration and Status
- Monitoring Oracle Database Resource Manager
- Interacting with Operating-System Resource Control
- Oracle Database Resource Manager Reference

Note: This chapter discusses using PL/SQL package procedures to administer the Resource Manager. An easier way to administer the Resource Manager is with the graphical user interface of Enterprise Manager. For instructions about administering Resource Manager with Enterprise Manager, see *Oracle Database 2 Day DBA*.

About Oracle Database Resource Manager

Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources.

The following sections provide an overview of the Resource Manager:

- What Solutions Does the Resource Manager Provide for Workload Management?
- Elements of the Resource Manager
- The Types of Resources Managed by the Resource Manager

- About Resource Manager Administration Privileges

What Solutions Does the Resource Manager Provide for Workload Management?

When database resource allocation decisions are left to the operating system, you may encounter the following problems with workload management:

- Excessive overhead

  Excessive overhead results from operating system context switching between Oracle Database server processes when the number of server processes is high.

- Inefficient scheduling

  The operating system deschedules database servers while they hold latches, which is inefficient.

- Inappropriate allocation of resources

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another.

- Inability to manage database-specific resources, such as parallel execution servers and active sessions

The Resource Manager helps to overcome these problems by allowing the database more control over how hardware resources are allocated. In an environment with multiple concurrent user sessions that run jobs with differing priorities, all sessions should not be treated equally. The Resource Manager enables you to classify sessions into groups based on session attributes, and to then allocate resources to those groups in a way that optimizes hardware utilization for your application environment.

With the Resource Manager, you can:

- Guarantee certain sessions a minimum amount of CPU regardless of the load on the system and the number of users.
- Distribute available CPU by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage can be given to ROLAP (relational online analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users.
- Manage the order of parallel statements in the parallel statement queue. Parallel statements from a critical application can be enqueued ahead of parallel statements from a low priority group of users.
- Limit the number of parallel servers that a group of users can use. This ensures that all the available parallel servers are not allocated to only one group of users.
- Create an active session pool. An **active session 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 terminate. The active session pool limits the total number of sessions actively competing for resources, thereby enabling active sessions to make faster progress.
- Manage runaway sessions or calls in the following ways:
  - By placing an absolute limit on the percentage of CPU that a group can consume

  - By detecting when a session or call consumes more than a specified amount of CPU or I/O, and then automatically either terminating the session or call, or switching it to a consumer group that is allocated a small amount of CPU, which would in effect mitigate the impact of the runaway session or call
- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit.
- Limit the amount of time that a session can be idle. This can be further defined to mean only sessions that are blocking other sessions.
- Allow a database to use different resource plans, based on changing workload requirements. You can dynamically change the resource plan, for example, from a daytime resource plan to a nighttime resource plan, without having to shut down and restart the instance. You can also schedule a resource plan change with Oracle Scheduler. See Chapter 28, "Oracle Scheduler Concepts" for more information.

Elements of the Resource Manager

The elements of the Resource Manager are described in the following table.

Element	Description
Resource consumer group	A group of sessions that are grouped together based on resource requirements. The Resource Manager allocates resources to resource consumer groups, not to individual sessions.
Resource plan	A container for directives that specify how resources are allocated to resource consumer groups. You specify how the database allocates resources by activating a specific resource plan.
Resource plan directive	Associates a resource consumer group with a particular plan and specifies how resources are to be allocated to that resource consumer group.

You use the `DBMS_RESOURCE_MANAGER` PL/SQL package to create and maintain these elements. The elements are stored in tables in the data dictionary. You can view information about them with data dictionary views.

> **See Also:** "Resource Manager Data Dictionary Views" on page 27-60

About Resource Consumer Groups

A resource consumer group (consumer group) is a collection of user sessions that are grouped together based on their processing needs. When a session is created, it is automatically mapped to a consumer group based on mapping rules that you set up. As a database administrator (DBA), you can manually switch a session to a different consumer group. Similarly, an application can run a PL/SQL package procedure that switches its session to a particular consumer group.

Because the Resource Manager allocates resources (such as CPU) only to consumer groups, when a session becomes a member of a consumer group, its resource allocation is determined by the allocation for the consumer group.

There are three special consumer groups that are always present in the data dictionary. They cannot be modified or deleted. They are:

- `SYS_GROUP`

This is the initial consumer group for all sessions created by user accounts SYS or SYSTEM. This initial consumer group can be overridden by session-to–consumer group mapping rules.

- OTHER_GROUPS

  This consumer group contains all sessions that have not been assigned to a consumer group. Every resource plan must contain a directive to OTHER_GROUPS.

  See Also:

  - Table 27–5, " Predefined Resource Consumer Groups" on page 27-59
  - "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8

About Resource Plan Directives

The Resource Manager allocates resources to consumer groups according to the set of resource plan directives (directives) that belong to the currently active resource plan. There is a parent-child relationship between a resource plan and its resource plan directives. Each directive references one consumer group, and no two directives for the currently active plan can reference the same consumer group.

A directive has several ways in which it can limit resource allocation for a consumer group. For example, it can control how much CPU the consumer group gets as a percentage of total CPU, and it can limit the total number of sessions that can be active in the consumer group. See "The Types of Resources Managed by the Resource Manager" on page 27-17 for more information.

About Resource Plans

In addition to the resource plans that are predefined for each Oracle database, you can create any number of resource plans. However, only one resource plan is active at a time. When a resource plan is active, each of its child resource plan directives controls resource allocation for a different consumer group. Each plan must include a directive that allocates resources to the consumer group named OTHER_GROUPS. OTHER_GROUPS applies to all sessions that belong to a consumer group that is not part of the currently active plan.

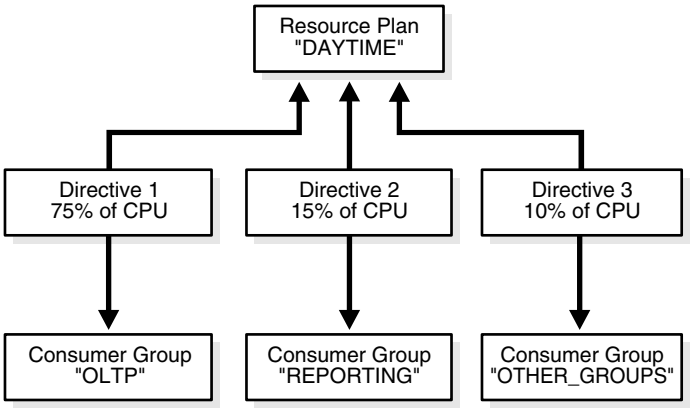
> **Note:** Although the term "resource plan" (or just "plan") denotes one element of the Resource Manager, in this chapter it is also used to refer to a complete *resource plan schema*, which includes the resource plan element itself, its resource plan directives, and the consumer groups that the directives reference. For example, when this chapter refers to the DAYTIME resource plan, it could mean either the resource plan element named DAYTIME, or the particular resource allocation schema that the DAYTIME resource plan and its directives define. Thus, for brevity, it is acceptable to say, "the DAYTIME plan favors interactive applications over batch applications."

Example: A Simple Resource Plan

Figure 27–1 shows a simple resource plan for an organization that runs online transaction processing (OLTP) applications and reporting applications simultaneously during the daytime. The currently active plan, DAYTIME, allocates CPU resources

among three resource consumer groups. Specifically, OLTP is allotted 75% of the CPU time, REPORTS is allotted 15%, and OTHER_GROUPS receives the remaining 10%.

Figure 27–1 A Simple Resource Plan



Oracle Database 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 27-24.

Note: The currently active resource plan does not enforce allocations until CPU usage is at 100%. If the CPU usage is below 100%, the database is not CPU-bound and hence there is no need to enforce allocations to ensure that all sessions get their designated resource allocation.

In addition, when allocations are enforced, unused allocation by any consumer group can be used by other consumer groups. In the previous example, if the OLTP group does not use all of its allocation, the Resource Manager permits the REPORTS group or OTHER_GROUPS group to use the unused allocation.

About Subplans

Instead of referencing a consumer group, a resource plan directive (directive) can reference another resource plan. In this case, the plan is referred to as a subplan. The subplan itself has directives that allocate resources to consumer groups and other subplans. The resource allocation scheme then works like this: The *top* resource plan (the currently active plan) divides resources among consumer groups and subplans. Each subplan allocates its portion of the total resource allocation among its consumer groups and subplans. You can create hierarchical plans with any number of subplans.

You create a resource subplan in the same way that you create a resource plan. To create a plan that is to be used only as a subplan, you use the SUB_PLAN argument in the package procedure DBMS_RESOURCE_MANAGER.CREATE_PLAN.

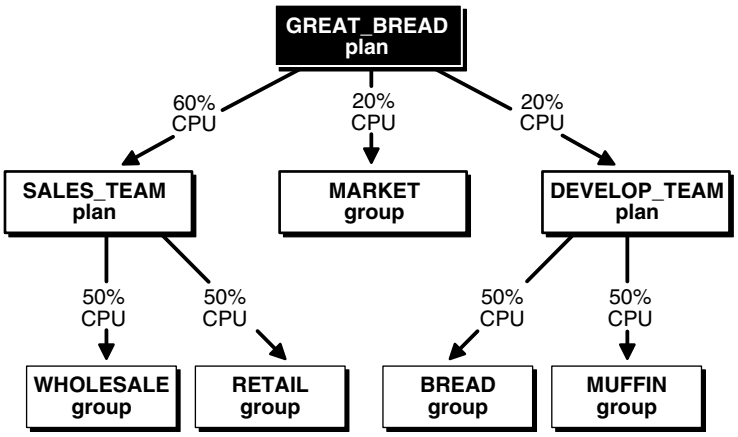
In any top level plan, you can reference a subplan only once. A subplan is not required to have a directive to OTHER_GROUPS and cannot be set as a resource plan.

Example: A Resource Plan with Subplans

In this example, the Great Bread Company allocates the CPU resource as shown in Figure 27–2. The figure illustrates a top plan (GREAT_BREAD) and all of its descendents. For simplicity, the requirement to include the OTHER_GROUPS consumer group is

ignored, and resource plan directives are not shown, even though they are part of the plan. Rather, the CPU percentages that the directives allocate are shown along the connecting lines between plans, subplans, and consumer groups.

Figure 27–2 A Resource Plan With Subplans



The GREAT_BREAD plan allocates resources as follows:

- 20% of CPU resources to the consumer group MARKET
- 60% of CPU resources to subplan SALES_TEAM, which in turn divides its share equally between the WHOLESALE and RETAIL consumer groups
- 20% of CPU resources to subplan DEVELOP_TEAM, which in turn divides its resources equally between the BREAD and MUFFIN consumer groups

It is possible for a subplan or consumer group to have multiple parents. An example would be if the MARKET group were included in the SALES_TEAM subplan. However, a plan cannot contain any loops. For example, the SALES_TEAM subplan cannot have a directive that references the GREAT_BREAD plan.

> **See Also:** "Putting It All Together: Oracle Database Resource Manager Examples" on page 27-37 for an example of a more complex resource plan.

About Resource Manager Administration Privileges

You must have the system privilege ADMINISTER_RESOURCE_MANAGER to administer the Resource Manager. This privilege (with the ADMIN option) is granted to database administrators through the DBA role.

Being an administrator for the Resource Manager enables you to execute all of the procedures in the DBMS_RESOURCE_MANAGER PL/SQL package.

You may, as an administrator with the ADMIN option, choose to grant the administrative privilege to other users or roles. To do so, use the DBMS_RESOURCE_MANAGER_PRIVS PL/SQL package. The relevant package procedures are listed in the following table.

Procedure	Description
GRANT_SYSTEM_PRIVILEGE	Grants the ADMINISTER_RESOURCE_MANAGER system privilege to a user or role.

Procedure	Description
REVOKE_SYSTEM_PRIVILEGE	Revokes the ADMINISTER_RESOURCE_MANAGER system privilege from a user or role.

The following PL/SQL block grants the administrative privilege to user HR, but does not grant HR the ADMIN option. Therefore, HR can execute all of the procedures in the DBMS_RESOURCE_MANAGER package, but HR cannot use the GRANT_SYSTEM_PRIVILEGE procedure to grant the administrative privilege to others.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE(
   GRANTEE_NAME   => 'HR',
   PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER',
   ADMIN_OPTION   => FALSE);
END;
/
```

You can revoke this privilege using the REVOKE_SYSTEM_PRVILEGE procedure.

Note: The ADMINISTER_RESOURCE_MANAGER system privilege can only be granted or revoked using the DBMS_RESOURCE_MANAGER_PRIVS package. It cannot be granted or revoked through the SQL GRANT or REVOKE statements.

See Also: *Oracle Database PL/SQL Packages and Types Reference.* contains detailed information about the Resource Manager packages:

- DBMS_RESOURCE_MANAGER
- DBMS_RESOURCE_MANAGER_PRIVS

Assigning Sessions to Resource Consumer Groups

This section describes the automatic and manual methods that database administrators, users, and applications can use to assign sessions to resource consumer groups. When a session is assigned to a resource consumer group, Oracle Database Resource Manager (the Resource Manager) can manage resource allocation for it.

Note: Sessions that are not assigned to a consumer group are placed in the consumer group OTHER_GROUPS.

This section includes the following topics:

- Overview of Assigning Sessions to Resource Consumer Groups
- Assigning an Initial Resource Consumer Group
- Specifying Session-to–Consumer Group Mapping Rules
- Switching Resource Consumer Groups
- Specifying Automatic Resource Consumer Group Switching
- Granting and Revoking the Switch Privilege

Overview of Assigning Sessions to Resource Consumer Groups

Before you enable the Resource Manager, you must specify how user sessions are assigned to resource consumer groups. You do this by creating *mapping rules* that enable the Resource Manager to automatically assign each session to a consumer group upon session startup, based upon session attributes. After a session is assigned to its initial consumer group and is running, you can call a procedure to manually switch the session to a different consumer group. You would typically do this if the session is using excessive resources and must be moved to a consumer group that is more limited in its resource allocation. You can also grant the *switch privilege* to users and to applications so that they can switch their sessions from one consumer group to another.

The database can also automatically switch a session from one consumer group to another (typically lower priority) consumer group when there are changes in session attributes or when a session exceeds designated resource consumption limits.

Assigning an Initial Resource Consumer Group

The initial consumer group of a session is determined by the mapping rules that you configure. For information on how to configure mapping rules, see "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8.

Specifying Session-to–Consumer Group Mapping Rules

This section provides background information about session-to–consumer group mapping rules, and describes how to create and prioritize them. The following topics are covered:

- About Session-to–Consumer Group Mapping Rules
- Creating Consumer Group Mapping Rules
- Modifying and Deleting Consumer Group Mapping Rules
- Creating Mapping Rule Priorities

About Session-to–Consumer Group Mapping Rules

By creating session-to–consumer group mapping rules, you can:

- Specify the initial consumer group for a session based on session attributes.
- Enable the Resource Manager to dynamically switch a running session to another consumer group based on changing session attributes.

The mapping rules are based on session attributes such as the user name, the service that the session used to connect to the database, or the name of the client program.

To resolve conflicts among mapping rules, the Resource Manager orders the rules by priority. For example, suppose user `SCOTT` connects to the database with the `SALES` service. If one mapping rule states that user `SCOTT` starts in the `MED_PRIORITY` consumer group, and another states that sessions that connect with the `SALES` service start in the `HIGH_PRIORITY` consumer group, mapping rule priorities resolve this conflict.

There are two types of session attributes upon which mapping rules are based: login attributes and run-time attributes. The login attributes are meaningful only at session login time, when the Resource Manager determines the initial consumer group of the session. Run-time attributes apply any time during and after session login. You can

reassign a logged in session to another consumer group by changing any of its run-time attributes.

You use the SET_CONSUMER_GROUP_MAPPING and SET_CONSUMER_GROUP_MAPPING_PRI procedures to configure the automatic assignment of sessions to consumer groups. You must use a pending area for these procedures. (You must create the pending area, run the procedures, optionally validate the pending area, and then submit the pending area. For examples of using the pending area, see "Creating a Complex Resource Plan" on page 27-25.)

A session is automatically switched to a consumer group through mapping rules at distinct points in time:

- When the session first logs in, the mapping rules are evaluated to determine the initial group of the session.
- If a session attribute is dynamically changed to a new value (which is only possible for run-time attributes), then the mapping rules are reevaluated, and the session might be switched to another consumer group.

Predefined Consumer Group Mapping Rules

Each Oracle database comes with a set of predefined consumer group mapping rules:

- As described in "About Resource Consumer Groups" on page 27-3, all sessions created by user accounts SYS or SYSTEM are initially mapped to the SYS_GROUP consumer group.
- Sessions performing a data load with Data Pump or performing backup or copy operations with RMAN are automatically mapped to the predefined consumer groups designated in Table 27–6 on page 27-60.

You can use the DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING procedure to modify or delete any of these predefined mapping rules.

See Also:

- "Assigning an Initial Resource Consumer Group" on page 27-8
- "Specifying Automatic Switching with Mapping Rules" on page 27-14

Creating Consumer Group Mapping Rules

You use the SET_CONSUMER_GROUP_MAPPING procedure to map a session attribute/value pair to a consumer group. The parameters for this procedure are the following:

Parameter	Description
ATTRIBUTE	The session attribute type, specified as a package constant
VALUE	The value of the attribute
CONSUMER_GROUP	The consumer group to map to for this attribute/value pair

ATTRIBUTE can be one of the following:

Attribute	Type	Description
ORACLE_USER	Login	The Oracle Database user name
SERVICE_NAME	Login	The database service name used by the client to establish a connection

Attribute	Type	Description
CLIENT_OS_USER	Login	The operating system user name of the client that is logging in
CLIENT_PROGRAM	Login	The name of the client program used to log in to the server
CLIENT_MACHINE	Login	The name of the computer from which the client is making the connection
CLIENT_ID	Login	The client identifier for the session The client identifier session attribute is set by the DBMS_SESSION.SET_IDENTIFIER procedure.
MODULE_NAME	Run-time	The module name in the currently running application as set by the DBMS_APPLICATION_INFO.SET_MODULE procedure or the equivalent OCI attribute setting
MODULE_NAME_ACTION	Run-time	A combination of the current module and the action being performed as set by either of the following procedures or their equivalent OCI attribute setting: ■ DBMS_APPLICATION_INFO.SET_MODULE ■ DBMS_APPLICATION_INFO.SET_ACTION The attribute is specified as the module name followed by a period (.), followed by the action name (*module_name.action_name*).
SERVICE_MODULE	Run-time	A combination of service and module names in this form: *service_name.module_name*
SERVICE_MODULE_ACTION	Run-time	A combination of service name, module name, and action name, in this form: *service_name.module_name.action_name*
ORACLE_FUNCTION	Run-time	An RMAN or Data Pump operation. Valid values are DATALOAD, BACKUP, and COPY. There are predefined mappings for each of these values. If your session is performing any of these functions, it is automatically mapped to a predefined consumer group. See Table 27–6 on page 27-60 for details.

For example, the following PL/SQL block causes user SCOTT to map to the DEV_GROUP consumer group every time that he logs in:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.ORACLE_USER, 'SCOTT', 'DEV_GROUP');
END;
/
```

Again, you must create a pending area before running the SET_CONSUMER_GROUP_MAPPING procedure.

You can use wildcards for the value of most attributes in the value parameter in the SET_CONSUMER_GROUP_MAPPING procedure. Specifically, you can use wildcards for the value of all attributes except for ORACLE_USER, SERVICE_MODULE, and SERVICE_MODULE_ACTION. To specify values with wildcards, use the same semantics as the SQL LIKE operator. Specifically, wildcards use the following semantics:

- % for a multicharacter wildcard
- _ for a single character wildcard

- \ to escape the wildcards

Modifying and Deleting Consumer Group Mapping Rules

To modify a consumer group mapping rule, run the SET_CONSUMER_GROUP_MAPPING procedure against the desired attribute/value pair, specifying a new consumer group. To delete a rule, run the SET_CONSUMER_GROUP_MAPPING procedure against the desired attribute/value pair and specify a NULL consumer group.

Creating Mapping Rule Priorities

To resolve conflicting mapping rules, you can establish a priority ordering of the session attributes from most important to least important. You use the SET_CONSUMER_GROUP_MAPPING_PRI procedure to set the priority of each attribute to a unique integer from 1 (most important) to 12 (least important). The following example illustrates this setting of priorities:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
    EXPLICIT => 1,
    SERVICE_MODULE_ACTION => 2,
    SERVICE_MODULE => 3,
    MODULE_NAME_ACTION => 4,
    MODULE_NAME => 5,
    SERVICE_NAME => 6,
    ORACLE_USER => 7,
    CLIENT_PROGRAM => 8,
    CLIENT_OS_USER => 9,
    CLIENT_MACHINE => 10,
    CLIENT_ID => 11,
    ORACLE_FUNCTION => 12);
END;
/
```

In this example, the priority of the database user name is set to 7 (less important), while the priority of the module name is set to 5 (more important).

> **Note:** SET_CONSUMER_GROUP_MAPPING_PRI requires that you include the pseudo-attribute EXPLICIT as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures, which are described in detail in *Oracle Database PL/SQL Packages and Types Reference*:
>
> - DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP
> - DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS
> - DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER

To illustrate how mapping rule priorities work, continuing with the previous example, assume that in addition to the mapping of user SCOTT to the DEV_GROUP consumer group, there is also a module name mapping rule as follows:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.MODULE_NAME, 'EOD_REPORTS', 'LOW_PRIORITY');
END;
/
```

Now if the application in user SCOTT's session sets its module name to EOD_REPORTS, the session is reassigned to the LOW_PRIORITY consumer group, because module name mapping has a higher priority than database user mapping.

You can query the view DBA_RSRC_MAPPING_PRIORITY to see the current priority ordering of session attributes.

To prevent unauthorized clients from setting their session attributes so that they map to higher priority consumer groups, user switch privileges for consumer groups are enforced. Thus, even though the attribute of a particular session matches a mapping pair, the mapping rule is ignored if the session does not have the switch privilege for the designated consumer group.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about setting the module name with the DBMS_APPLICATION_INFO.SET_MODULE procedure
- "Granting and Revoking the Switch Privilege" on page 27-16

Switching Resource Consumer Groups

This section describes ways to switch the resource consumer group of a session.

This section contains the following topics:

- Manually Switching Resource Consumer Groups
- Enabling Users or Applications to Manually Switch Consumer Groups

Manually Switching Resource Consumer Groups

The DBMS_RESOURCE_MANAGER PL/SQL package provides two procedures that enable you 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 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 (terminating) a session of a user who is using excessive CPU, you can change that user's consumer group to one that is allocated fewer resources.

Switching a Single Session The SWITCH_CONSUMER_GROUP_FOR_SESS procedure causes the specified session to immediately be moved into the specified resource consumer group. In effect, this procedure can raise or lower priority of the session.

The following PL/SQL block switches a specific session to a new consumer group. The session identifier (SID) is 17, the session serial number (SERIAL#) is 12345, and the new consumer group is the HIGH_PRIORITY consumer group.

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345',
   'HIGH_PRIORITY');
END;
/
```

The SID, session serial number, and current resource consumer group for a session are viewable using the V$SESSION view.

See Also: *Oracle Database Reference* for details about the V$SESSION view.

Switching All Sessions for a User The SWITCH_CONSUMER_GROUP_FOR_USER procedure changes the resource consumer group for all sessions pertaining to the specified user name. The following PL/SQL block switches all sessions that belong to user HR to the LOW_GROUP consumer group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('HR',
    'LOW_GROUP');
END;
/
```

Enabling Users or Applications to Manually Switch Consumer Groups

You can grant a user the switch privilege so that he can switch his current consumer group using the SWITCH_CURRENT_CONSUMER_GROUP procedure in the DBMS_SESSION package. A user can run this procedure from an interactive session, for example from SQL*Plus, or an application can call this procedure to switch its session, effectively dynamically changing its priority.

The SWITCH_CURRENT_CONSUMER_GROUP procedure enables users to switch to only those consumer groups 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 the following:

Parameter	Description
NEW_CONSUMER_GROUP	The consumer group to which the user is switching.
OLD_CONSUMER_GROUP	Returns the name of the consumer group from which the user switched. Can be used to switch back later.
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, raises an error.

The following SQL*Plus session illustrates switching to a new consumer group. By printing the value of the output parameter old_group, the example illustrates how the old consumer group name is saved.

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
  DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('BATCH_GROUP', old_group, FALSE);
  DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
END;
/
```

The following line is output:

```
OLD GROUP = OLTP_GROUP
```

Note that the Resource Manager considers a switch to have taken place even if the SWITCH_CURRENT_CONSUMER_GROUP procedure is called to switch the session to the consumer group that it is already in.

Note: The 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 the consumer group assignment of a session at session startup, or as particular modules are called.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for additional examples and more information about the `DBMS_SESSION` package

Specifying Automatic Resource Consumer Group Switching

You can configure the Resource Manager to automatically switch a session to another consumer group when a certain condition is met. Automatic switching can occur when:

- A session attribute changes, causing a new mapping rule to take effect.
- A session exceeds the CPU or I/O resource consumption limits set by its consumer group.

The following sections provide details:

- Specifying Automatic Switching with Mapping Rules
- Specifying Automatic Switching by Setting Resource Limits

Specifying Automatic Switching with Mapping Rules

If a session attribute changes while the session is running, then the session-to–consumer group mapping rules are reevaluated. If a new rule takes effect, then the session might be moved to a different consumer group. See "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8 for more information.

Specifying Automatic Switching by Setting Resource Limits

This section describes managing runaway sessions or calls that use CPU or I/O resources beyond a specified limit. A runaway session is a SQL query, while a runaway call is a PL/SQL call.

When you create a resource plan directive for a consumer group, you can specify limits for CPU and I/O resource consumption for sessions in that group. You can then specify the action that is to be taken if any single call within a session exceeds one of these limits. The possible actions are the following:

- The session is dynamically switched to a designated consumer group.

  The target consumer group is typically one that has lower resource allocations. The session's user must have *switch privileges* on the new consumer group, otherwise the switch cannot occur. See "Granting and Revoking the Switch Privilege" on page 27-16 for more information.
- The session is killed (terminated).
- The session's current SQL statement is aborted.

The following are the resource plan directive attributes that are involved in this type of automatic session switching.

- `SWITCH_GROUP`

- SWITCH_TIME
- SWITCH_ESTIMATE
- SWITCH_IO_MEGABYTES
- SWITCH_IO_REQS
- SWITCH_FOR_CALL

See "Creating Resource Plan Directives" on page 27-29 for descriptions of these attributes.

Switches occur for sessions that are running and consuming resources, not waiting for user input or waiting for CPU cycles. After a session is switched, it continues in the target consumer group until it becomes idle, at which point it is switched back to its original consumer group. However, if SWITCH_FOR_CALL is set to TRUE, then the Resource Manager does not wait until the session is idle to return it to its original resource consumer group. Instead, the session is returned when the current top-level call completes. A **top-level call** in PL/SQL is an entire PL/SQL block treated as one call. A top-level call in SQL is an individual SQL statement.

The Resource Manager views a session as idle if a certain amount of time passes between calls. This time interval is not configurable.

SWITCH_FOR_CALL is useful for three-tier applications where the middle tier server is using session pooling.

A switched 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.

The following are examples of automatic switching based on resource limits:

Example 1

The following PL/SQL block creates a resource plan directive for the OLTP group that switches any session in that group to the LOW_GROUP consumer group if a call in the sessions exceeds 5 seconds of CPU time. This example prevents unexpectedly long queries from consuming too many resources. The switched-to consumer group is typically one with lower resource allocations.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'LOW_GROUP',
   SWITCH_TIME      => 5);
END;
/
```

Example 2

The following PL/SQL block creates a resource plan directive for the OLTP group that temporarily switches any session in that group to the LOW_GROUP consumer group if the session exceeds 10,000 I/O requests or exceeds 2,500 Megabytes of data transferred. The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                => 'DAYTIME',
   GROUP_OR_SUBPLAN    => 'OLTP',
```

```
   COMMENT             => 'OLTP group',
   MGMT_P1             => 75,
   SWITCH_GROUP        => 'LOW_GROUP',
   SWITCH_IO_REQS      => 10000,
   SWITCH_IO_MEGABYTES => 2500,
   SWITCH_FOR_CALL     => TRUE);
END;
/
```

Example 3

The following PL/SQL block creates a resource plan directive for the OLTP group that kills (terminates) any session that exceeds 60 seconds of CPU time. This example prevents runaway queries from consuming too many resources.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'KILL_SESSION',
   SWITCH_TIME      => 60);
END;
/
```

See Also: "Creating Resource Plan Directives" on page 27-29

Granting and Revoking the Switch Privilege

Using the DBMS_RESOURCE_MANAGER_PRIVS PL/SQL package, you can grant or revoke the switch privilege to a user, role, or PUBLIC. The switch privilege enables a user or application to switch a session to a specified resource consumer group. It also enables the database to automatically switch a session to a consumer group specified in a session-to–consumer group mapping rule or specified in the SWITCH_GROUP parameter of a resource plan directive. The package also enables you to revoke the switch privilege. The relevant package procedures are listed in the following table.

Procedure	Description
GRANT_SWITCH_CONSUMER_GROUP	Grants permission to a user, role, or PUBLIC to switch to a specified resource consumer group.
REVOKE_SWITCH_CONSUMER_GROUP	Revokes permission for a user, role, or PUBLIC to switch to a specified resource consumer group.

OTHER_GROUPS has switch privileges granted to PUBLIC. Therefore, all users are automatically granted the switch privilege for this consumer group.

See Also:

- "Enabling Users or Applications to Manually Switch Consumer Groups" on page 27-13
- "Specifying Automatic Resource Consumer Group Switching" on page 27-14

Granting the Switch Privilege

The following example grants user SCOTT the privilege to switch to consumer group OLTP.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   GRANTEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP',
   GRANT_OPTION   =>  TRUE);
END;
/
```

User SCOTT is also granted permission to grant switch privileges for OLTP to others.

If you grant permission to a role to switch to a particular resource consumer group, then any user who is granted that role and has enabled that role can switch his session to that 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 OLTP.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   REVOKEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP');
END;
/
```

If you revoke a user's switch privileges for a particular consumer group, any subsequent attempts by that user to switch to that consumer group, either manually or automatically through consumer group mapping rules, will fail. The user's session will then be automatically assigned to OTHER_GROUPS.

If you revoke from a role the switch privileges to a consumer group, any users who had switch privileges for the consumer group only through that role are no longer able to switch to that consumer group.

If you revoke switch privileges to a consumer group from PUBLIC, any users other than those who are explicitly assigned switch privileges either directly or through a role are no longer able to switch to that consumer group.

The Types of Resources Managed by the Resource Manager

Resource plan directives specify how resources are allocated to resource consumer groups or subplans. Each directive can specify several different methods for allocating resources to its consumer group or subplan. The following sections summarize these resource allocation methods:

- CPU
- Degree of Parallelism Limit
- Parallel Target Percentage
- Parallel Queue Timeout
- Active Session Pool with Queuing
- Automatic Consumer Group Switching

- Canceling SQL and Terminating Sessions
- Execution Time Limit
- Undo Pool
- Idle Time Limit

CPU

To manage CPU resources, Resource Manager allocates resources among consumer groups and redistributes CPU resources that were allocated but were not used. You can also set a limit on the amount of CPU resources that can be allocated to a particular consumer group.

Resource Manager provides the following resource plan directive attributes to control CPU resource allocation:

- Management Attributes
- Maximum Utilization Limit

Management Attributes

Management attributes enable you to specify how CPU resources are to be allocated among consumer groups and subplans. Multiple levels of CPU resource allocation (up to eight levels) provide a means of prioritizing CPU usage within a plan. Consumer groups and subplans at level 2 get resources that were not allocated at level 1 or that were allocated at level 1 but were not completely consumed by the consumer groups or subplans at level 1. Similarly, resource consumers at level 3 are allocated resources only when some allocation remains from levels 1 and 2. The same rules apply to levels 4 through 8. 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.

Use the management attributes `MGMT_P`*n*, where *n* is an integer between 1 and 8, to specify multiple levels of CPU resource allocation. For example, use the `MGMT_P1` directive attribute to specify CPU resource allocation at level 1 and `MGMT_P2` directive attribute to specify resource allocation at level 2.

Use management attributes with parallel statement directive attributes, such as Degree of Parallelism Limit and Parallel Target Percentage, to control parallel statement queuing. When parallel statement queuing is used, management attributes are used to determine which consumer group is allowed to issue the next parallel statement. For example, if you set the `MGMT_P1` directive attribute for a consumer group to 80, that group has an 80% chance of issuing the next parallel statement.

> **See Also:** *Oracle Database VLDB and Partitioning Guide* for information about parallel statement queuing

Management attributes also enable you to specify CPU resource allocation for Exadata I/O.

> **See Also:** The Exadata documentation for information about using management attributes for Exadata I/O

Table 27–1 illustrates a simple resource plan with three levels.

Table 27–1 A Simple Three-Level Resource Plan

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation
HIGH_GROUP	80%		
LOW_GROUP		50%	
MAINT_SUBPLAN		50%	
OTHER_GROUPS			100%

High priority applications run within HIGH_GROUP, which is allocated 80% of CPU. Because HIGH_GROUP is at level one, it gets priority for CPU utilization, but only up to 80% of CPU. This leaves a remaining 20% of CPU to be shared 50-50 by LOW_GROUP and the MAINT_SUPLAN at level 2. Any unused allocation from levels 1 and 2 are then available to OTHER_GROUPS at level 3. Because OTHER_GROUPS has no sibling consumer groups or subplans at its level, 100% is specified.

Within a particular level, CPU allocations are not fixed. If there is not sufficient load in a particular consumer group or subplan, residual CPU can be allocated to remaining consumer groups or subplans. Thus, when there is only one level, unused allocation by any consumer group or subplan can be redistributed to other "sibling" consumer groups or subplans. If there are multiple levels, then the unused allocation is distributed to the consumer groups or subplans at the next level. If the last level has unused allocations, these allocations can be redistributed to all other levels in proportion to their designated allocations.

As an example of redistribution of unused allocations from one level to another, if during a particular period, HIGH_GROUP consumes only 25% of CPU, then 75% is available to be shared by LOW_GROUP and MAINT_SUBPLAN. Any unused portion of the 75% at level 2 is then made available to OTHER_GROUPS at level 3. However, if OTHER_GROUPS has no session activity at level 3, then the 75% at level 2 can be redistributed to all other consumer groups and subplans in the plan proportionally.

Maximum Utilization Limit

In the previous scenario, suppose that due to inactivity elsewhere, LOW_GROUP acquires 90% of CPU. Suppose that you do not want to allow LOW_GROUP to use 90% of the server because you do not want non-critical sessions to inundate the CPUs. The MAX_UTILIZATION_LIMIT attribute of resource plan directives can prevent this situation.

Use the MAX_UTILIZATION_LIMIT attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

Setting the MAX_UTILIZATION_LIMIT attribute is optional. If you omit this attribute for a consumer group, there is no limit on the amount of CPU that the consumer group can use. Therefore, if all the other applications are idle, a consumer group that does not have MAX_UTILIZATION_LIMIT set can be allocated 100% of the CPU resources.

You can also use the MAX_UTILIZATION_LIMIT attribute as the sole means of limiting CPU utilization for consumer groups, without specifying level limits.

Table 27–2 shows a variation of the previous plan. In this plan, using MAX_UTILIZATION_LIMIT, CPU utilization is capped at 75% for LOW_GROUP, 50% for MAINT_SUBPLAN, and 75% for OTHER_GROUPS. (Note that the sum of all maximum utilization limits can exceed 100%. Each limit is applied independently.)

Table 27–2 A Three-Level Resource Plan with Maximum Utilization Limits

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	Maximum Utilization Limit
HIGH_GROUP	80%			
LOW_GROUP		50%		75%
MAINT_SUBPLAN		50%		50%
OTHER_GROUPS			100%	75%

In the example described in Table 27–2, if HIGH_GROUP is using only 10% of the CPU at a given time, then the remaining 90% is available to LOW_GROUP and the consumer groups in MAINT_SUBPLAN at level 2. If LOW_GROUP uses only 20% of the CPU, then 70% can be allocated to MAINT_SUBPLAN. However, MAINT_SUBPLAN has a MAX_UTILIZATION_LIMIT of 50%. Therefore, even though more CPU resources are available, the server cannot allocate more than 50% of the CPU to the consumer groups that belong to the subplan MAINT_SUBPLAN.

You can set MAX_UTILIZATION_LIMIT for both a subplan and the consumer groups that the subplan contains. In such cases, the limit for a consumer group is computed using the limits specified for the subplan and that consumer group. For example, the MAINT_SUBPLAN contains the consumer groups MAINT_GROUP1 and MAINT_GROUP2. MAINT_GROUP1 has MAX_UTILIZATION_LIMIT set to 40%. However, the limit for MAINT_SUBPLAN is set to 50%. Therefore, the limit for consumer group MAINT_GROUP1 is computed as 40% of 50%, or 20%. For an example of how to compute MAX_UTILIZATION_LIMIT for a consumer group when limits are specified for both the consumer group and the subplan to which the group belongs, see "Example 4 - Specifying a Maximum Utilization Limit for Consumer Groups and Subplans" on page 27-42.

See Also:

- "Creating Resource Plan Directives" on page 27-29
- "Putting It All Together: Oracle Database Resource Manager Examples" on page 27-37

Degree of Parallelism Limit

You can limit the maximum degree of parallelism for any operation within a consumer group. The degree of parallelism is the number of parallel execution servers that are associated with a single operation. Use the PARALLEL_DEGREE_LIMIT_P1 directive attribute to specify the degree of parallelism for a consumer group.

See Also: *Oracle Database VLDB and Partitioning Guide* for more information about degree of parallelism in producer/consumer operations

The degree of parallelism limit applies to one operation within a consumer group; it does not limit the total degree of parallelism across all operations within the consumer group. However, you can combine both the PARALLEL_DEGREE_LIMIT_P1 and the PARALLEL_TARGET_PERCENTAGE directive attributes to achieve the desired control. For more information about the PARALLEL_TARGET_PERCENTAGE attribute, see "Parallel Target Percentage" on page 27-21.

Parallel Target Percentage

It is possible for a single consumer group to launch enough parallel statements to use all the available parallel servers. If this happens, when a high-priority parallel statement from a different consumer group is run, then no parallel servers are available to allocate to this group. You can avoid such a scenario by limiting the number of parallel servers that can be used by a particular consumer group.

> **Note:** This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).

Use the `PARALLEL_TARGET_PERCENTAGE` directive attribute to specify the maximum percentage of the parallel server pool that a particular consumer group can use. The number of parallel servers used by a particular consumer group is counted as the sum of the parallel servers used by all sessions in that consumer group.

> **See Also:** *Oracle Database VLDB and Partitioning Guide* for information about parallel statement queuing

For example, assume that the total number of parallel servers is 32, as set by the `PARALLEL_SERVERS_TARGET` initialization parameter, and the `PARALLEL_TARGET_PERCENTAGE` directive attribute for the consumer group `MY_GROUP` is set to 50%. This consumer group can use a maximum of 50% of 32, or 16 parallel servers.

If your resource plan has management attributes (`MGMT_P1`, `MGMT_P2`, and so on), then a separate parallel statement queue is managed as a First In First Out (FIFO) queue for each management attribute.

If your resource plan does not have any management attributes, then a single parallel statement queue is managed as a FIFO queue.

In the case of an Oracle Real Application Clusters (Oracle RAC) environment, the target number of parallel servers is the sum of (`PARALLEL_TARGET_PERCENTAGE` * `PARALLEL_SERVERS_TARGET` / 100) across all Oracle RAC instances. If a consumer group is using the number of parallel servers computed above or more, then it has exceeded its limit, and its parallel statements will be queued.

If a consumer group does not have any parallel statements running within an Oracle RAC database, then the first parallel statement is allowed to exceed the limit specified by `PARALLEL_TARGET_PERCENTAGE`.

> **Note:** In an Oracle Real Application Clusters (Oracle RAC) environment, the `PARALLEL_TARGET_PERCENTAGE` attribute applies to the entire cluster and not to a single instance.

Managing Parallel Statement Queuing Using Parallel Target Percentage

The `PARALLEL_TARGET_PERCENTAGE` attribute enables you to specify when parallel statements from a consumer group can be queued. Oracle Database maintains a separate parallel statement queue for each consumer group.

A parallel statement from a consumer group is not run and instead added to the parallel statement queue of that consumer group if the following conditions are met:

- `PARALLEL_DEGREE_POLICY` is set to `AUTO`.

Setting this parameter to AUTO enables automatic degree of parallelism (Auto DOP), parallel statement queuing, and in-memory parallel execution.

Note that parallel statements which have PARALLEL_DEGREE_POLICY set to MANUAL or LIMITED are executed immediately and not added to the parallel statement queue.

- The number of active parallel servers across all consumer groups exceeds the PARALLEL_SERVERS_TARGET initialization parameter setting. This condition applies regardless of whether you specify PARALLEL_TARGET_PERCENTAGE. If PARALLEL_TARGET_PERCENTAGE is not specified, then it defaults to 100%.
- The sum of the number of active parallel servers for the consumer group and the degree of parallelism of the parallel statement exceeds the target number of active parallel servers.

  The target number of active parallel servers is computed as follows:

  PARALLEL_TARGET_PERCENTAGE/100 * PARALLEL_SERVERS_TARGET

Parallel Queue Timeout

When you use parallel statement queuing, if the database does not have sufficient resources to execute a parallel statement, the statement is queued until the required resources become available. However, there is a chance that a parallel statement may be waiting in the parallel statement queue for longer than is desired. You can prevent such scenarios by specifying the maximum time a parallel statement can wait in the parallel statement queue.

Note: This functionality is available starting with Oracle Database 11g Release 2 (11.2.0.2).

The PARALLEL_QUEUE_TIMEOUT directive attribute enables you to specify the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out. The PARALLEL_QUEUE_TIMEOUT attribute can be set for each consumer group. This attribute is applicable even if you do not specify other management attributes (MGMT_P1, MGMT_P2, and so on) in your resource plan.

See Also: *Oracle Database VLDB and Partitioning Guide* for more information about parallel statement queuing

Note: Because the parallel statement queue is clusterwide, all directives related to the parallel statement queue are also clusterwide.

When a parallel statement is timed out, the statement execution ends with the following error message:

```
ORA-07454: queue timeout, n second(s), exceeded
```

If you want more per-workload management, then you must use the following directive attributes:

- MGMT_P*n*

  Management attributes control how a parallel statement is selected from the parallel statement queue for execution. You can prioritize the parallel statements

of one consumer group over another by setting a higher value for the management attributes of that group.

- PARALLEL_TARGET_PERCENTAGE
- PARALLEL_QUEUE_TIMEOUT
- PARALLEL_DEGREE_LIMIT_P1

> **See Also:** "Example of Managing Parallel Statements Using Directive Attributes" on page 27-45 for more information about the combined use of all the parallel server directive attributes

Although parallel server usage is monitored for all sessions, the parallel server directive attributes you set affect only sessions for which parallel statement queuing is enabled (PARALLEL_DEGREE_POLICY is set to AUTO). If a session has the PARALLEL_DEGREE_POLICY set to MANUAL, parallel statements from this session are not queued. However, any parallel servers used by such sessions are included in the count that is used to determine the limit for PARALLEL_TARGET_PERCENTAGE. Even if this limit is exceeded, parallel statements from this session are not queued.

Active Session Pool with Queuing

You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum defines the **active session pool**. An **active session** is a session that is actively processing a transaction or SQL statement. Specifically, an active session is either in a transaction, holding a user enqueue, or has an open cursor and has not been idle for over 5 seconds. An active session is considered active even if it is blocked, for example waiting for an I/O request to complete. When the active session pool is full, a session that is trying to process a call is placed into a queue. When an active session completes, the first session in the queue can then be removed from the queue and scheduled for execution. You can also specify a period after which a session in the execution queue times out, causing the call to terminate with an error.

Active session limits should not be used for OLTP workloads. In addition, active session limits should not be used to implement connection pooling or parallel statement queuing.

To manage parallel statements, you must use parallel statement queuing with the PARALLEL_TARGET_PERCENTAGE attribute and management attributes (MGMT_P1, MGMT_P2, and so on).

Automatic Consumer Group Switching

This method enables you to control resource allocation by specifying criteria that, if met, causes the automatic switching of a session to a specified consumer group. Typically, this method is used to switch a session from a high-priority consumer group—one that receives a high proportion of system resources—to a lower priority consumer group because that session exceeded the expected resource consumption for a typical session in the group.

See "Specifying Automatic Switching by Setting Resource Limits" on page 27-14 for more information.

Canceling SQL and Terminating Sessions

You can also specify directives to cancel long-running SQL queries or to terminate long-running sessions based on the amount of system resources consumed. See

"Specifying Automatic Switching by Setting Resource Limits" on page 27-14 for more information.

Execution Time Limit

You can specify a maximum execution time allowed for an operation. If the database estimates that an operation will run longer than the specified maximum execution time, then 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 total amount of undo for uncommitted transactions 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 undo is terminated. No other members of the consumer group can perform further data manipulation until undo space is freed from the pool.

Idle Time Limit

You can specify an amount of time that a session can be idle, after which it is terminated. You can also specify a more stringent idle time limit that applies to sessions that are idle and blocking other sessions.

Creating a Simple Resource Plan

You can quickly create a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure. This procedure enables you to both create consumer groups and allocate resources to them by executing a single procedure call. 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, specifying resource plan directives, and so on.

You specify the following arguments for the `CREATE_SIMPLE_PLAN` procedure:

Parameter	Description
`SIMPLE_PLAN`	Name of the plan
`CONSUMER_GROUP1`	Consumer group name for first group
`GROUP1_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP2`	Consumer group name for second group
`GROUP2_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP3`	Consumer group name for third group
`GROUP3_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP4`	Consumer group name for fourth group
`GROUP4_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP5`	Consumer group name for fifth group
`GROUP5_PERCENT`	CPU resource allocated to this group
`CONSUMER_GROUP6`	Consumer group name for sixth group

Parameter	Description
GROUP6_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP7	Consumer group name for seventh group
GROUP7_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP8	Consumer group name for eighth group
GROUP8_PERCENT	CPU resource allocated to this group

You can specify up to eight consumer groups with this procedure. The only resource allocation method supported is CPU. The plan uses the EMPHASIS CPU allocation policy (the default) and each consumer group uses the ROUND_ROBIN scheduling policy (also the default). Each consumer group specified in the plan is allocated its CPU percentage at level 2. Also implicitly included in the plan are SYS_GROUP (a system-defined group that is the initial consumer group for the users SYS and SYSTEM) and OTHER_GROUPS. The SYS_GROUP consumer group is allocated 100% of the CPU at level 1, and OTHER_GROUPS is allocated 100% of the CPU at level 3.

Example: Creating a Simple Plan with the CREATE_SIMPLE_PLAN Procedure

The following PL/SQL block creates a simple resource plan with two user-specified consumer groups:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'SIMPLE_PLAN1',
   CONSUMER_GROUP1 => 'MYGROUP1', GROUP1_PERCENT => 80,
   CONSUMER_GROUP2 => 'MYGROUP2', GROUP2_PERCENT => 20);
END;
/
```

Executing the preceding statements creates the following plan:

Consumer Group	Level 1	Level 2	Level 3
SYS_GROUP	100%	-	-
MYGROUP1	-	80%	-
MYGROUP2	-	20%	-
OTHER_GROUPS	-	-	100%

See Also:

- "Creating a Resource Plan" on page 27-28 for more information on the EMPHASIS CPU allocation policy
- "Creating Resource Consumer Groups" on page 27-27 for more information on the ROUND_ROBIN scheduling policy
- "Elements of the Resource Manager" on page 27-3

Creating a Complex Resource Plan

When your situation calls for a more complex resource plan, you must create the plan, with its directives and consumer groups, in a staging area called the pending area, and then validate the plan before storing it in the data dictionary.

The following is a summary of the steps required to create a complex resource plan.

> **Note:** A complex resource plan is any resource plan that is not created with the `DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN` procedure.

Step 1: Create a pending area.

Step 2: Create, modify, or delete consumer groups.

Step 3: Create the resource plan.

Step 4: Create resource plan directives.

Step 5: Validate the pending area.

Step 6: Submit the pending area.

You use procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package to complete these steps. The following sections provide details:

- About the Pending Area
- Creating a Pending Area
- Creating Resource Consumer Groups
- Creating a Resource Plan
- Creating Resource Plan Directives
- Validating the Pending Area
- Submitting the Pending Area
- Clearing the Pending Area

See Also:

- Predefined Consumer Group Mapping Rules on page 27-60
- *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- "Elements of the Resource Manager" on page 27-3

About the Pending Area

The **pending area** is a staging area where you can create a new resource plan, update an existing plan, or delete a plan without affecting currently running applications. When you create a pending area, the database initializes it and then copies existing plans into the pending area so that they can be updated.

> **Tip:** After you create the pending area, if you list all plans by querying the `DBA_RSRC_PLANS` data dictionary view, you see two copies of each plan: one with the `PENDING` status, and one without. The plans with the `PENDING` status reflect any changes you made to the plans since creating the pending area. Pending changes can also be viewed for consumer groups using `DBA_RSRC_CONSUMER_GROUPS` and for resource plan directives using `DBA_RSRC_PLAN_DIRECTIVES`. See Resource Manager Data Dictionary Views on page 27-60 for more information.

After you make changes in the pending area, you validate the pending area and then submit it. Upon submission, all pending changes are applied to the data dictionary, and the pending area is cleared and deactivated.

If you attempt to create, update, or delete a plan (or create, update, or delete consumer groups or resource plan directives) without first creating the pending area, you receive an error message.

Submitting the pending area does not activate any new plan that you create; it just stores new or updated plan information in the data dictionary. However, if you modify a plan that is currently active, the plan is reactivated with the new plan definition. See "Enabling Oracle Database Resource Manager and Switching Plans" on page 27-35 for information about activating a resource plan.

When you create a pending area, no other users can create one until you submit or clear the pending area or log out.

Creating a Pending Area

You create a pending area with the `CREATE_PENDING_AREA` procedure.

Example: Creating a pending area:

The following PL/SQL block creates and initializes a pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
END;
/
```

Creating Resource Consumer Groups

You create a resource consumer group using the `CREATE_CONSUMER_GROUP` procedure. You can specify the following parameters:

Parameter	Description
`CONSUMER_GROUP`	Name to assign to the consumer group.
`COMMENT`	Any comment.
`CPU_MTH`	Deprecated. Use `MGMT_MTH`.
`MGMT_MTH`	The resource allocation method for distributing CPU among sessions in the consumer group. The default is `'ROUND-ROBIN'`, which uses a round-robin scheduler to ensure that sessions are fairly executed. `'RUN-TO-COMPLETION'` specifies that long-running sessions are scheduled ahead of other sessions. This setting helps long-running sessions (such as batch processes) complete sooner.

Example: Creating a Resource Consumer Group

The following PL/SQL block creates a consumer group called `OLTP` with the default (`ROUND-ROBIN`) method of allocating resources to sessions in the group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   CONSUMER_GROUP => 'OLTP',
   COMMENT        => 'OLTP applications');
END;
/
```

See Also:

- "Updating a Consumer Group" on page 27-50
- "Deleting a Consumer Group" on page 27-50

Creating a Resource Plan

You create a resource plan with the CREATE_PLAN procedure. You can specify the parameters shown in the following table. The first two parameters are required. The remainder are optional.

Parameter	Description
PLAN	Name to assign to the plan.
COMMENT	Any descriptive comment.
CPU_MTH	Deprecated. Use MGMT_MTH.
ACTIVE_SESS_POOL_MTH	Active session pool resource allocation method. 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 the order in which queued inactive sessions are removed from the queue and added to the active session pool. FIFO_TIMEOUT is the default and only method available.
MGMT_MTH	Resource allocation method for specifying how much CPU each consumer group or subplan gets. 'EMPHASIS', the default method, is for single-level or multilevel plans that use percentages to specify how CPU is distributed among consumer groups. 'RATIO' is for single-level plans that use ratios to specify how CPU is distributed.
SUB_PLAN	If TRUE, the plan cannot be used as the top plan; it can be used as a subplan only. Default is FALSE.

Example: Creating a Resource Plan

The following PL/SQL block creates a resource plan named DAYTIME:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
   PLAN    => 'DAYTIME',
   COMMENT => 'More resources for OLTP applications');
END;
/
```

About the RATIO CPU Allocation Method

The RATIO method is an alternate CPU allocation method intended for simple plans that have only a single level of CPU allocation. Instead of percentages, you specify numbers corresponding to the ratio of CPU that you want to give to each consumer group. To use the RATIO method, you set the MGMT_MTH argument for the CREATE_PLAN

procedure to 'RATIO'. See "Creating Resource Plan Directives" on page 27-29 for an example of a plan that uses this method.

See Also:

- "Updating a Plan" on page 27-50
- "Deleting a Plan" on page 27-50

Creating Resource Plan Directives

You use the CREATE_PLAN_DIRECTIVE procedure to create resource plan directives. Each directive belongs to a plan or subplan and allocates resources to either a consumer group or subplan.

Note: The set of directives for a resource plan and its subplans can name a particular subplan only once.

You can specify directives for a particular consumer group in a top plan and its subplans. However, Oracle recommends that the set of directives for a resource plan and its subplans name a particular consumer group only once.

You can specify the following parameters:

Parameter	Description
PLAN	Name of the resource plan to which the directive belongs.
GROUP_OR_SUBPLAN	Name of the consumer group or subplan to which to allocate resources.
COMMENT	Any comment.
CPU_P1	Deprecated. Use MGMT_P1.
CPU_P2	Deprecated. Use MGMT_P2.
CPU_P3	Deprecated. Use MGMT_P3.
CPU_P4	Deprecated. Use MGMT_P4.
CPU_P5	Deprecated. Use MGMT_P5.
CPU_P6	Deprecated. Use MGMT_P6.
CPU_P7	Deprecated. Use MGMT_P7.
CPU_P8	Deprecated. Use MGMT_P8.
ACTIVE_SESS_POOL_P1	Specifies the maximum number of concurrently active sessions for a consumer group. Other sessions await execution in an inactive session queue. Default is UNLIMITED.
QUEUEING_P1	Specifies time (in seconds) after which a session in an inactive session queue (waiting for execution) times out and the call is aborted. Default is UNLIMITED.
PARALLEL_DEGREE_LIMIT_P1	Specifies a limit on the degree of parallelism for any operation. Default is UNLIMITED.

Parameter	Description
SWITCH_GROUP	Specifies the consumer group to which a session is switched if switch criteria are met. If the group name is 'CANCEL_SQL', then the current call is canceled when switch criteria are met. If the group name is 'KILL_SESSION', then the session is killed when switch criteria are met. Default is NULL. If the group name is 'CANCEL_SQL', the SWITCH_FOR_CALL parameter is always set to TRUE, overriding the user-specified setting.
SWITCH_TIME	Specifies the time (in CPU seconds) that a call can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_ESTIMATE	If TRUE, the database estimates the execution time of each call, and if estimated execution time exceeds SWITCH_TIME, the session is switched to the SWITCH_GROUP before beginning the call. Default is FALSE. The execution time estimate is obtained from the optimizer. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics. In general, you should expect statistics to be no more accurate than ± 10 minutes.
MAX_EST_EXEC_TIME	Specifies the maximum execution time (in CPU seconds) allowed for a call. If the optimizer estimates that a call will take longer than MAX_EST_EXEC_TIME, the call is not allowed to proceed and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is UNLIMITED. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics. In general, you should expect statistics to be no more accurate than ± 10 minutes.
UNDO_POOL	Sets a maximum in kilobytes (K) on the total amount of undo for uncommitted transactions that can be generated by a consumer group. Default is UNLIMITED.
MAX_IDLE_TIME	Indicates the maximum session idle time, in seconds. Default is NULL, which implies unlimited.
MAX_IDLE_BLOCKER_TIME	Indicates the maximum session idle time of a blocking session, in seconds. Default is NULL, which implies unlimited.
SWITCH_TIME_IN_CALL	Deprecated. Use SWITCH_FOR_CALL.
MGMT_P1	For a plan with the MGMT_MTH parameter set to EMPHASIS, specifies the CPU percentage to allocate at the first level. For MGMT_MTH set to RATIO, specifies the weight of CPU usage. Default is NULL for all MGMT_P*n* parameters.
MGMT_P2	For EMPHASIS, specifies CPU percentage to allocate at the second level. Not applicable for RATIO.
MGMT_P3	For EMPHASIS, specifies CPU percentage to allocate at the third level. Not applicable for RATIO.
MGMT_P4	For EMPHASIS, specifies CPU percentage to allocate at the fourth level. Not applicable for RATIO.
MGMT_P5	For EMPHASIS, specifies CPU percentage to allocate at the fifth level. Not applicable for RATIO.

Parameter	Description
MGMT_P6	For EMPHASIS, specifies CPU percentage to allocate at the sixth level. Not applicable for RATIO.
MGMT_P7	For EMPHASIS, specifies CPU percentage to allocate at the seventh level. Not applicable for RATIO.
MGMT_P8	For EMPHASIS, specifies CPU percentage to allocate at the eighth level. Not applicable for RATIO.
SWITCH_IO_MEGABYTES	Specifies the number of megabytes of I/O that a session can transfer (read and write) before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_IO_REQS	Specifies the number of I/O requests that a session can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_FOR_CALL	If TRUE, a session that was automatically switched to another consumer group (according to SWITCH_TIME, SWITCH_IO_MEGABYTES, or SWITCH_IO_REQS) is returned to its original consumer group when the top level call completes. Default is NULL.
MAX_UTILIZATION_LIMIT	Absolute maximum CPU utilization percentage permitted for the consumer group. This value overrides any level allocations for CPU (MGMT_P1 through MGMT_P8), and also imposes a limit on total CPU utilization when unused allocations are redistributed. You can specify this attribute and leave MGMT_P1 through MGMT_P8 NULL. You cannot specify this attribute for a subplan.
PARALLEL_TARGET_PERCENTAGE	Specifies the maximum percentage of the parallel server pool that a particular consumer group can use. The number of parallel servers used by a particular consumer group is counted as the sum of the parallel servers used by all sessions in that consumer group.
PARALLEL_QUEUE_TIMEOUT	Specifies the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.

Example 1:

The following PL/SQL block creates a resource plan directive for plan DAYTIME. (It assumes that the DAYTIME plan and OLTP consumer group are already created in the pending area.)

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75);
END;
/
```

This directive assigns 75% of CPU resources to the OLTP consumer group at level 1.

To complete the plan shown in Figure 27–1 on page 27-5, you would create the REPORTING consumer group, and then execute the following PL/SQL block:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
```

```
   GROUP_OR_SUBPLAN         => 'REPORTING',
   COMMENT                  => 'Reporting group',
   MGMT_P1                  => 15,
   PARALLEL_DEGREE_LIMIT_P1 => 8,
   ACTIVE_SESS_POOL_P1      => 4);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
   GROUP_OR_SUBPLAN         => 'OTHER_GROUPS',
   COMMENT                  => 'This one is required',
   MGMT_P1                  => 10);
END;
/
```

In this plan, consumer group REPORTING has a maximum degree of parallelism of 8 for any operation, while none of the other consumer groups are limited in their degree of parallelism. In addition, the REPORTING group has a maximum of 4 concurrently active sessions.

Example 2:

This example uses the RATIO method to allocate CPU, which uses ratios instead of percentages. Suppose your application suite offers three service levels to clients: Gold, Silver, and Bronze. You create three consumer groups named GOLD_CG, SILVER_CG, and BRONZE_CG, and you create the following resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    MGMT_MTH         => 'RATIO',
    COMMENT          => 'Plan that supports three service levels');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'GOLD_CG',
    COMMENT          => 'Gold service level customers',
    MGMT_P1          => 10);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'SILVER_CG',
    COMMENT          => 'Silver service level customers',
    MGMT_P1          => 5);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'BRONZE_CG',
    COMMENT          => 'Bronze service level customers',
    MGMT_P1          => 2);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
    (PLAN            => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT          => 'Lowest priority sessions',
    MGMT_P1          => 1);
END;
/
```

The ratio of CPU allocation is 10:5:2:1 for the GOLD_CG, SILVER_CG, BRONZE_CG, and OTHER_GROUPS consumer groups, respectively.

If sessions exist only in the GOLD_CG and SILVER_CG consumer groups, then the ratio of CPU allocation is 10:5 between the two groups.

How Resource Plan Directives Interact

You may have occasion to reference the same consumer group from the top plan and any number of subplans. This results in multiple resource plan directives referring to the same consumer group. Although this is allowed, Oracle strongly recommends that you avoid referencing the same consumer group from a top plan and any of its subplans.

When multiple resource plan directives refer to the same consumer group, the following rules apply:

- The parallel degree limit for the consumer group will be the *minimum* of all the incoming values.
- 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.
- The undo pool for the consumer group will be the *sum* of all the incoming values.
- If there is more than one `SWITCH_TIME`, `SWITCH_IO_MEGABYTES`, or `SWITCH_IO_REQS`, Oracle Database Resource Manager (the Resource Manager) chooses the *most restrictive* of all incoming values. Specifically:
  - `SWITCH_TIME` = *`min`* (all incoming `SWITCH_TIME` values)
  - `SWITCH_IO_MEGABYTES` = *`min`* (all incoming `SWITCH_IO_MEGABYTES` values)
  - `SWITCH_IO_REQS` = *`min`* (all incoming `SWITCH_IO_REQS` values)
  - `SWITCH_ESTIMATE = TRUE` overrides `SWITCH_ESTIMATE = FALSE`

  Note: If both plan directives specify the same switch time, but different switch groups, then the choice about which group to switch to is statically but arbitrarily decided by the Resource Manager.

- `SWITCH_FOR_CALL` is `TRUE` if any of the incoming values are `TRUE`.
- The maximum estimated execution time will be the *most restrictive* of all incoming values. Specifically:

  `MAX_EST_EXEC_TIME` = *`min`* (all incoming `MAX_EST_EXEC_TIME` values)
- The maximum idle time is the *minimum* of all incoming values.
- The maximum idle blocker time is the *minimum* of all incoming values.

See Also:

- "Updating a Resource Plan Directive" on page 27-51
- "Deleting a Resource Plan Directive" on page 27-51

Validating the Pending Area

At any time when you are making changes in the pending area, you can call `VALIDATE_PENDING_AREA` to ensure that the pending area is valid so far.

The following rules must be adhered to, and are checked by the validate procedure:

- No plan can contain any loops. A loop occurs when a subplan contains a directive that references a plan that is above the subplan in the plan hierarchy. For example, a subplan cannot reference the top plan.

- All plans and resource consumer groups referred to by plan directives must exist.
- All plans must have plan directives that point to either plans or resource consumer groups.
- All percentages in any given level must not add up to greater than 100.
- A plan that is currently being used as a top plan by an active instance cannot be deleted.
- The following 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
  - SWITCH_IO_REQS
  - SWITCH_IO_MEGABYTES
  - MAX_EST_EXEC_TIME
  - UNDO_POOL
  - MAX_IDLE_TIME
  - MAX_IDLE_BLOCKER_TIME
  - SWITCH_FOR_CALL
  - MAX_UTILIZATION_LIMIT
- There can be no more than 31 resource consumer groups in any active plan. Also, at most, a plan can have 31 children.
- Plans and resource consumer groups cannot have the same name.
- There must be a plan directive for OTHER_GROUPS somewhere in any active plan. This ensures that a session that is not part of any of the consumer groups included in the currently active plan is allocated resources (as specified by the directive for OTHER_GROUPS).

VALIDATE_PENDING_AREA raises an error if any of the preceding rules are violated. You can then make changes to fix any problems and call the 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 might be part of some plan to be implemented in the future.

Example: Validating the Pending Area:

The following PL/SQL block validates the pending area.

```
BEGIN
  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
END;
/
```

See Also: "About the Pending Area" on page 27-26

Submitting the Pending Area

After you have validated your changes, call the SUBMIT_PENDING_AREA procedure to make your changes active.

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 plans, 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 might 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.

Example: Submitting the Pending Area:

The following PL/SQL block submits the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

See Also: "About the Pending Area" on page 27-26

Clearing the Pending Area

There is also a procedure for clearing the pending area at any time. This PL/SQL block causes all of your changes to be cleared from the pending area and deactivates the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
END;
/
```

After calling CLEAR_PENDING_AREA, you must call the CREATE_PENDING_AREA procedure before you can again attempt to make changes.

See Also: "About the Pending Area" on page 27-26

Enabling Oracle Database Resource Manager and Switching Plans

You enable Oracle Database Resource Manager (the Resource Manager) by setting the RESOURCE_MANAGER_PLAN initialization parameter. This parameter specifies the top plan, identifying the plan to be used for the current instance. If no plan is specified with this parameter, the Resource Manager is not enabled.

By default the Resource Manager is not enabled, except during preconfigured maintenance windows, described later in this section.

The following statement in a text initialization parameter file activates the Resource Manager upon database startup and sets the top plan as mydb_plan.

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

You can also activate or deactivate the Resource Manager, or change the current top plan, using the DBMS_RESOURCE_MANAGER.SWITCH_PLAN package procedure or the ALTER SYSTEM statement.

The following SQL statement sets the top plan to mydb_plan, and activates the Resource Manager if it is not already active:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
```

An error message is returned if the specified plan does not exist in the data dictionary.

Automatic Enabling of the Resource Manager by Oracle Scheduler Windows

The Resource Manager automatically activates if an Oracle Scheduler window that specifies a resource plan opens. When the Scheduler window closes, the resource plan associated with the window is disabled, and the resource plan that was running before the Scheduler window opened is reenabled. (If no resource plan was enabled before the window opened, then the Resource Manager is disabled.) In an Oracle Real Application Clusters environment, a Scheduler window applies to all instances, so the window's resource plan is enabled on every instance.

Note that by default a set of automated maintenance tasks run during **maintenance windows**, which are predefined Scheduler windows that are members of the MAINTENANCE_WINDOW_GROUP window group and which specify the DEFAULT_MAINTENANCE_PLAN resource plan. Thus, the Resource Manager activates by default during maintenance windows. You can modify these maintenance windows to use a different resource plan, if desired.

Note: If you change the plan associated with maintenance windows, then ensure that you include the subplan ORA$AUTOTASK_SUB_PLAN and the consumer group ORA$DIAGNOSTICS in the new plan.

See Also:

- "Windows" on page 28-11
- Chapter 26, "Managing Automated Database Maintenance Tasks"

Disabling Plan Switches by Oracle Scheduler Windows

In some cases, the automatic change of Resource Manager plans at Scheduler window boundaries may be undesirable. For example, if you have an important task to finish, and if you set the Resource Manager plan to give your task priority, then you expect that the plan will remain the same until you change it. However, because a Scheduler window could activate after you have set your plan, the Resource Manager plan might change while your task is running.

To prevent this situation, you can set the RESOURCE_MANAGER_PLAN initialization parameter to the name of the plan that you want for the system and prepend "FORCE:" to the name, as shown in the following SQL statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'FORCE:mydb_plan';
```

Using the prefix FORCE: indicates that the current resource plan can be changed only when the database administrator changes the value of the RESOURCE_MANAGER_PLAN initialization parameter. This restriction can be lifted by rerunning the command without preceding the plan name with "FORCE:".

The DBMS_RESOURCE_MANAGER.SWITCH_PLAN package procedure has a similar capability.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information on DBMS_RESOURCE_MANAGER.SWITCH_PLAN.

Disabling the Resource Manager

To disable the Resource Manager, complete the following steps:

1. Issue the following SQL statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

2. Disassociate the Resource Manager from all Oracle Scheduler windows.

   To do so, for any Scheduler window that references a resource plan in its resource_plan attribute, use the DBMS_SCHEDULER.SET_ATTRIBUTE procedure to set resource_plan to the empty string (''). Qualify the window name with the SYS schema name if you are not logged in as user SYS. You can view Scheduler windows with the DBA_SCHEDULER_WINDOWS data dictionary view. See "Altering Windows" on page 29-57 and *Oracle Database PL/SQL Packages and Types Reference* for more information.

Note: By default, all maintenance windows reference the DEFAULT_MAINTENANCE_PLAN resource plan. To completely disable the Resource Manager, you must alter all maintenance windows to remove this plan. However, use caution, because resource consumption by automated maintenance tasks will no longer be regulated, which may adversely affect the performance of your other sessions. See Chapter 26, "Managing Automated Database Maintenance Tasks" for more information on maintenance windows.

Putting It All Together: Oracle Database Resource Manager Examples

This section provides some examples of resource plans. The following examples are presented:

- Multilevel Plan Example
- Examples of Using the Maximum Utilization Limit Attribute
- Example of Using Several Resource Allocation Methods
- Example of Managing Parallel Statements Using Directive Attributes
- An Oracle-Supplied Mixed Workload Plan

Multilevel Plan Example

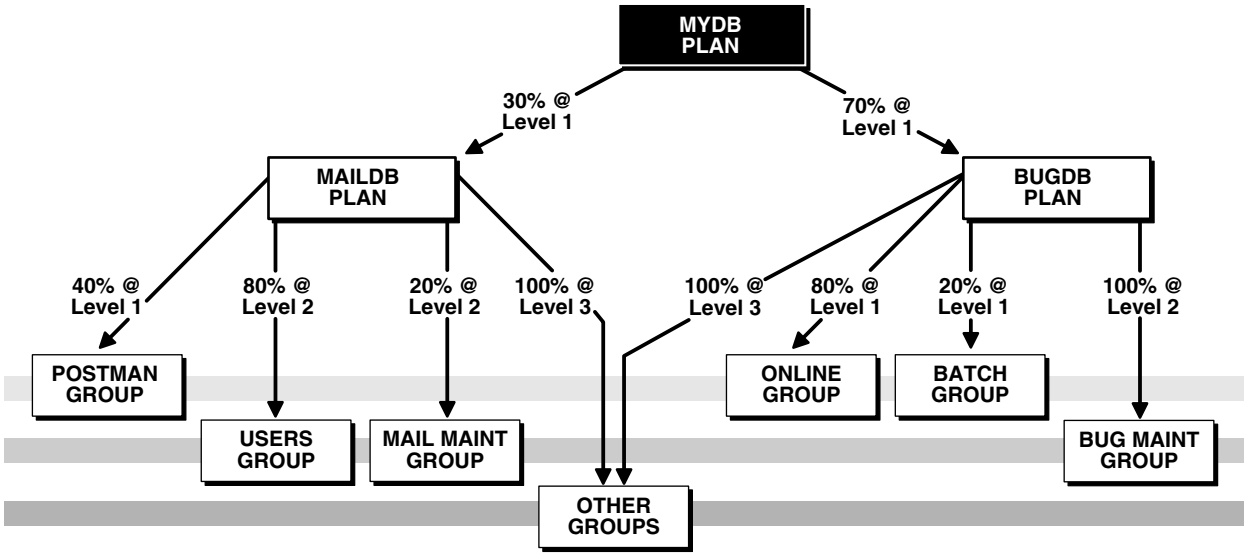
The following PL/SQL block creates a multilevel plan as illustrated in Figure 27–3 on page 27-39. Default resource allocation method settings are used for all plans and resource consumer groups.

```
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 => 'Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
```

```
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Batch_group',
   COMMENT => 'Resource consumer group/method for batch job bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Postman_group',
   COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Online_group',
   COMMENT => 'online bug users sessions at level 1', MGMT_P1 => 80, MGMT_P2=> 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Batch_group',
   COMMENT => 'batch bug users sessions at level 1', MGMT_P1 => 20, MGMT_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Maint_group',
   COMMENT => 'bug maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Postman_group',
   COMMENT => 'mail postman at level 1', MGMT_P1 => 40, MGMT_P2 => 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Users_group',
   COMMENT => 'mail users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 80);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Maint_group',
   COMMENT => 'mail maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', MGMT_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', MGMT_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_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 Plan Schema



In this plan schema, CPU resources are allocated as follows:

- Under `mydb_plan`, 30% of CPU is allocated to the `maildb_plan` subplan, and 70% is allocated to the `bugdb_plan` subplan. Both subplans are at level 1. Because `mydb_plan` itself has no levels below level 1, any resource allocations that are unused by either subplan at level 1 can be used by its sibling subplan. Thus, if `maildb_plan` uses only 20% of CPU, then 80% of CPU is available to `bugdb_plan`.
- `maildb_plan` and `bugdb_plan` define allocations at levels 1, 2, and 3. The levels in these subplans are independent of levels in their parent plan, `mydb_plan`. That is, all plans and subplans in a plan schema have their own level 1, level 2, level 3, and so on.
- Of the 30% of CPU allocated to `maildb_plan`, 40% of that amount (effectively 12% of total CPU) is allocated to `Postman_group` at level 1. Because `Postman_group` has no siblings at level 1, there is an implied 60% remaining at level 1. This 60% is then shared by `Users_group` and `Mail_Maint_group` at level 2, at 80% and 20%, respectively. In addition to this 60%, `Users_group` and `Mail_Maint_group` can also use any of the 40% not used by `Postman_group` at level 1.
- CPU resources not used by either `Users_group` or `Mail_Maint_group` at level 2 are allocated to `OTHER_GROUPS`, because in multilevel plans, unused resources are reallocated to consumer groups or subplans at the next lower level, not to siblings at the same level. Thus, if `Users_group` uses only 70% instead of 80%, the remaining 10% cannot be used by `Mail_Maint_group`. That 10% is available only to `OTHER_GROUPS` at level 3.
- The 70% of CPU allocated to the `bugdb_plan` subplan is allocated to its consumer groups in a similar fashion. If either `Online_group` or `Batch_group` does not use its full allocation, the remainder may be used by `Bug_Maint_group`. If `Bug_Maint_group` does not use all of that allocation, the remainder goes to `OTHER_GROUPS`.

Examples of Using the Maximum Utilization Limit Attribute

You can use the `MAX_UTILIZATION_LIMIT` directive attribute to limit the CPU utilization for applications. One of the most common scenarios in which this attribute can be used is for database consolidation.

During database consolidation, you may need to be able to do the following:

- Manage the performance impact that one application can have on another.

  One method of managing this performance impact is to create a consumer group for each application and allocate resources to each consumer group.

- Limit the utilization of each application.

  Typically, in addition to allocating a specific percentage of the CPU resources to each consumer group, you may need to limit the maximum CPU utilization for each group. This limit prevents a consumer group from using all of the CPU resources when all the other consumer groups are idle.

  In some cases, you may want all application users to experience consistent performance regardless of the workload from other applications. This can be achieved by specifying a maximum utilization limit for each consumer group in a resource plan.

The following examples demonstrate how to use the `MAX_UTILIZATION_LIMIT` resource plan directive attribute to:

- Restrict total database CPU utilization
- Quarantine runaway queries
- Limit CPU usage for applications
- Limit CPU utilization during maintenance windows

Example 1 - Restricting Overall Database CPU Utilization

In this example, regardless of database load, system workload from Oracle Database never exceeds 90% of CPU, leaving 10% of CPU for other applications sharing the server.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'MAXCAP_PLAN',
    COMMENT => 'Limit overall database CPU');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN              => 'MAXCAP_PLAN',
    GROUP_OR_SUBPLAN  => 'OTHER_GROUPS',
    COMMENT           => 'This group is mandatory',
    MAX_UTILIZATION_LIMIT => 90);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Because there is no plan directive other than the one for `OTHER_GROUPS`, all sessions are mapped to `OTHER_GROUPS`.

Example 2 - Quarantining Runaway Queries

In this example, runaway queries are switched to a consumer group with a maximum utilization limit of 20%, limiting the amount of resources that they can consume until you can intervene. A runaway query is characterized here as one that takes more than 10 minutes of CPU time. Assume that session mapping rules start all sessions in `START_GROUP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'START_GROUP',
     COMMENT        => 'Sessions start here');

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'QUARANTINE_GROUP',
     COMMENT        => 'Sessions switched here to quarantine them');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'Quarantine_plan',
    COMMENT => 'Quarantine runaway queries');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'START_GROUP',
    COMMENT               => 'Max CPU 10 minutes before switch',
    MGMT_P1               => 75,
    switch_group          => 'QUARANTINE_GROUP',
    switch_time           => 600);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'OTHER_GROUPS',
    COMMENT               => 'Mandatory',
    MGMT_P1               => 25);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'QUARANTINE_GROUP',
    COMMENT               => 'Limited CPU',
    MGMT_P2               => 100,
    MAX_UTILIZATION_LIMIT => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Caution: Although you could set the maximum utilization limit to zero for `QUARANTINE_GROUP`, thus completely quarantining runaway queries, it is recommended that you avoid doing this. If the runaway query is holding any resources—PGA memory, locks, and so on—required by any other session, then a zero allocation setting could lead to a deadlock.

Example 3 - LImiting CPU for Applications

In this example, assume that mapping rules map application sessions into one of four application groups. Each application group is allocated a maximum utilization limit of 30%. This limits CPU utilization of any one application to 30%. The sum of the `MAX_UTILIZATION_LIMIT` values exceeds 100%, which is permissible and acceptable in a situation where all applications are not active simultaneously.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

```
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Apps group 1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_GROUP',
     COMMENT        => 'Apps group 2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP3_GROUP',
     COMMENT        => 'Apps group 3');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP4_GROUP',
     COMMENT        => 'Apps group 4');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'apps_plan',
    COMMENT => 'Application consolidation');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'apps_plan',
    GROUP_OR_SUBPLAN      => 'APP1_GROUP',
    COMMENT               => 'Apps group 1',
    MAX_UTILIZATION_LIMIT => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'apps_plan',
    GROUP_OR_SUBPLAN      => 'APP2_GROUP',
    COMMENT               => 'Apps group 2',
    MAX_UTILIZATION_LIMIT => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'apps_plan',
    GROUP_OR_SUBPLAN      => 'APP3_GROUP',
    COMMENT               => 'Apps group 3',
    MAX_UTILIZATION_LIMIT => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'apps_plan',
    GROUP_OR_SUBPLAN      => 'APP4_GROUP',
    COMMENT               => 'Apps group 4',
    MAX_UTILIZATION_LIMIT => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'apps_plan',
    GROUP_OR_SUBPLAN      => 'OTHER_GROUPS',
    COMMENT               => 'Mandatory',
    MAX_UTILIZATION_LIMIT => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

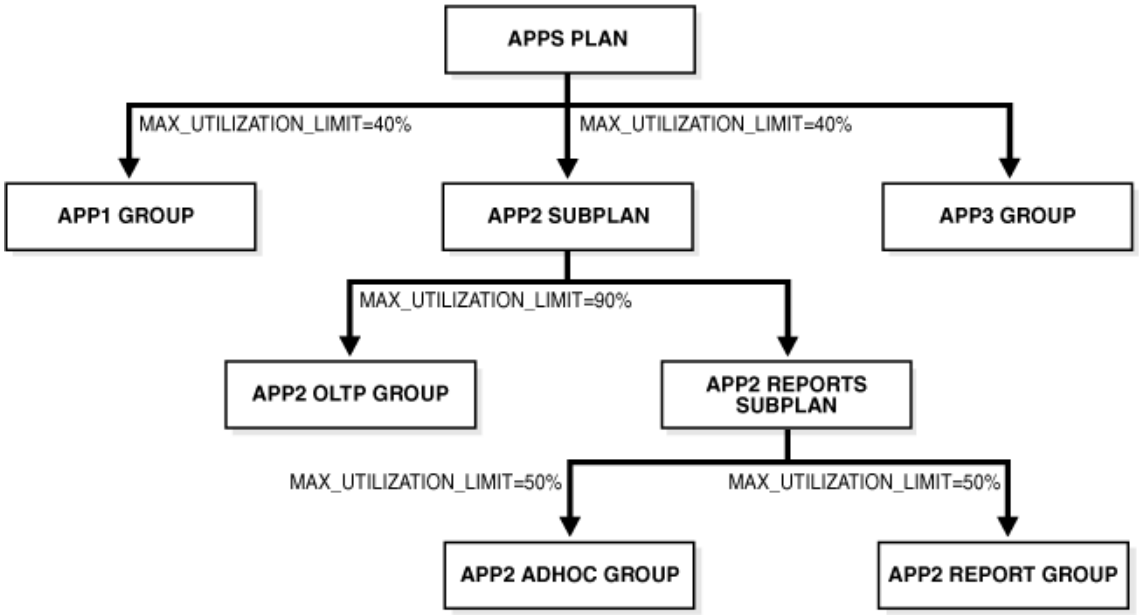
If all four application groups can fully use the CPU allocated to them (30% in this case), then the minimum CPU that is allocated to each application group is computed as a ratio of the application group's limit to the total of the limits of all application groups. In this example, all four application groups are allocated a maximum utilization limit of 30%. Therefore, when all four groups fully use their limits, the CPU allocation to each group is 30/(30+30+30+30) = 25%.

Example 4 - Specifying a Maximum Utilization Limit for Consumer Groups and Subplans

The following example describes how the maximum utilization limit is computed for scenarios, such as the one in Figure 27–4, where you set `MAX_UTILIZATION_LIMIT` for a

subplan and for consumer groups within the subplan. For simplicity, the requirement to include the OTHER_GROUPS consumer group is ignored, and resource plan directives are not shown, even though they are part of the plan.

Figure 27–4 Resource Plan with Maximum Utilization for Subplan and Consumer Groups



The following PL/SQL block creates the plan described in Figure 27–4.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Group for application #1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_OLTP_GROUP',
     COMMENT        => 'Group for OLTP activity in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_ADHOC_GROUP',
     COMMENT        => 'Group for ad-hoc queries in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_REPORT_GROUP',
     COMMENT        => 'Group for reports in application #2');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APPS_PLAN',
    COMMENT => 'Plan for managing 3 applications');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APP2_SUBPLAN',
    COMMENT => 'Subplan for managing application #2',
    SUB_PLAN => TRUE);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APP2_REPORTS_SUBPLAN',
    COMMENT => 'Subplan for managing reports in application #2',
    SUB_PLAN => TRUE);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APPS_PLAN',
```

```
    GROUP_OR_SUBPLAN      => 'APP1_GROUP',
    COMMENT               => 'Limit CPU for application #1 to 40%',
    MAX_UTILIZATION_LIMIT => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APPS_PLAN',
    GROUP_OR_SUBPLAN      => 'APP2_SUBPLAN',
    COMMENT               => 'Limit CPU for application #2 to 40%',
    MAX_UTILIZATION_LIMIT => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN      => 'APP2_OLTP_GROUP',
    COMMENT               => 'Limit CPU for OLTP to 90% of application #2',
    MAX_UTILIZATION_LIMIT => 90);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN      => 'APP2_REPORTS_SUBPLAN',
    COMMENT               => 'Subplan for ad-hoc and normal reports for
application #2');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN      => 'APP2_ADHOC_GROUP',
    COMMENT               => 'Limit CPU for ad-hoc queries to 50% of application
#2 reports',
    MAX_UTILIZATION_LIMIT => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN      => 'APP2_REPORT_GROUP',
    COMMENT               => 'Limit CPU for reports to 50% of application #2
reports',
    MAX_UTILIZATION_LIMIT => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                  => 'APPS_PLAN',
    GROUP_OR_SUBPLAN      => 'OTHER_GROUPS',
    COMMENT               => 'No directives for default users');

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

In this example, the maximum CPU utilization for the consumer group `APP1_GROUP` and subplan `APP2_SUBPLAN` is set to 40%. The limit for the consumer groups `APP2_ADHOC_GROUP` and `APP2_REPORT_GROUP` is set to 50%.

Because there is no limit specified for the subplan `APP2_REPORTS_SUBPLAN`, it inherits the limit of its parent subplan `APP2_SUBPLAN`, which is 40%. The absolute limit for the consumer group `APP2_REPORT_GROUP` is computed as 50% of its parent subplan, which is 50% of 40%, or 20%.

Similarly, because the consumer group `APP2_ADHOC_GROUP` is contained in the subplan `APP2_REPORTS_SUBPLAN`, its limit is computed as a percentage of its parent subplan. The maximum utilization limit for the consumer group `APP2_ADHOC_GROUP` is 50% of 40%, or 20%.

The maximum CPU utilization for the consumer group `APP2_OLTP_GROUP` is set to 90%. The parent subplan of `APP2_OLTP_GROUP`, `APP2_SUBPLAN`, has a limit of 40%. Therefore, the absolute limit for the group `APP2_OLTP_GROUP` is 90% of 40%, or 36%.

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) application. 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 Consumer Group Switching	Maximum Estimated Execution Time	Undo Pool
`oltp`	Level 1: 80%		Switch to group: `batch` Switch time: 3 secs		200K
`batch`	Level 2: 100%	Pool size: 5 Timeout: 600 secs	--	3600 secs	--
`OTHER_GROUPS`	Level 3: 100%	--	--		--

By assigning only 80% of the CPU to `oltp` at level 1, `batch` is guaranteed to get at least 20%, plus any of `oltp`'s unused CPU resources. `OTHER_GROUPS`, however, is not guaranteed any CPU resources. It gets CPU resources only if `batch` cannot consume all of its allocation. A similar-looking plan would give `oltp` 80% and `batch` 20%, both at level 1, and `OTHER_GROUPS` 100% at level 2. With this plan, `oltp`'s unused CPU allocation would be given to `OTHER_GROUPS`, not `batch`.

The following statements create the preceding plan, which is named `erp_plan`:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', MGMT_P1 => 80,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3, UNDO_POOL => 200);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', MGMT_P2 => 100,
  ACTIVE_SESS_POOL_P1 => 5, QUEUEING_P1 => 600, MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Example of Managing Parallel Statements Using Directive Attributes

A typical data warehousing environment consists of different types of users with varying resource requirements. Users with common processing needs are grouped into a consumer group. The consumer group `URGENT_GROUP` consists of users who run

reports that provide important information to top management. This group generates a large number of parallel queries. Users from the consumer group ETL_GROUP import data from source systems and perform extract, transform, and load (ETL) operations. The group OTHER_GROUPS contains users who execute ad-hoc queries. You must manage the requirements of these diverse groups of users while optimizing performance.

You can use the following directive attributes to manage and optimize the execution of parallel statements:

- MGMT_P*n*
- PARALLEL_TARGET_PERCENTAGE
- PARALLEL_QUEUE_TIMEOUT
- PARALLEL_DEGREE_LIMIT_P1

Table 27–3 describes the resource allocations of the plan DW_PLAN, which can be used to manage the needs of the data warehouse users. This plan contains the consumer groups URGENT_GROUP, ETL_GROUP, and OTHER_GROUPS. This example demonstrates the use of directive attributes in ensuring that one application or consumer group does not use all the available parallel servers.

Table 27–3 Resource Plan with Parallel Statement Directives

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	PARALLEL_ DEGREE_ LIMIT_P1	PARALLEL_ TARGET_ PERCENTAGE	PARALLEL_ QUEUE_ TIMEOUT
URGENT_GROUP	100%			12		
ETL_GROUP		100%		8	50%	
OTHER_GROUPS			100%	2	50%	360

In this example, the parameter PARALLEL_SERVERS_TARGET is set to 64, which means that the number of parallel servers available is 64. The total number of parallel servers that can be used for parallel statement execution before URGENT_GROUP sessions with PARALLEL_DEGREE_POLICY set to AUTO are added to the parallel statement queue is equal to 64. Because the PARALLEL_TARGET_PERCENTAGE attribute of ETL_GROUP and OTHER_GROUPS is 50%, the maximum number of parallel servers that can be used by these groups is 50% of 64, or 32 parallel servers each.

Note that parallel statements from a consumer group will only be queued if the PARALLEL_DEGREE_POLICY parameter is set to AUTO and the total number of active servers for the consumer group is higher than PARALLEL_SERVERS_TARGET. If PARALLEL_DEGREE_POLICY is set to MANUAL or LIMITED, then the statements are run provided there are enough parallel servers available. The parallel servers used by such a statement will count toward the total number of parallel servers used by the consumer group. However, the parallel statement will not be added to the parallel statement queue.

> **Tip:** For low-priority applications, it is a common practice to set low values for PARALLEL_DEGREE_LIMIT_P1 and PARALLEL_TARGET_PERCENTAGE.

Because URGENT_GROUPS has 100% of the allocation at level 1, its parallel statements will always be dequeued ahead of the other consumer groups from the parallel statement queue. Although URGENT_GROUPS has no PARALLEL_TARGET_PERCENTAGE directive attribute, a statement issued by a session in this group might still be queued if there are not enough available parallel servers to run it.

The degree of parallelism, represented by PARALLEL_DEGREE_LIMIT_P1, is set to 12 for URGENT_GROUP. Therefore, each parallel statement from URGENT_GROUP can use a maximum of 12 parallel servers. Similarly, each parallel statement from the ETL_GROUP can use a maximum of 8 parallel servers and each parallel statement from the OTHER_GROUPS can use 2 parallel servers.

Suppose, at a given time, the only parallel statements are from the ETL_GROUP, and they are using 26 out of the 32 parallel servers available to this group. Sessions from this consumer group have PARALLEL_DEGREE_POLICY set to AUTO. If another parallel statement with the PARALLEL_DEGREE_LIMIT_P1 attribute set to 8 is launched from ETL_GROUP, then this query cannot be run immediately because the available parallel servers in the ETL_GROUP is 32-26=6 parallel servers. The new parallel statement is queued until the number of parallel servers it requires is available in ETL_GROUP.

While the parallel statements in ETL_GROUP are being executed, suppose a parallel statement is launched from OTHER_GROUPS. This group still has 32 parallel servers available and so the parallel statement is executed.

The PARALLEL_QUEUE_TIMEOUT attribute for OTHER_GROUPS is set to 360. Therefore, any parallel statement from this group can remain in the parallel server queue for 360 seconds only. After this time, the parallel statement is removed from the queue and the error ORA-07454 is returned.

An Oracle-Supplied Mixed Workload Plan

Oracle Database includes a predefined resource plan, MIXED_WORKLOAD_PLAN, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle. MIXED_WORKLOAD_PLAN is defined as follows:

	CPU Resource Allocation				
Group or Subplan	**Level 1**	**Level 2**	**Level 3**	**Automatic Consumer Group Switching**	**Max Degree of Parallelism**
BATCH_GROUP			100%		
INTERACTIVE_GROUP		85%		Switch to group: BATCH_GROUP Switch time: 60 seconds Switch for call: TRUE	1
ORA$AUTOTASK_SUB_PLAN		5%			
ORA$DIAGNOSTICS		5%			
OTHER_GROUPS		5%			
SYS_GROUP	100%				

In this plan, because INTERACTIVE_GROUP is intended for short transactions, any call that consumes more than 60 seconds of CPU time is automatically switched to BATCH_GROUP, which is intended for longer batch operations.

You can use this predefined plan if it is appropriate for your environment. (You can modify the plan, or delete it if you do not intend to use it.) Note that there is nothing special about the names BATCH_GROUP and INTERACTIVE_GROUP. The names reflect only the intended purposes of the groups, and it is up to you to map application sessions to these groups and adjust CPU resource allocation percentages accordingly so that you achieve proper resource management for your interactive and batch applications. For example, to ensure that your interactive applications run under the INTERACTIVE_

GROUP consumer group, you must map your interactive applications' user sessions to this consumer group based on user name, service name, program name, module name, or action, as described in "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8. You must map your batch applications to the BATCH_GROUP in the same way. Finally, you must enable this plan as described in "Enabling Oracle Database Resource Manager and Switching Plans" on page 27-35.

See Table 27–4 on page 27-59 and Table 27–5 on page 27-59 for explanations of the other resource consumer groups and subplans in this plan.

Managing Multiple Database Instances on a Single Server

Oracle Database provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. This method is called instance caging. Instance caging and Oracle Database Resource Manager (the Resource Manager) work together to support desired levels of service across multiple instances.

This section contains:

- About Instance Caging
- Enabling Instance Caging

About Instance Caging

You might decide to run multiple Oracle database instances on a single multi-CPU server. A typical reason to do so would be server consolidation—using available hardware resources more efficiently. When running multiple instances on a single server, the instances compete for CPU. One resource-intensive database instance could significantly degrade the performance of the other instances. For example, on a 16-CPU system with four database instances, the operating system might be running one database instance on the majority of the CPUs during a period of heavy load for that instance. This could degrade performance in the other three instances. CPU allocation decisions such as this are made solely by the operating system; the user generally has no control over them.

A simple way to limit CPU consumption for each database instance is to use instance caging. **Instance caging** is a method that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously. In the previous example, if you use instance caging to limit the number of CPUs to four for each of the four instances, there is less likelihood that one instance can interfere with the others. When constrained to four CPUs, an instance might become CPU-bound. This is when the Resource Manager begins to do its work to allocate CPU among the various database sessions according to the resource plan that you set for the instance. Thus, instance caging and the Resource Manager together provide a simple, effective way to manage multiple instances on a single server.

There are two typical approaches to instance caging for a server:

- Over-provisioning—You would use this approach for non-critical databases such as development and test systems, or low-load non-critical production systems. In this approach, the sum of the CPU limits for each instance exceeds the actual number of CPUs on the system. For example, on a 4-CPU system with four database instances, you might limit each instance to three CPUs. When a server is over-provisioned in this way, the instances can impact each other's performance. However, instance caging limits the impact and helps provide somewhat predictable performance. However, if one of the instances has a period of high load, the CPUs are available to handle it. This is a reasonable approach for

non-critical systems, because one or more of the instances may frequently be idle or at a very low load.

- Partitioning—This approach is for critical production systems, where you want to prevent instances from interfering with each other. You allocate CPUs such that the sum of all allocations is equal to the number of CPUs on the server. For example, on a 16-server system, you might allocate 8 CPUs to the first instance, 4 CPUs to the second, and 2 each to the remaining two instances. By dedicating CPU resources to each database instance, the load on one instance cannot affect another's, and each instance performs predictably.

Using Instance Caging with Maximum Utilization Limit

If you enable instance caging and set a maximum utilization limit in your resource plan, then the absolute limit is computed as a percentage of the allocated CPU resources.

For example, if you enable instance caging and set the `CPU_COUNT` to 4, and a consumer group has a maximum utilization limit of 50%, then the consumer group can use a maximum of 50% of 4 CPUs, which is 2 CPUs.

Enabling Instance Caging

To enable instance caging, do the following for each instance on the server:

1. Enable the Resource Manager by assigning a resource plan, and ensure that the resource plan has CPU directives, using the `MGMT_P1` through `MGMT_P8` parameters.

   See "Enabling Oracle Database Resource Manager and Switching Plans" on page 27-35 for instructions.

2. Set the `cpu_count` initialization parameter.

   This is a dynamic parameter, and can be set with the following statement:

   ```
   ALTER SYSTEM SET CPU_COUNT = 4;
   ```

Maintaining Consumer Groups, Plans, and Directives

This section provides instructions for maintaining consumer groups, resource plans, and resource plan directives for Oracle Database Resource Manager (the Resource Manager). You perform maintenance tasks using the `DBMS_RESOURCE_MANAGER` PL/SQL package. The following topics are covered:

- Updating a Consumer Group
- Deleting a Consumer Group
- Updating a Plan
- Deleting a Plan
- Updating a Resource Plan Directive
- Deleting a Resource Plan Directive

See Also:

- Predefined Consumer Group Mapping Rules on page 27-60
- *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.

Updating a Consumer Group

You use the UPDATE_CONSUMER_GROUP procedure to update consumer group information. The pending area must be created first, and then submitted after the consumer group is updated. If you do not specify the arguments for the UPDATE_CONSUMER_GROUP procedure, then they remain unchanged in the data dictionary.

Deleting a Consumer Group

The DELETE_CONSUMER_GROUP procedure deletes the specified consumer group. The pending area must be created first, and then submitted after the consumer group is deleted. Upon deletion of a consumer group, all users having the deleted group as their initial consumer group are assigned the OTHER_GROUPS as their initial consumer group. All currently running sessions belonging to a deleted consumer group are assigned to a new consumer group, based on the consumer group mapping rules. If no consumer group is found for a session through mapping, the session is switched to the OTHER_GROUPS.

You cannot delete a consumer group if it is referenced by a resource plan directive.

Updating a Plan

You use the UPDATE_PLAN procedure to update plan information. The pending area must be created first, and then submitted after the plan is updated. If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary. The following PL/SQL block updates the COMMENT parameter.

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN(
   PLAN => 'DAYTIME',
   NEW_COMMENT => '50% more resources for OLTP applications');
END;
/
```

Deleting a Plan

The DELETE_PLAN procedure deletes the specified plan as well as all the plan directives associated with it. The pending area must be created first, and then submitted after the plan is deleted.

The following PL/SQL block deletes the great_bread plan and its directives.

```
BEGIN
  DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
END;
/
```

The resource consumer groups referenced by the deleted directives 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 and those subplans and resource consumer groups that are not marked by the database as mandatory. If DELETE_PLAN_CASCADE encounters an error, then it rolls back, leaving the plan unchanged.

You cannot delete the currently active plan.

Updating a Resource Plan Directive

Use the UPDATE_PLAN_DIRECTIVE procedure to update plan directives. The pending area must be created first, and then submitted after the resource plan directive is updated. If you do not specify an argument for the UPDATE_PLAN_DIRECTIVE procedure, then its corresponding parameter in the directive remains unchanged.

The following example adds a comment to a directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN             => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN => 'MYGROUP1',
         NEW_COMMENT      => 'Higher priority'
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

To clear (nullify) a comment, pass a null string (''). To clear (zero or nullify) any numeric directive parameter, set its new value to -1:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN                  => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN      => 'MYGROUP1',
         NEW_MAX_EST_EXEC_TIME => -1
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Deleting a Resource Plan Directive

To delete a resource plan directive, use the DELETE_PLAN_DIRECTIVE procedure. The pending area must be created first, and then submitted after the resource plan directive is deleted.

Viewing Database Resource Manager Configuration and Status

You can use several static data dictionary views and dynamic performance views to view the current configuration and status of Oracle Database Resource Manager (the Resource Manager). This section provides the following examples:

- Viewing Consumer Groups Granted to Users or Roles
- Viewing Plan Information
- Viewing Current Consumer Groups for Sessions
- Viewing the Currently Active Plans

> **See Also:** *Oracle Database Reference* for details on all static data dictionary views and dynamic performance 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 always starts in the SALES consumer group, can switch to the MARKETING group through a specific grant, and can switch to the DEFAULT_CONSUMER_GROUP (OTHER_GROUPS) and LOW_GROUP groups because they are granted to PUBLIC. SCOTT also can grant the SALES group but not the MARKETING group to other users.

```
SELECT * FROM dba_rsrc_consumer_group_privs;

GRANTEE            GRANTED_GROUP                  GRANT_OPTION INITIAL_GROUP
------------------ ------------------------------ ------------ -------------
PUBLIC             DEFAULT_CONSUMER_GROUP         YES          YES
PUBLIC             LOW_GROUP                      NO           NO
SCOTT              MARKETING                      NO           NO
SCOTT              SALES                          YES          YES
SYSTEM             SYS_GROUP                      NO           YES
```

SCOTT was granted the ability to switch to these groups using the DBMS_RESOURCE_MANAGER_PRIVS package.

Viewing Plan Information

This example uses the DBA_RSRC_PLANS view to display all of the resource plans defined in the database. All plans have a NULL status, meaning that they are not in the pending area.

> **Note:** Plans in the pending area have a status of PENDING. Plans in the pending area are being edited.

```
SELECT plan,status,comments FROM dba_rsrc_plans;

PLAN                         STATUS   COMMENTS
---------------------------- -------- ----------------------------------------
DSS_PLAN                              Example plan for DSS workloads that prio...
ETL_CRITICAL_PLAN                     Example plan for DSS workloads that prio...
MIXED_WORKLOAD_PLAN                   Example plan for a mixed workload that p...
ORA$AUTOTASK_SUB_PLAN                 Default sub-plan for automated maintenan...
DEFAULT_MAINTENANCE_PLAN              Default plan for maintenance windows tha...
DEFAULT_PLAN                          Default, basic, pre-defined plan that pr...
INTERNAL_QUIESCE                      Plan for quiescing the database.  This p...
INTERNAL_PLAN                         Internally-used plan for disabling the r...
ORA$AUTOTASK_HIGH_SUB_PLAN            Default sub-plan for high-priority, auto...
```

Viewing Current Consumer Groups for Sessions

You can use the V$SESSION view to display the consumer groups that are currently assigned to sessions.

```
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
   ...
```

Viewing the Currently Active Plans

This example sets mydb_plan, as created by the example shown earlier in "Multilevel Plan Example" on page 27-37, as the top level plan. It then queries the V$RSRC_PLAN view to display the currently active plans. The view displays the current top level plan and all of its descendent subplans.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;

System altered.

SELECT name, is_top_plan FROM v$rsrc_plan;

NAME             IS_TOP_PLAN
----------------------------
MYDB_PLAN        TRUE
MAILDB_PLAN      FALSE
BUGDB_PLAN       FALSE
```

Monitoring Oracle Database Resource Manager

Use the following dynamic performance views to help you monitor the results of your Oracle Database Resource Manager settings:

- V$RSRC_PLAN
- V$RSRC_CONSUMER_GROUP
- V$RSRC_SESSION_INFO
- V$RSRC_PLAN_HISTORY
- V$RSRC_CONS_GROUP_HISTORY
- V$RSRCMGRMETRIC
- V$RSRCMGRMETRIC_HISTORY

These views provide:

- Current status information
- History of resource plan activations
- Current and historical statistics on resource consumption and CPU waits by both resource consumer group and session

In addition, historical statistics are available through the DBA_HIST_RSRC_PLAN and DBA_HIST_RSRC_CONSUMER_GROUP views, which contain Automatic Workload Repository (AWR) snapshots of the V$RSRC_PLAN_HISTORY and V$RSRC_CONS_GROUP_HISTORY, respectively.

For assistance with tuning, the views V$RSRCMGRMETRIC and V$RSRCMGRMETRIC_HISTORY show how much time was spent waiting for CPU and how much CPU was consumed per minute for every consumer group for the past hour. These metrics can be viewed graphically with Enterprise Manager, on the Resource Manager Statistics page.

V$RSRC_PLAN This view displays the currently active resource plan and its subplans.

```
SELECT name, is_top_plan FROM v$rsrc_plan;

NAME                             IS_TOP_PLAN
-------------------------------- -----------
```

```
DEFAULT_PLAN                    TRUE
ORA$AUTOTASK_SUB_PLAN           FALSE
ORA$AUTOTASK_HIGH_SUB_PLAN      FALSE
```

The plan for which IS_TOP_PLAN is TRUE is the currently active (top) plan, and the other plans are subplans of either the top plan or of other subplans in the list.

This view also contains other information, including the following:

- The INSTANCE_CAGING column shows whether instance caging is enabled.
- The CPU_MANAGED column shows whether CPU is being managed.
- The PARALLEL_EXECUTION_MANAGED column shows whether parallel statement queuing is enabled.

See Also: *Oracle Database Reference*

V$RSRC_CONSUMER_GROUP Use the V$RSRC_CONSUMER_GROUP view to monitor resources consumed, including CPU, I/O, and parallel servers. It can also be used to monitor statistics related to CPU resource management, runaway query management, parallel statement queuing, and so on. All of the statistics are cumulative from the time when the plan was activated.

```
SELECT name, active_sessions, queue_length,
  consumed_cpu_time, cpu_waits, cpu_wait_time
  FROM v$rsrc_consumer_group;

NAME               ACTIVE_SESSIONS QUEUE_LENGTH CONSUMED_CPU_TIME  CPU_WAITS CPU_WAIT_TIME
------------------ --------------- ------------ ----------------- ---------- -------------
OLTP_ORDER_ENTRY                 1            0             29690        467          6709
OTHER_GROUPS                     0            0           5982366       4089         60425
SYS_GROUP                        1            0           2420704        914         19540
DSS_QUERIES                      4            2           4594660       3004         55700
```

In the preceding query results, the DSS_QUERIES consumer group has four sessions in its active session pool and two more sessions queued for activation.

A key measure in this view is CPU_WAIT_TIME. This indicates the total time that sessions in the consumer group waited for CPU because of resource management. Not included in this measure are waits due to latch or enqueue contention, I/O waits, and so on.

See Also: *Oracle Database Reference*

V$RSRC_SESSION_INFO Use this view to monitor the status of one or more sessions. The view shows how the session has been affected by the Resource Manager. It provides information such as:

- The consumer group that the session currently belongs to.
- The consumer group that the session originally belonged to.
- The session attribute that was used to map the session to the consumer group.
- Session state (RUNNING, WAIT_FOR_CPU, QUEUED, and so on).
- Current and cumulative statistics for metrics, such as CPU consumed, wait times, and queued time. Current statistics reflect statistics for the session since it joined its current consumer group. Cumulative statistics reflect statistics for the session in all consumer groups to which it has belonged since it was created.

```
SELECT se.sid sess_id, co.name consumer_group,
```

```
 se.state, se.consumed_cpu_time cpu_time, se.cpu_wait_time, se.queued_time
 FROM v$rsrc_session_info se, v$rsrc_consumer_group co
 WHERE se.current_consumer_group_id = co.id;

SESS_ID CONSUMER_GROUP     STATE     CPU_TIME CPU_WAIT_TIME QUEUED_TIME
------- ------------------ -------- --------- ------------- -----------
    113 OLTP_ORDER_ENTRY   WAITING     137947         28846           0
    135 OTHER_GROUPS       IDLE        785669         11126           0
    124 OTHER_GROUPS       WAITING      50401         14326           0
    114 SYS_GROUP          RUNNING        495             0           0
    102 SYS_GROUP          IDLE         88054            80           0
    147 DSS_QUERIES        WAITING     460910        512154           0
```

CPU_WAIT_TIME in this view has the same meaning as in the V$RSRC_CONSUMER_GROUP view, but applied to an individual session.

You can join this view with the V$SESSION view for more information about a session.

See Also: *Oracle Database Reference*

V$RSRC_PLAN_HISTORY This view shows when resource plans were enabled or disabled on the instance. Each resource plan activation or deactivation is assigned a sequence number. For each entry in the view, the V$RSRC_CONS_GROUP_HISTORY view has a corresponding entry for each consumer group in the plan that shows the cumulative statistics for the consumer group. The two views are joined by the SEQUENCE# column in each.

```
SELECT sequence# seq, name plan_name,
to_char(start_time, 'DD-MON-YY HH24:MM') start_time,
to_char(end_time, 'DD-MON-YY HH24:MM') end_time, window_name
FROM v$rsrc_plan_history;

 SEQ PLAN_NAME                  START_TIME      END_TIME        WINDOW_NAME
---- -------------------------- --------------- --------------- ----------------
   1                            29-MAY-07 23:05 29-MAY-07 23:05
   2 DEFAULT_MAINTENANCE_PLAN   29-MAY-07 23:05 30-MAY-07 02:05 TUESDAY_WINDOW
   3                            30-MAY-07 02:05 30-MAY-07 22:05
   4 DEFAULT_MAINTENANCE_PLAN   30-MAY-07 22:05 31-MAY-07 02:05 WEDNESDAY_WINDOW
   5                            31-MAY-07 02:05 31-MAY-07 22:05
   6 DEFAULT_MAINTENANCE_PLAN   31-MAY-07 22:05                 THURSDAY_WINDOW
```

A null value under PLAN_NAME indicates that no plan was active.

AWR snapshots of this view are stored in the DBA_HIST_RSRC_PLAN view.

See Also: *Oracle Database Reference*

V$RSRC_CONS_GROUP_HISTORY This view helps you understand how resources were shared among the consumer groups over time. The sequence# column corresponds to the column of the same name in the V$RSRC_PLAN_HISTORY view. Therefore, you can determine the plan that was active for each row of consumer group statistics.

```
SELECT sequence# seq, name, cpu_wait_time, cpu_waits,
consumed_cpu_time FROM v$rsrc_cons_group_history;

 SEQ NAME                      CPU_WAIT_TIME  CPU_WAITS CONSUMED_CPU_TIME
---- ------------------------- ------------- ---------- -----------------
   2 SYS_GROUP                         18133        691          33364431
   2 OTHER_GROUPS                      51252        825         181058333
```

```
  2 ORA$AUTOTASK_MEDIUM_GROUP            21          5           4019709
  2 ORA$AUTOTASK_URGENT_GROUP            35          1            198760
  2 ORA$AUTOTASK_STATS_GROUP              0          0                 0
  2 ORA$AUTOTASK_SPACE_GROUP              0          0                 0
  2 ORA$AUTOTASK_SQL_GROUP                0          0                 0
  2 ORA$AUTOTASK_HEALTH_GROUP             0          0                 0
  2 ORA$DIAGNOSTICS                       0          0           1072678
  4 SYS_GROUP                         40344         85          42519265
  4 OTHER_GROUPS                     123295       1040         371481422
  4 ORA$AUTOTASK_MEDIUM_GROUP             1          4           7433002
  4 ORA$AUTOTASK_URGENT_GROUP         22959        158          19964703
  4 ORA$AUTOTASK_STATS_GROUP              0          0                 0
    .
    .
  6 ORA$DIAGNOSTICS                       0          0                 0
```

AWR snapshots of this view are stored in the `DBA_HIST_RSRC_CONSUMER_GROUP` view. Use `DBA_HIST_RSRC_CONSUMER_GROUP` with `DBA_HIST_RSRC_PLAN` to determine the plan that was active for each historical set of consumer group statistics.

See Also:

- *Oracle Database Reference*
- *Oracle Database Performance Tuning Guide* for information about the AWR.

V$RSRCMGRMETRIC This view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute. It provides real-time metrics for each consumer group and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

Use this view to compare the maximum possible CPU utilization and average CPU utilization percentage for consumer groups with other consumer group settings such as CPU time used, time waiting for CPU, average number of sessions that are consuming CPU, and number of sessions that are waiting for CPU allocation. For example, you can view the amount of CPU resources a consumer group used and how long it waited for resource allocation. Or, you can view how many sessions from each consumer group are executed against the total number of active sessions.

To track CPU consumption in terms of CPU utilization, use the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a consumer group. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a consumer group can use. This limit is set using the `MAX_UTILIZATION_LIMIT` directive attribute.

```
SELECT consumer_group_name, cpu_utilization_limit,
avg_cpu_utilization FROM v$rsrcmgrmetric;
```

Use the `CPU_CONSUMED_TIME` and `CPU_TIME_WAIT` columns to track CPU consumption and throttling in milliseconds. The column `NUM_CPUS` represents the number of CPUs that Resource Manager is managing.

```
SELECT consumer_group_name, cpu_consumed_time,
cpu_wait_time, num_cpus FROM v$rsrcmgrmetric;
```

To track the CPU consumption and throttling in terms of number of sessions, use the `RUNNING_SESSIONS_LIMIT`, `AVG_RUNNING_SESSIONS`, and `AVG_WAITING_SESSIONS`

columns. RUNNING_SESSIONS_LIMIT lists the maximum number of sessions, from a particular consumer group, that can be running at any time. This limit is defined by the MAX_UTILIZATION_LIMIT directive attribute that you set either for the consumer group or for a subplan that contains the consumer group. For each consumer group, AVG_RUNNING_SESSIONS lists the average number of sessions that are consuming CPU and AVG_WAITING_SESSIONS lists the average number of sessions that are waiting for CPU.

```
SELECT sequence#, consumer_group_name, running_sessions_limit,
avg_running_sessions, avg_waiting_sessions FROM v$rsrcmgrmetric;
```

See Also: *Oracle Database Reference*

V$RSRCMGRMETRIC_HISTORY

The columns in the V$RSRCMGRMETRIC_HISTORY are the same view as V$RSRCMGRMETRIC. The only difference between these views is that V$RSRCMGRMETRIC contains metrics for the past one minute only, whereas V$RSRCMGRMETRIC_HISTORY contains metrics for the last 60 minutes.

See Also: *Oracle Database Reference*

Interacting with Operating-System Resource Control

Many operating systems provide tools for resource management. These tools often contain "workload manager" or "resource manager" in their names, and are intended to allow multiple applications to share the resources of a single server, using an administrator-defined policy. Examples are Hewlett Packard's Process Resource Manager or Solaris Containers, Zones, and Resource Pools.

Guidelines for Using Operating-System Resource Control

If you choose to use operating-system resource control with Oracle Database, then you must use it judiciously, according to the following guidelines:

- If you have multiple instances on a node, and you want to distribute resources among them, then each instance should be assigned to a dedicated operating-system resource manager group or managed entity. To run multiple instances in the managed entity, use instance caging to manage how the CPU resources within the managed entity should be distributed among the instances. When Oracle Database Resource Manager is managing CPU resources, it expects a fixed amount of CPU resources for the instance. Without instance caging, it expects the available CPU resources to be equal to the number of CPUs in the managed entity. With instance caging, it expects the available CPU resources to be equal to the value of the CPU_COUNT initialization parameter. If there are less CPU resources than expected, then the Oracle Database Resource Manager is not as effective at enforcing the resource allocations in the resource plan. See "Managing Multiple Database Instances on a Single Server" on page 27-48 for information about instance caging.
- The dedicated entity running all the instance's processes must run at one priority (or resource consumption) level.
- The CPU resources assigned to the dedicated entity cannot be changed more frequently than once every few minutes.

  If the operating-system resource manager is rapidly changing the CPU resources allocated to an Oracle instance, then the Oracle Database Resource Manager might

not manage CPU resources effectively. In particular, if the CPU resources allocated to the Oracle instance changes more frequently than every couple of minutes, then these changes might not be observed by Oracle because it only checks for such changes every couple of minutes. In these cases, Oracle Database Resource Manager can over-schedule processes if it concludes that more CPU resources are available than there actually are, and it can under-schedule processes if it concludes that less CPU resources are available than there actually are. If it over-schedules processes, then the `MAX_UTILIZATION_LIMIT` directives might be exceeded, and the CPU directives might not be accurately enforced. If it under-schedules processes, then the Oracle instance might not fully use the server's resources.

- Process priority management must not be enabled.
- Management of individual database processes at different priority levels (for example, using the `nice` command on UNIX platforms) is not supported. Severe consequences, including instance crashes, can result. Similar undesirable results are possible if operating-system resource control is permitted to manage the memory to which an Oracle Database instance is pinned.

Oracle Database Resource Manager Reference

The following sections provide reference information for Oracle Database Resource Manager (the Resource Manager):

- Predefined Resource Plans and Consumer Groups
- Predefined Consumer Group Mapping Rules
- Resource Manager Data Dictionary Views

Predefined Resource Plans and Consumer Groups

Table 27–4 lists the resource plans and Table 27–5 lists the resource consumer groups that are predefined in each Oracle database. You can verify these by querying the views `DBA_RSRC_PLANS` and `DBA_RSRC_CONSUMER_GROUPS`.

The following query displays the CPU allocations in the example plan `DSS_PLAN`:

```
SELECT group_or_subplan, mgmt_p1, mgmt_p2, mgmt_p3, mgmt_p4
   FROM dba_rsrc_plan_directives WHERE plan = 'DSS_PLAN';

GROUP_OR_SUBPLAN                  MGMT_P1    MGMT_P2    MGMT_P3    MGMT_P4
------------------------------ ---------- ---------- ---------- ----------
SYS_GROUP                              75          0          0          0
DSS_CRITICAL_GROUP                      0         75          0          0
DSS_GROUP                               0          0         75          0
ETL_GROUP                               0          0          0         45
BATCH_GROUP                             0          0          0         45
ORA$DIAGNOSTICS                         0          5          0          0
ORA$AUTOTASK_SUB_PLAN                   0          5          0          0
OTHER_GROUPS                            0          0          0         10
```

Table 27–4 Predefined Resource Plans

Resource Plan	Description
DEFAULT_MAINTENANCE_PLAN	Default plan for maintenance windows. See "About Resource Allocations for Automated Maintenance Tasks" on page 26-6 for details of this plan. Because maintenance windows are regular Oracle Scheduler windows, you can change the resource plan associated with them, if desired. If you do change a maintenance window resource plan, ensure that you include the subplan ORA$AUTOTASK_SUB_PLAN and the consumer group ORA$DIAGNOSTICS in the new plan.
DEFAULT_PLAN	Basic default plan that prioritizes SYS_GROUP operations and allocates minimal resources for automated maintenance and diagnostics operations.
DSS_PLAN	Example plan for a data warehouse that prioritizes critical DSS queries over non-critical DSS queries and ETL operations.
ETL_CRITICAL_PLAN	Example plan for a data warehouse that prioritizes ETL operations over DSS queries.
INTERNAL_PLAN	For disabling the resource manager. For internal use only.
INTERNAL_QUIESCE	For quiescing the database. This plan cannot be activated directly. To activate, use the QUIESCE command.
MIXED_WORKLOAD_PLAN	Example plan for a mixed workload that prioritizes interactive operations over batch operations. See "An Oracle-Supplied Mixed Workload Plan" on page 27-47 for details.

Table 27–5 Predefined Resource Consumer Groups

Resource Consumer Group	Description
BATCH_GROUP	Consumer group for batch operations. Referenced by the example plan MIXED_WORKLOAD_PLAN.
DSS_CRITICAL_GROUP	Consumer group for critical DSS queries. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
DSS_GROUP	Consumer group for non-critical DSS queries. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
ETL_GROUP	Consumer group for ETL jobs. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
INTERACTIVE_GROUP	Consumer group for interactive, OLTP operations. Referenced by the example plan MIXED_WORKLOAD_PLAN.
LOW_GROUP	Consumer group for low-priority sessions.
ORA$DIAGNOSTICS	Consumer group used by database processes that create diagnostic dumps when critical errors occur.
ORA$AUTOTASK_HEALTH_GROUP	Reserved for future use. Included in ORA$AUTOTASK_HIGH_SUB_PLAN.
ORA$AUTOTASK_MEDIUM_GROUP	Consumer group for medium-priority maintenance tasks.
ORA$AUTOTASK_SPACE_GROUP	Consumer group for Automatic Segment Advisor maintenance task. Included in ORA$AUTOTASK_HIGH_SUB_PLAN.
ORA$AUTOTASK_SQL_GROUP	Consumer group for Automatic SQL Tuning Advisor maintenance task. Included in ORA$AUTOTASK_HIGH_SUB_PLAN.
ORA$AUTOTASK_STATS_GROUP	Consumer group for optimizer statistics gathering maintenance task. Included in ORA$AUTOTASK_HIGH_SUB_PLAN.

Table 27–5 (Cont.) Predefined Resource Consumer Groups

Resource Consumer Group	Description
ORA$AUTOTASK_URGENT_GROUP	Consumer group for urgent maintenance tasks.
OTHER_GROUPS	Default consumer group for all sessions that do not have an explicit initial consumer group, are not mapped to a consumer group with session-to–consumer group mapping rules, or are mapped to a consumer group that is not in the currently active resource plan. OTHER_GROUPS must have a resource plan directive specified in every plan. It cannot be assigned explicitly to sessions through mapping rules.
SYS_GROUP	Consumer group for system administrators. It is the initial consumer group for all sessions created by user accounts SYS or SYSTEM. This initial consumer group can be overridden by session-to–consumer group mapping rules.

Predefined Consumer Group Mapping Rules

Table 27–6 summarizes the consumer group mapping rules that are predefined in Oracle Database. You can verify these rules by querying the view DBA_RSRC_GROUP_MAPPINGS. You can use the DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING procedure to modify or delete any of these mapping rules.

Table 27–6 Predefined Consumer Group Mapping Rules

Attribute	Value	Mapped Consumer Group	Notes
ORACLE_USER	SYS	SYS_GROUP	
ORACLE_USER	SYSTEM	SYS_GROUP	
ORACLE_FUNCTION	BACKUP	BATCH_GROUP	The session is running a backup operation with RMAN. The session is automatically switched to BATCH_GROUP when the operation begins.
ORACLE_FUNCTION	COPY	BATCH_GROUP	The session is running a copy operation with RMAN. The session is automatically switched to BATCH_GROUP when the operation begins.
ORACLE_FUNCTION	DATALOAD	ETL_GROUP	The session is performing a data load operation with Data Pump. The session is automatically switched to ETL_GROUP when the operation begins.

See Also: "Specifying Session-to–Consumer Group Mapping Rules" on page 27-8

Resource Manager Data Dictionary Views

Table 27–7 lists views that are associated with the Resource Manager.

Table 27–7 Resource Manager Data Dictionary Views

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 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	Lists all resource plans that exist in the database.
DBA_RSRC_GROUP_MAPPINGS	Lists all of the various mapping pairs for all of the session attributes.
DBA_RSRC_MAPPING_PRIORITY	Lists the current mapping priority of each attribute.
DBA_HIST_RSRC_PLAN	Displays historical information about resource plan activation. This view contains AWR snapshots of V$RSRC_PLAN_HISTORY.
DBA_HIST_RSRC_CONSUMER_GROUP	Displays historical statistical information about consumer groups. This view contains AWR snapshots of V$RSRC_CONS_GROUP_HISTORY.
DBA_USERS USERS_USERS	DBA view contains information about all users of the database. It contains the initial resource consumer group for each user. USER view contains information about the current user. It contains the current user's initial resource consumer group.
V$RSRC_CONS_GROUP_HISTORY	For each entry in the view V$RSRC_PLAN_HISTORY, contains an entry for each consumer group in the plan showing the cumulative statistics for the consumer group.
V$RSRC_CONSUMER_GROUP	Displays information about active resource consumer groups. This view can be used for tuning.
V$RSRCMGRMETRIC	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past minute.
V$RSRCMGRMETRIC_HISTORY	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past hour on a minute-by-minute basis. If a new resource plan is enabled, the history is cleared.
V$RSRC_PLAN	Displays the names of all currently active resource plans.
V$RSRC_PLAN_HISTORY	Shows when Resource Manager plans were enabled or disabled on the instance. It helps you understand how resources were shared among the consumer groups over time.
V$RSRC_SESSION_INFO	Displays Resource Manager statistics for each session. Shows how the session has been affected by the Resource Manager. Can be used for tuning.
V$SESSION	Lists session information for each current session. Specifically, lists the name of the resource consumer group of each current session.

See Also: *Oracle Database Reference* for detailed information about the contents of each of these views

28

Oracle Scheduler Concepts

In this chapter:

- Overview of Oracle Scheduler
- About Jobs and Supporting Scheduler Objects
- More About Jobs
- Scheduler Architecture
- Scheduler Support for Oracle Data Guard
- Oracle Scheduler and Editions

Overview of Oracle Scheduler

Oracle Database includes Oracle Scheduler, an enterprise job scheduler to help you simplify the scheduling of hundreds or even thousands of tasks. Oracle Scheduler (the Scheduler) is implemented by the procedures and functions in the `DBMS_SCHEDULER` PL/SQL package.

The Scheduler enables you to control when and where various computing tasks take place in the enterprise environment. The Scheduler helps you effectively manage and plan these tasks. By ensuring that many routine computing tasks occur without manual intervention, you can lower operating costs, implement more reliable routines, minimize human error, and shorten the time windows needed.

The Scheduler provides sophisticated, flexible enterprise scheduling functionality, which you can use to:

- Run **database program units**

  You can run program units, that is, PL/SQL anonymous blocks, PL/SQL stored procedures, and Java stored procedures on the local database or on one or more remote Oracle databases.

- Run external executables, (executables that are external to the database)

  You can run **external executables**, such as applications, shell scripts, and batch files, on the local system or on one or more remote systems. Remote systems do not require an Oracle Database installation; they require only a Scheduler agent. Scheduler agents are available for all platforms supported by Oracle Database and some additional platforms.

- Schedule job execution using the following methods:
  - Time-based scheduling

You can schedule a job to run at a particular date and time, either once or on a repeating basis. You can define complex repeat intervals, such as "every Monday and Thursday at 3:00 a.m. except on public holidays" or "the last Wednesday of each business quarter." See "Creating, Running, and Managing Jobs" on page 29-2 for more information.

– Event-based scheduling

  You can start jobs in response to system or business events. Your applications can detect events and then signal the Scheduler. Depending on the type of signal sent, the Scheduler starts a specific job. Examples of event-based scheduling include starting jobs when a file arrives on a system, when inventory falls below predetermined levels, or when a transaction fails. Beginning with Oracle Database 11*g* Release 2, a Scheduler object called a file watcher simplifies the task of configuring a job to start when a file arrives on a local or remote system. See "Using Events to Start Jobs" on page 29-30 for more information.

– Dependency scheduling

  You can set the Scheduler to run tasks based on the outcome of one or more previous tasks. You can define complex dependency chains that include branching and nested chains. See "Creating and Managing Job Chains" on page 29-41 for more information.

- Prioritize jobs based on business requirements.

  The Scheduler provides control over resource allocation among competing jobs, thus aligning job processing with your business needs. This is accomplished in the following ways:

  – Controlling Resources by Job Class

    You can group jobs that share common characteristics and behavior into larger entities called job classes. You can prioritize among the classes by controlling the resources allocated to each class. Therefore, you can ensure that your critical jobs have priority and enough resources to complete. For example, for a critical project to load a data warehouse, you can combine all the data warehousing jobs into one class and give it priority over other jobs by allocating a high percentage of the available resources to it. You can also assign relative priorities to the jobs within a job class.

  – Controlling Job Prioritization based on Schedules

    You can change job priorities based on a schedule. Because your definition of a critical job can change over time, the Scheduler also enables you to change the priority among your jobs over that time frame. For example, extract, transfer, and load (ETL) jobs used to load a data warehouse may be critical during non-peak hours but not during peak hours. Additionally, jobs that must run during the close of a business quarter may need to take priority over the ETL jobs. In these cases, you can change the priority among the job classes by changing the resources allocated to each class. See "Creating Job Classes" on page 29-54 and "Creating Windows" on page 29-57 for more information.

- Manage and monitor jobs

  You can manage and monitor the multiple states that jobs go through, from creation to completion. The Scheduler logs activity so that you can easily track information such as the status of the job and the last run time of the job by querying views using Enterprise Manager or SQL. These views provide valuable information about jobs and their execution that can help you schedule and better

manage your jobs. For example, a DBA can easily track all jobs that failed for a particular user. See "Scheduler Data Dictionary Views" on page 30-24.

When you create a multiple-destination job, a job that is defined at one database but that runs on multiple remote hosts, you can monitor the status of the job at each destination individually or the overall status of the parent job as a whole.

For advanced job monitoring, your applications can subscribe to job state change notifications that the Scheduler delivers in event queues. The Scheduler can also send e-mail notifications when a job changes state.

See "Monitoring and Managing the Scheduler" on page 30-10.

- Execute and manage jobs in a clustered environment

  A cluster is a set of database instances that cooperates to perform the same task. Oracle Real Application Clusters (Oracle RAC) provides scalability and reliability without any change to your applications. The Scheduler fully supports execution of jobs in such a clustered environment. To balance the load on your system and for better performance, you can also specify the database service where you want a job to run. See "Using the Scheduler in Real Application Clusters Environments" on page 28-25 for more information.

About Jobs and Supporting Scheduler Objects

To use the Scheduler, you create *Scheduler objects*. Schema objects define the what, when, and where for job scheduling. Scheduler objects enable a modular approach to managing tasks. One advantage of the modular approach is that objects can be reused when creating new tasks that are similar to existing tasks.

The principal Scheduler object is the job. A **job** defines the action to perform, the schedule for the action, and the location or locations where the action takes place. Most other scheduler objects are created to support jobs.

> **Note:** The Oracle Scheduler job replaces the `DBMS_JOB` package, which is still supported for backward compatibility. This chapter assumes that you are only using Scheduler jobs. If you are using both at once, or migrating from `DBMS_JOB` to Scheduler jobs, see Appendix A, "Support for DBMS_JOB in Release 11gR2."

The Scheduler objects include:

- Programs
- Schedules
- Jobs
- Destinations
- Chains
- File Watchers
- Credentials
- Job Classes
- Windows
- Groups

Each of these objects is described in detail later in this section.

Because Scheduler objects belong to schemas, you can grant object privileges on them. Some Scheduler objects, including job classes, windows, and window groups, are always created in the `SYS` schema, even if the user is not `SYS`. All other objects are created in the user's own schema or in a designated schema.

See Also:

- "Scheduler Privileges" on page 30-23

Programs

A program object (program) describes what is to be run by the Scheduler. A program includes:

- An action: For example, the name of a stored procedure, the name of an executable found in the operating system file system (an "external executable"), or the text of a PL/SQL anonymous block.
- A type: `STORED_PROCEDURE`, `PLSQL_BLOCK`, or `EXTERNAL`, where `EXTERNAL` indicates an external executable.
- Number of arguments: The number of arguments that the stored procedure or external executable accepts.

A program is a separate entity from a job. A job runs at a certain time or because a certain event occurred, and invokes a certain program. You can create jobs that point to existing program objects, which means that different jobs can use the same program and run the program at different times and with different settings. With the right privileges, different users can use the same program without having to redefine it. Therefore, you can create program libraries, where users can select from a list of existing programs.

If a stored procedure or external executable referenced by the program accepts arguments, you define these arguments in a separate step after creating the program. You can optionally define a default value for each argument.

See Also:

- "Creating Programs" on page 29-21
- "Jobs" on page 28-5 for an overview of jobs

Schedules

A schedule object (schedule) specifies when and how many times a job is run. Schedules can be shared by multiple jobs. For example, the end of a business quarter may be a common time frame for many jobs. Rather than defining an end-of-quarter schedule each time a new job is defined, job creators can point to a named schedule.

There are two types of schedules:

- time schedules

  With time schedules, you can schedule jobs to run immediately or at a later time. Time schedules include a start date and time, optional end date and time, and optional repeat interval.
- event schedules

With event schedules, you can specify that a job executes when a certain event occurs, such as inventory falling below a threshold or a file arriving on a system. For more information on events, see "Using Events to Start Jobs" on page 29-30.

See Also: "Creating Schedules" on page 29-25

Jobs

A job object (job) is a collection of metadata that describes a user-defined task. It defines what must be executed (the action), when (the one-time or recurring schedule or a triggering event), where (the destinations), and with what credentials. A job has an owner, which is the schema in which it is created.

You define where a job runs by specifying a one or more destinations. Destinations are also Scheduler objects and are described later in this section. If you do not specify a destination, it is assumed that the job runs on the local database.

You specify the job action in one of the following ways:

- By specifying as a job attribute the database program unit or external executable to be run. This is known as specifying the job action **inline**.
- By specifying as a job attribute the name of an existing program object (program), that specifies the database program unit or external executable to be run. The job owner must have the `EXECUTE` privilege on the program or the `EXECUTE ANY PROGRAM` system privilege.

A job that runs a database program unit is known as a **database job**. A job that runs an external executable is known as an **external job**.

You specify the job schedule in one of the following ways:

- By setting attributes of the job object to define start and end dates and a repeat interval, or to define an event that starts the job. This is known as specifying the schedule **inline**.
- By specifying as a job attribute the name of an existing schedule object (schedule), which defines start and end dates and a repeat interval, or defines an event.

You specify the job destinations in one of the following ways:

- By specifying as a job attribute a single named destination object. In this case, the job runs on one remote location.
- By specifying as a job attribute a named destination group, which is equivalent to a list of remote locations. In this case, the job runs on all remote locations.
- By not specifying a destination attribute, in which case the job runs locally. The job runs either of the following:
  - A database program unit on the local database (the database on which the job is created)
  - An external executable on the local host, depending on the job action type

You specify the job credentials in one of the following ways:

- By specifying as a job attribute a named credential object, which contains either of the following:
  - A database user name and password (for database jobs)
  - A host operating system user name and password (for external jobs)

  The job runs as the user named in the credential.

- By allowing the credential attribute of the job to remain `NULL`, in which case a local database job runs as the job owner, and a local external job runs with default credentials. (See Table 28–1 on page 28-17.) The job owner is the schema in which the job was created.

Jobs that run database program units at one or more remote locations are called **remote database jobs**. Jobs that run external executables at one or more remote locations are called **remote external jobs**.

After you create a job and enable it, the Scheduler automatically runs the job according to its schedule or when the specified event is detected. You can view the run status of job and its job log by querying data dictionary views. If a job runs on multiple destinations, you can query the status of the job at each destination.

See Also:

- "Destinations" on page 28-6
- "More About Jobs" on page 28-15
- "Creating Jobs" on page 29-2
- "Scheduler Data Dictionary Views" on page 30-24

Destinations

A destination object (destination) defines a location for running a job.

There are two types of destinations:

- External destination: Specifies a remote host name and IP address for running a remote external job.
- Database destination: Specifies a remote database instance for running a remote database job.

Jobs that run external executables (external jobs) must specify external destinations, and jobs that run database program units (database jobs) must specify database destinations.

If you specify a destination when you create a job, the job runs on that destination. If you do not specify a destination, the job runs locally, on the system on which it is created.

You can also create a destination group, which consists of a list of destinations, and reference this destination group when creating a job. In this case, the job runs on all destinations in the group.

Note: Destination groups can also include the keyword `LOCAL` as a group member, indicating that the job also runs on the local host or local database.

See Also: "Groups" on page 28-14

No object privileges are required to use a destination created by another user.

About Destinations and Scheduler Agents

The remote location specified in a destination object must have a Scheduler agent running, and the agent must be registered with the database creating the job. The

Scheduler agent enables the local Scheduler to communicate with the remote host, start and stop jobs there, and return remote job status to the local database. For complete details, see "Creating Destinations" on page 29-7.

External Destinations You cannot explicitly create external destinations. They are created in your local database when you register a Scheduler agent with that database. The name assigned to the external destination is the name of the agent. You can configure an agent name after you install it, or you can accept the default agent name, which is the first part of the host name (before the first dot separator). For example, if you install an agent on the host `dbhost1.us.example.com`, the agent name defaults to `DBHOST1`.

Database Destinations You create database destinations with the `DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION` procedure.

Note: If you have multiple database instances running on the local host, you can run jobs on the other instances by creating database destinations for those instances. Thus, "remote" database instances do not necessarily have to reside on remote hosts. The local host must be running a Scheduler agent to support running remote database jobs on these additional instances.

See Also:

- "Creating Destinations" on page 29-7
- "Installing and Configuring the Scheduler Agent on a Remote Host" on page 30-6

File Watchers

A file watcher object (file watcher) defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job. You create a file watcher and then create any number of event-based jobs or event schedules that reference the file watcher. When the file watcher detects the arrival of the designated file, it raises a file arrival event. The job started by the file arrival event can retrieve the event message to learn about the newly arrived file.

A file watcher can watch for a file on the local system (the same host computer running Oracle Database) or a remote system, provided that the remote system is running the Scheduler agent.

To use file watchers, the database Java virtual machine (JVM) component must be installed.

See "About File Watchers" on page 29-35 for more information.

See Also: "Creating File Watchers and File Watcher Jobs" on page 29-36

Credentials

A **credential** is a user name and password pair stored in a dedicated database object. A job uses a credential to authenticate itself with a database instance or the operating system so that it can run. You use credentials for:

- Remote database jobs—The credential contains a database user name and password. The stored procedure or PL/SQL block specified in the remote database job runs as this database user.
- External jobs (local or remote)—The credential contains a host operating system user name and password. The external executable of the job then runs with this user name and password.
- File watchers—The credential contains a host operating system user name and password. The job that processes the file arrival event uses this user name and password to access the arrived file.

You can query the `*_SCHEDULER_CREDENTIALS` views to see a list of credentials in the database. Credential passwords are stored obfuscated, and are not displayed in the `*_SCHEDULER_CREDENTIALS` views.

See Also: "Creating Credentials" on page 29-6

Chains

Chains are the means by which you can implement dependency scheduling, in which job starts depend on the outcomes of one or more previous jobs. A chain consists of multiple steps that are combined using dependency rules. The dependency rules define the conditions that can be used to start or stop a step or the chain itself. Conditions can include the success, failure, or completion- or exit-codes of previous steps. Logical expressions, such as AND/OR, can be used in the conditions. In a sense, a chain resembles a decision tree, with many possible paths for selecting which tasks run and when.

In its simplest form, a chain consists of two or more Scheduler program objects (programs) that are linked together for a single, combined objective. An example of a chain might be "run program A followed by program B, and then run program C only if programs A and B complete successfully, otherwise wait an hour and then run program D."

As an example, you might want to create a chain to combine the different programs necessary for a successful financial transaction, such as validating and approving a loan application, and then funding the loan.

A Scheduler job can point to a chain instead of pointing to a single program object. The job then serves to start the chain. This job is referred to as the **chain job**. Multiple chain jobs can point to the same chain, and more than one of these jobs can run simultaneously, thereby creating multiple instances of the same chain, each at a different point of progress in the chain.

Each position within a chain is referred to as a **step**. Typically, after an initial set of chain steps has started, the execution of successive steps depends on the completion of one or more previous steps. Each step can point to one of the following:

- A program object (program)

  The program can run a database program unit (such as a stored procedure or PL/SQL anonymous block) or an external executable.
- Another chain (a nested chain)

  Chains can be nested to any level.
- An event schedule, inline event, or file watcher

  After starting a step that points to an event schedule or that has an inline event specification, the step waits until the specified event is raised. Likewise, a step that

references a file watcher inline or that points to an event schedule that references a file watcher waits until the file arrival event is raised. For a file arrival event or any other type of event, when the event occurs, the step completes, and steps that are dependent on the event step can run. A common example of an event in a chain is a user intervention, such an approval or rejection.

Multiple steps in the chain can invoke the same program or nested chain.

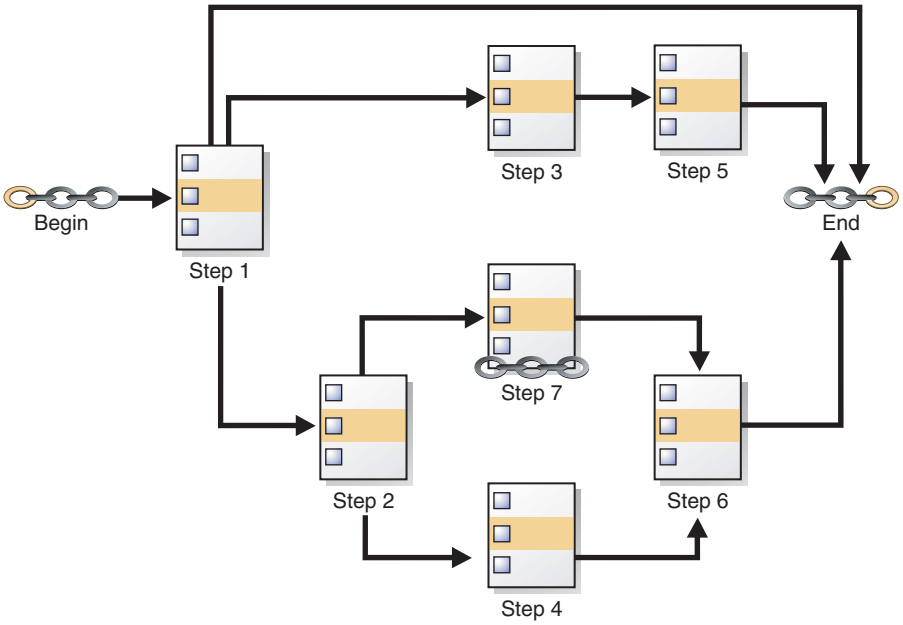
For each step, you can specify either a database destination or an external destination on which the step should run. If a destination is not specified, the step runs on the originating (local) database or the local host. Each step in a chain can run on a different destination.

Figure 28–1 on page 28-9 shows a chain with multiple branches. In this chain, rules could have been defined as follows:

- If Step 1 completes successfully, start Step 2.
- If Step 1 fails with error code 20100, start Step 3.
- If Step 1 fails with any other error code, end the chain.

Additional rules would govern the running of steps 4, 5, 6, and 7.

Figure 28–1 Chain with Multiple Branches



While a job pointing to a chain is running, the current state of all steps of the running chain can be monitored. For every step, the Scheduler creates a **step job** with the same job name and owner as the chain job. Each step job additionally has a step job subname to uniquely identify it. The step job subname is included as the `JOB_SUBNAME` column in the views `*_SCHEDULER_RUNNING_JOBS`, `*_SCHEDULER_JOB_LOG`, and `*_SCHEDULER_JOB_RUN_DETAILS`, and as the `STEP_JOB_SUBNAME` column in the `*_SCHEDULER_RUNNING_CHAINS` views.

See Also: "Creating and Managing Job Chains" on page 29-41

Job Classes

You typically create job classes only when you are in the role of Scheduler administrator.

Job classes provide a way to:

- Assign the same set of attribute values to member jobs

  Each job class specifies a set of attributes, such as logging level. When you assign a job to a job class, the job inherits those attributes. For example, you can specify the same policy for purging log entries for all payroll jobs.

- Set service affinity for member jobs

  You can set the `service` attribute of a job class to a desired database service name. This determines the instances in a Real Application Clusters environment that run the member jobs, and optionally, the system resources that are assigned to member jobs. See "Service Affinity when Using the Scheduler" on page 28-26 for more information.

- Set resource allocation for member jobs

  Job classes provide the link between the Database Resource Manager and the Scheduler, because each job class can specify a resource consumer group as an attribute. Member jobs then belong to the specified consumer group and are assigned resources according to settings in the current resource plan.

  Alternatively, you can leave the `resource_consumer_group` attribute `NULL` and set the `service` attribute of a job class to a desired database service name. That service can in turn be mapped to a resource consumer group. If both the `resource_consumer_group` and `service` attributes are set, and the designated service maps to a resource consumer group, the resource consumer group named in the `resource_consumer_group` attribute takes precedence.

  See Chapter 27, "Managing Resources with Oracle Database Resource Manager" for more information on mapping services to consumer groups.

- Group jobs for prioritization

  Within the same job class, you can assign priority values of 1-5 to individual jobs so that if two jobs in the class are scheduled to start at the same time, the one with the higher priority takes precedence. This ensures that you do not have a less important job preventing the timely completion of a more important one.

  If two jobs have the same assigned priority value, the job with the earlier start date takes precedence. If no priority is assigned to a job, its priority defaults to 3.

  > **Note:** Job priorities are used only to prioritize among jobs in the same class.
  >
  > There is no guarantee that a high priority job in class A will be started before a low priority job in class B, even if they share the same schedule. Prioritizing among jobs of different classes depends on the current resource plan and on the designated resource consumer group or service name of each job class.

When defining job classes, try to classify jobs by functionality. Consider dividing jobs into groups that access similar data, such as marketing, production, sales, finance, and human resources.

Some of the restrictions to keep in mind are:

- A job must be part of exactly one class. When you create a job, you can specify which class the job is part of. If you do not specify a class, the job automatically becomes part of the class `DEFAULT_JOB_CLASS`.
- Dropping a class while there are still jobs in that class results in an error. You can force a class to be dropped even if there are still jobs that are members of that class, but all jobs referring to that class are then automatically disabled and assigned to the class `DEFAULT_JOB_CLASS`. Jobs belonging to the dropped class that are already running continue to run under class settings determined at the start of the job.

See Also: "Creating Job Classes" on page 29-54

Windows

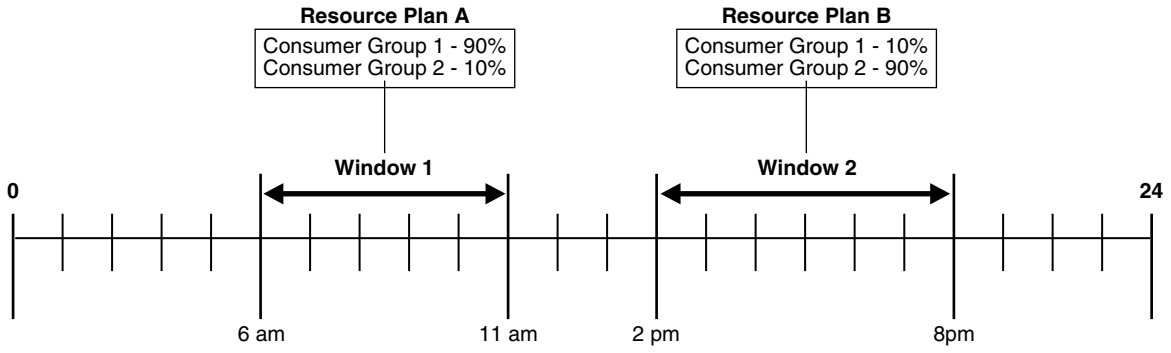
You typically create windows only when you are in the role of Scheduler administrator.

You create windows to automatically start jobs or to change resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time with a well-defined beginning and end, such as "from 12am-6am".

Windows work with job classes to control resource allocation. Each window specifies the resource plan to activate when the window **opens** (becomes active), and each job class specifies a resource consumer group or specifies a database service, which can map to a consumer group. A job that runs within a window, therefore, has resources allocated to it according to the consumer group of its job class and the resource plan of the window.

Figure 28–2 shows a workday that includes two windows. In this configuration, jobs belonging to the job class that links to `Consumer Group 1` get more resources in the morning than in the afternoon. The opposite is true for jobs in the job class that links to `Consumer Group 2`.

Figure 28–2 Windows help define the resources that are allocated to jobs



See Chapter 27, "Managing Resources with Oracle Database Resource Manager" for more information on resource plans and consumer groups.

You can assign a priority to each window. If windows overlap, the window with the highest priority is chosen over other windows with lower priorities. The Scheduler automatically opens and closes windows as window start times and end times come and go.

A job can name a window in its `schedule_name` attribute. The Scheduler then starts the job when the window opens. If a window is already open, and a new job is created that points to that window, the new job does not start until the next time the window opens.

Note: If necessary, you can temporarily block windows from switching the current resource plan. For more information, see "Enabling Oracle Database Resource Manager and Switching Plans" on page 27-35, or the discussion of the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure in *Oracle Database PL/SQL Packages and Types Reference*.

See Also: "Creating Windows" on page 29-57

Overlapping Windows

Although Oracle does not recommend it, windows can overlap.

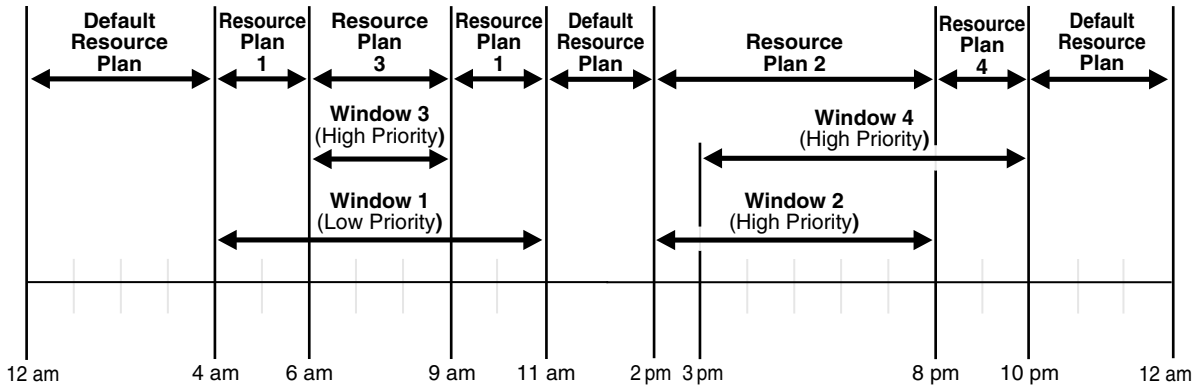
Because only one window can be active at one time, the following rules are used to determine which window is active when windows overlap:

- If windows of the same priority overlap, the window that is active will stay open. However, if the overlap is with a window of higher priority, the lower priority window will close and the window with the higher priority will open. Jobs currently running that had a schedule naming the low priority window may be stopped depending on the behavior you assigned when you created the job.
- If, at the end of a window, there are multiple windows defined, the window with the highest priority opens. If all windows have the same priority, the window that has the highest percentage of time remaining opens.
- An open window that is dropped automatically closes. At that point, the previous rule applies.

Whenever two windows overlap, an entry is written in the Scheduler log.

Examples of Overlapping Windows Figure 28–3 illustrates a typical example of how windows, resource plans, and priorities might be determined for a 24 hour schedule. In the following two examples, assume that Window1 has been associated with Resource Plan1, Window2 with Resource Plan2, and so on.

Figure 28–3 Windows and Resource Plans (Example 1)



In Figure 28–3, the following occurs:

- From 12AM to 4AM

  No windows are open, so a default resource plan is in effect.

- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows. Therefore, Resource Plan 1 is in effect.

- From 6AM to 9AM

  Window3 will open because it has a higher priority than Window1, so Resource Plan 3 is in effect.

- From 9AM to 11AM

  Even though Window1 was closed at 6AM because of a higher priority window opening, at 9AM, this higher priority window is closed and Window1 still has two hours remaining on its original schedule. It will be reopened for these remaining two hours and resource plan will be in effect.

- From 11AM to 2PM

  A default resource plan is in effect because no windows are open.

- From 2PM to 3PM

  Window2 will open so Resource Plan 2 is in effect.

- From 3PM to 8PM

  Window4 is of the same priority as Window2, so it does not interrupt Window2 and Resource Plan 2 is in effect.

- From 8PM to 10PM

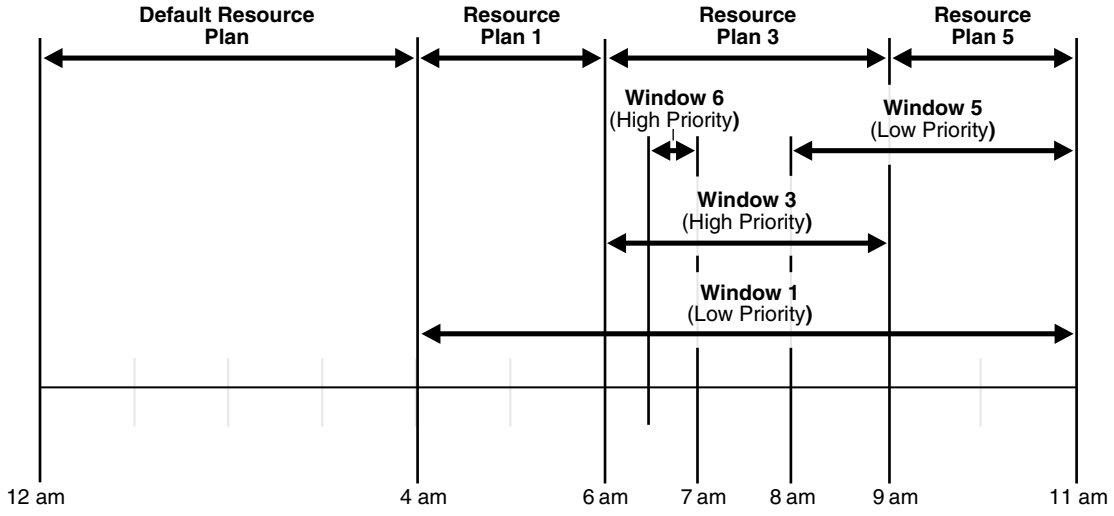
  Window4 will open so Resource Plan 4 is in effect.

- From 10PM to 12AM

  A default resource plan is in effect because no windows are open.

Figure 28–4 illustrates another example of how windows, resource plans, and priorities might be determined for a 24 hour schedule.

Figure 28–4 Windows and Resource Plans (Example 2)



In Figure 28–4, the following occurs:

- From 12AM to 4AM

  A default resource plan is in effect.

- From 4AM to 6AM

  Window1 has been assigned a low priority, but it opens because there are no high priority windows, so Resource Plan 1 is in effect.

- From 6AM to 9AM

  Window3 will open because it has a higher priority than Window1. Note that Window6 does not open because another high priority window is already in effect.

- From 9AM to 11AM

  At 9AM, Window5 or Window1 are the two possibilities. They both have low priorities, so the choice is made based on which has a greater percentage of its duration remaining. Window5 has a larger percentage of time remaining compared to the total duration than Window1. Even if Window1 were to extend to, say, 11:30AM, Window5 would have 2/3 * 100% of its duration remaining, while Window1 would have only 2.5/7 * 100%, which is smaller. Thus, Resource Plan 5 will be in effect.

Groups

A group designates a list of Scheduler objects. Instead of passing a list of objects as an argument to a `DBMS_SCHEDULER` package procedure, you create a group that has those objects as its members, and then pass the group name to the procedure.

There are three types of groups:

- Database destination groups: Members are database destinations, for running remote database jobs.
- External destination groups: Members are external destinations, for running remote external jobs.
- Window groups: Members are Scheduler windows.

All members of a group must be of the same type and each member must be unique.

You create a group with the `DBMS_SCHEDULER.CREATE_GROUP` procedure.

Destination Groups

When you want a job to run at multiple destinations, you create a database destination group or external destination group and assign it to the `destination_name` attribute of the job. Specifying a destination group as the `destination_name` attribute of a job is the only valid way to specify multiple destinations for the job.

Window Groups

You typically create window groups only when you are in the role of Scheduler administrator.

You can group windows for ease of use in scheduling jobs. If a job must run during multiple time periods throughout the day, week, and so on, you can create a window for each time period, and then add the windows to a window group. You can then set the `schedule_name` attribute of the job to the name of this window group, and the job executes during all the time periods specified by the windows in the window group.

For example, if you had a window called "Weekends" and a window called "Weeknights," you could add these two windows to a window group called "Downtime." The data warehousing staff could then create a job to run queries according to this Downtime window group—on weeknights and weekends—when the queries could be assigned a high percentage of available resources.

If a window in a window group is already open, and a new job is created that points to that window group, the job is not started until the next window in the window group opens.

See Also:

- "Creating Destination Groups for Multiple-Destination Jobs" on page 29-8
- "Creating Window Groups" on page 29-61
- "Windows" on page 28-11

More About Jobs

This section contains:

- Job Categories
- Job Instances
- Job Arguments
- How Programs, Jobs, and Schedules are Related

See Also:

- "Creating Jobs" on page 29-2
- "Viewing the Job Log" on page 29-65

Job Categories

The Scheduler supports the following types of jobs:

- Database Jobs
- External Jobs
- Multiple-Destination Jobs
- Chain Jobs
- Detached Jobs
- Lightweight Jobs

Database Jobs

Database jobs run Oracle Database program units, including PL/SQL anonymous blocks, PL/SQL stored procedures, and Java stored procedures. For a database job where the action is specified inline, `job_type` is set to `'PLSQL_BLOCK'` or `'STORED_PROCEDURE'`, and `job_action` contains either the text of a PL/SQL anonymous block or the name of a stored procedure. (If a program is a named program object rather than program action specified inline, the corresponding `program_type` and `program_action` must be set accordingly.)

Database jobs that run on the originating database—the database on which they were created—are known as **local database jobs**, or just jobs. Database jobs that run on a

target database other than the originating database are known as **remote database jobs**.

You can view run results for both local database and remote database jobs in the job log views on the originating database.

Local Database Jobs A local database job runs on the originating database, as the database user who is the job owner. The job owner is the name of the schema in which the job was created.

Remote Database Job The target database for a remote database job can be an Oracle database on a remote host or another database instance on the same host as the originating database. You identify a remote database job by specifying the name of an existing database destination object in the `destination_name` attribute of the job.

Creating a remote database job requires Oracle Database 11*g* Release 2 or later. However, the target database for the job can be any release of Oracle Database. No patch is required for the target database; you only need to install a Scheduler agent on the target database host (even if the target database host is the same as the originating database host) and register the agent with the originating database. The agent must be installed from Oracle Client 11*g* Release 2 or higher.

Remote database jobs must run as a user that is valid on the target database. You specify the required user name and password with a credential object that you assign to the remote database job.

See Also:

- "Credentials" on page 28-7
- "Creating Jobs" on page 29-2
- "Enabling and Disabling Databases for Remote Jobs" on page 30-5
- "Viewing the Job Log" on page 29-65

External Jobs

External jobs run external executables. An **external executable** is an operating system executable that runs outside the database, that is, external to the database. For an external job, `job_type` is specified as `'EXECUTABLE'`. (If using named programs, the corresponding `program_type` would be `'EXECUTABLE'`.) The `job_action` (or corresponding `program_action`) is the full operating system–dependent path of the desired external executable, excluding any command line arguments. An example might be `/usr/local/bin/perl` or `C:\perl\bin\perl`.

Note that a Windows batch file is not directly executable and must be run a command prompt (`cmd.exe`).

Like a database job, you can assign a schema when you create the external job. That schema then becomes the job owner. Although it is possible to create an external job in the `SYS` schema, Oracle recommends against this practice.

Both the `CREATE JOB` and `CREATE EXTERNAL JOB` privileges are required to create local or remote external jobs.

External executables must run as some operating system user. Thus, the Scheduler enables you to assign operating system credentials to any external job that you create. Like remote database jobs, you specify these credentials with a Scheduler credential object (a credential) and assign the credential to the external job.

There are two types of external jobs: local external jobs and remote external jobs. A **local external job** runs its external executable on the same computer as the database that schedules the job. A **remote external job** runs its executable on a remote host. The remote host does not need to have an Oracle database; you need only install and register a Scheduler agent.

> **Note:** On Windows, the host user that runs the external executable must be assigned the `Log on as a batch job` logon privilege.

The following sections provide more details on local external jobs and remote external jobs:

- About Local External Jobs
- About Remote External Jobs

> **See Also:**
>
> - "Credentials" on page 28-7
> - "Enabling and Disabling Databases for Remote Jobs" on page 30-5

About Local External Jobs A local external job runs its external executable on the same computer as the Oracle database that schedules the job. For such a job, the `destination_name` job attribute is `NULL`.

Local external jobs write stdout and stderr output to log files in the directory *ORACLE_HOME*/scheduler/log. You can retrieve the contents of these files with `DBMS_SCHEDULER.GET_FILE`.

You do not have to assign a credential to a local external job, although Oracle strongly recommends that you do so for improved security. If you do not assign a credential, the job runs with default credentials. Table 28–1 shows the default credentials for different platforms and different job owners.

Table 28–1 Default Credentials for Local External Jobs

Job in SYS Schema?	Platform	Default Credentials
Yes	All	User who installed Oracle Database.
No	UNIX and Linux	Values of the `run-user` and `run-group` attributes specified in the file *ORACLE_HOME*/rdbms/admin/externaljob.ora
No	Windows	User that the `OracleJobScheduler`*`SID`* Windows service runs as (either the Local System account or a named local or domain user). Note: You must manually enable and start this service. For improved security, Oracle recommends using a named user instead of the Local System account.

> **Note:** Default credentials are included for compatibility with previous releases of Oracle Database, and may be deprecated in a future release. It is, therefore, best to assign a credential to every local external job.

To disable the running of local external jobs that were not assigned credentials, remove the `run_user` attribute from the *ORACLE_HOME*/rdbms/admin/externaljob.ora file (UNIX and Linux) or stop the `OracleJobScheduler` service (Windows). These steps do not disable the running of local external jobs in the `SYS` schema.

See Also:

- Your operating system–specific documentation for any post-installation configuration steps to support local external jobs
- Example 29–8, "Creating a Local External Job and Retrieving stdout" on page 29-14

About Remote External Jobs A remote external job runs its external executable on a remote host. The remote host may or may not have Oracle Database installed. To enable remote external jobs to run on a specific remote host, you must install a Scheduler agent on the remote host and register it with the local database. The database communicates with the agent to start external executables and to retrieve execution results.

When creating a remote external job, you specify the name of an existing external destination object in the `destination_name` attribute of the job.

Remote external jobs write stdout and stderr output to log files in the directory *AGENT_HOME*/data/log. You can retrieve the contents of these files with `DBMS_SCHEDULER.GET_FILE`. Example 29–8, "Creating a Local External Job and Retrieving stdout" on page 29-14 illustrates how to retrieve stdout output. Although this example is for a local external job, the method is the same for remote external jobs.

See Also:

- "Credentials" on page 28-7
- "Enabling and Disabling Databases for Remote Jobs" on page 30-5

Multiple-Destination Jobs

A multiple-destination job is a job whose instances run on multiple target databases or hosts, but can be controlled and monitored from one central database. For DBAs or system administrators who must manage multiple databases or multiple hosts, a multiple-destination job can make administration considerably easier. With a multiple-destination job, you can:

- Specify several databases or hosts on which a job must run.
- Modify a job that is scheduled on multiple targets with a single operation.
- Stop jobs running on one or more remote targets.
- Determine the status (running, completed, failed, and so on) of the job instance at each of the remote targets.
- Determine the overall status of the collection of job instances.

A multiple-destination job can be viewed as a single entity for certain purposes and as a collection of independently running jobs for other purposes. When creating or altering the job metadata, the multiple-destination job looks like a single entity. However, when the job instances are running, they are better viewed as a collection of jobs that are nearly identical copies of each other. The job created at the source database is known as the **parent job**, and the job instances that run at the various destinations are known as **child jobs**.

You create a multiple-destination job by assigning a destination group to the `destination_name` attribute of the job. The job runs at all destinations in the group at its scheduled time, or upon the detection of a specified event. The local host can be included as one of the destinations on which the job runs.

For a job whose action is a database program unit, you must specify a database destination group in the `destination_name` attribute. The members of a database destination group include database destinations and the keyword `LOCAL`, which indicates the originating (local) database. For a job whose action is an external executable, you must specify an external destination group in the `destination_name` attribute. The members of an external destination group include external destinations and the keyword `LOCAL`, which indicates the local host.

Note: Database destinations do not necessarily have to reference remote databases; they can reference additional database instances running on the same host as the database that creates the job.

Multiple-Destination Jobs and Time Zones

Some job destinations might be in time zones that are different from that of the database on which the parent job is created (the *originating database*). In this case, the start time of the job is always based on the time zone of the originating database. So, if you create the parent job in London, England, specify a start time of 8:00 p.m., and specify destinations at Tokyo, Los Angeles, and New York, then all child jobs start at 8:00 p.m. London time. Start times at all destinations may not be exact, due to varying system loads, issues that require retries, and so on.

Event-Based Multiple-Destination Jobs

In the case of a multiple-destination job that is event-based, when the parent job detects the event at its host, it starts all the child jobs at all destinations. The child jobs themselves do not detect events at their respective hosts.

See Also:

- "Creating Multiple-Destination Jobs" on page 29-9
- "Monitoring Multiple Destination Jobs" on page 29-67
- "Destination Groups" on page 28-14
- "Using Events to Start Jobs" on page 29-30

Chain Jobs

The **chain** is the Scheduler mechanism that enables dependency-based scheduling. In its simplest form, it defines a group of program objects and the dependencies among them. A job can point to a chain instead of pointing to a single program object. The job then serves to start the chain. For a chain job, `job_type` is set to `'CHAIN'`.

See Also:

- "Chains" on page 28-8
- "Creating and Managing Job Chains" on page 29-41

Detached Jobs

You use a detached job to start a script or application that runs in a separate process, independently and asynchronously to the Scheduler. A detached job typically starts

another process and then exits. Upon exit (when the job action is completed) a detached job remains in the running state. The running state indicates that the asynchronous process that the job started is still active. When the asynchronous process finishes its work, it must connect to the database and call `DBMS_SCHEDULER.END_DETACHED_JOB_RUN`, which ends the job.

Detached jobs cannot be executed using `run_job` to manually trigger execution, when the `use_current_session` parameter set to `TRUE`.

A job is detached if it points to a program object (program) that has its `detached` attribute set to `TRUE` (a **detached program**).

You use a detached job under the following two circumstances:

- When it is impractical to wait for the launched asynchronous process to complete because would hold resources unnecessarily.

  An example is sending a request to an asynchronous Web service. It could take hours or days for the Web service to respond, and you do not want to hold a Scheduler job slave while waiting for the response. (See "Scheduler Architecture" on page 28-23 for information about job slaves.)

- When it is impossible to wait for the launched asynchronous process to complete because the process shuts down the database.

  An example would be using a Scheduler job to launch an RMAN script that shuts down the database, makes a cold backup, and then restarts the database. See Example 29–5 on page 29-11.

A detached job works as follows:

1. When it is time for the job to start, the job coordinator assigns a job slave to the job, and the job slave runs the program action defined in the detached program. The program action can be a PL/SQL block, a stored procedure, or an external executable.
2. The program action performs an immediate-return call of another script or executable, referred to here as Process A, and then exits. Because the work of the program action is complete, the job slave exits, but leaves the job in a running state.
3. Process A performs its processing. If it runs any DML against the database, it must commit its work. When processing is complete, Process A logs in to the database and calls `END_DETACHED_JOB_RUN`.
4. The detached job is logged as completed.

You can also call `STOP_JOB` to end a running detached job.

> **See Also:** "Creating Detached Jobs" on page 29-11 for an example of performing a cold backup of the database with a detached job

Lightweight Jobs

Use lightweight jobs when you have many short-duration jobs that run frequently. Under certain circumstances, using lightweight jobs can deliver a small performance gain.

Lightweight jobs have the following characteristics:

- Unlike regular jobs, they are not schema objects.
- They have significantly better create and drop times over regular jobs because they do not have the overhead of creating a schema object.

- They have lower average session create time than regular jobs.
- They have a small footprint on disk for job metadata and run-time data.

You designate a lightweight job by setting the `job_style` job attribute to `'LIGHTWEIGHT'`. The other job style is `'REGULAR'`, which is the default.

Like programs and schedules, regular jobs are schema objects. In releases before Oracle Database 11*g* Release 1, the only job style supported by the Scheduler was regular.

A regular job offers the maximum flexibility but does entail some overhead when it is created or dropped. The user has fine-grained control of the privileges on the job, and the job can have as its action a program or a stored procedure owned by another user.

If a relatively small number of jobs that run infrequently need to be created, then regular jobs are preferred over lightweight jobs.

A lightweight job must reference a program object (program) to specify a job action. The program must be already enabled when the lightweight job is created, and the program type must be either `'PLSQL_BLOCK'` or `'STORED_PROCEDURE'`. Because lightweight jobs are not schema objects, you cannot grant privileges on them. A lightweight job inherits privileges from its specified program. Thus, any user who has a certain set of privileges on the program has corresponding privileges on the lightweight job.

> **See Also:** "Creating Jobs" on page 29-2 and "Examples of Using the Scheduler" on page 30-17 for examples of creating lightweight jobs

Job Instances

A job instance represents a specific run of a job. Jobs that are scheduled to run only once have only one instance. Jobs that have a repeating schedule or that run each time an event occurs have multiple instances, each run of the job representing an instance. For example, a job that is scheduled to run only on Tuesday, Oct. 8th 2009 has one instance, a job that runs daily at noon for a week has seven instances, and a job that runs when a file arrives on a remote system has one instance for each file arrival event.

Multiple-destination jobs have one instance for each destination. If a multiple-destination job has a repeating schedule, then there is one instance for each run of the job at each destination.

When a job is created, only one entry is added to the Scheduler's job table to represent the job. Depending on the logging level set, each time the job runs, an entry is added to the job log. Therefore, if you create a job that has a repeating schedule, there is one entry in the job views (`*_SCHEDULER_JOBS`) and multiple entries in the job log. Each job instance log entry provides information about a particular run, such as the job completion status and the start and end time. Each run of the job is assigned a unique log id that appears in both the job log and job run details views (`*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`).

> **See Also:**
>
> - "Monitoring Jobs" on page 29-64
> - "Scheduler Data Dictionary Views" on page 30-24

Job Arguments

When a job references a program object (program), you can supply job arguments to override the default program argument values, or provide values for program arguments that have no default value. You can also provide argument values to an inline action (for example, a stored procedure) that the job specifies.

A job cannot be enabled until all required program argument values are defined, either as defaults in a referenced program object, or as job arguments.

A common example of a job is one that runs a set of nightly reports. If different departments require different reports, you can create a program for this task that can be shared among different users from different departments. The program action runs a reports script, and the program has one argument: the department number. Each user can then create a job that points to this program and can specify the department number as a job argument.

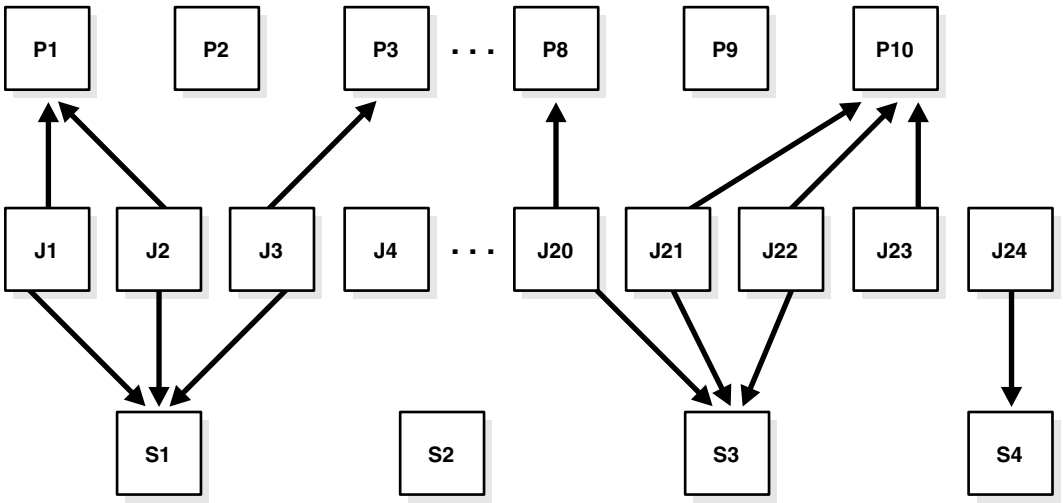
See Also:

- "Setting Job Arguments" on page 29-10
- "Defining Program Arguments" on page 29-22
- "Creating Jobs" on page 29-2

How Programs, Jobs, and Schedules are Related

To define what is executed and when, you assign relationships among programs, jobs, and schedules. Figure 28–5 illustrates examples of such relationships.

Figure 28–5 Relationships Among Programs, Jobs, and Schedules



To understand Figure 28–5, consider a situation where tables are being analyzed. In this example, program `P1` analyzes a table using the `DBMS_STATS` package. The program has an input parameter for the table name. Two jobs, `J1` and `J2`, both point to the same program, but each supplies a different table name. Additionally, schedule `S1` specifies a run time of 2:00 a.m. every day. The end result is that the two tables named in `J1` and `J2` are analyzed daily at 2:00 a.m.

Note that `J4` points to no other entity, so it is self-contained with all relevant information defined in the job itself. `P2`, `P9` and `S2` illustrate that you can leave a

program or schedule unassigned if you want. You can, for example, create a program that calculates a year-end inventory and temporarily leave it unassigned to any job.

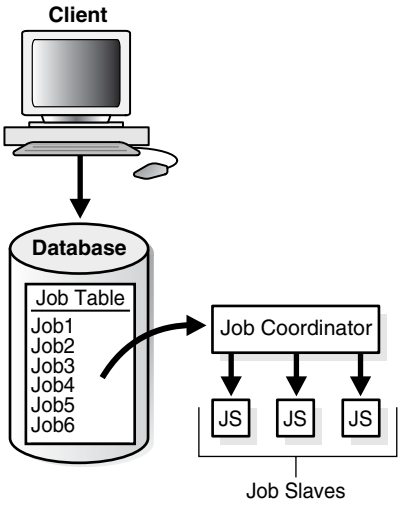
Scheduler Architecture

This section discusses the Scheduler architecture, and describes:

- The Job Table
- The Job Coordinator
- How Jobs Execute
- After Jobs Complete
- Using the Scheduler in Real Application Clusters Environments

Figure 28–6 illustrates how jobs are handled by the database.

Figure 28–6 Scheduler Components



The Job Table

The job table is a container for all the jobs, with one table for each database. The job table stores information for all jobs such as the owner name or the level of logging. You can find this information in the `*_SCHEDULER_JOBS` views.

Jobs are database objects, and therefore, can accumulate and take up too much space. To avoid this, job objects are automatically dropped by default after completion. This behavior is controlled by the `auto_drop` job attribute.

See "Scheduler Data Dictionary Views" on page 30-24 for the available job views and administration.

The Job Coordinator

The job coordinator, under the control of the database, controls and starts job slaves, making use of the information in the job table.

The job coordinator background process (`cjqNNN`) is automatically started and stopped on an as-needed basis. At database startup, the job coordinator is not started, but the

database does monitor whether there are any jobs to be executed, or windows to be opened in the near future. If so, it starts the coordinator.

As long as there are jobs or windows running, the coordinator continues to run. After there has been a certain period of Scheduler inactivity and there are no jobs or windows scheduled in the near future, the coordinator is automatically stopped.

When the database determines whether to start the job coordinator, it takes the service affinity of jobs into account. For example, if there is only one job scheduled in the near future and this job belongs to a job class that has service affinity for only two out of the four Oracle RAC instances, only the job coordinators for those two instances are started. See "Service Affinity when Using the Scheduler" on page 28-26 for more information.

Job Coordinator Actions

The job coordinator:

- Controls and spawns the job slaves
- Queries the job table
- Picks up jobs from the job table on a regular basis and places them in a memory cache. This improves performance by reducing trips to the disk
- Takes jobs from the memory cache and passes them to job slaves for execution
- Cleans up the job slave pool when slaves are no longer needed
- Goes to sleep when no jobs are scheduled
- Wakes up when a new job is about to be executed or a job was created using the `CREATE_JOB` procedure
- Upon database, startup after an abnormal database shutdown, recovers any jobs that were running.

You do not need to set the time that the job coordinator checks the job table; the system chooses the time frame automatically.

One job coordinator is used per instance. This is also the case in Oracle RAC environments.

> **See Also:** "Scheduler Data Dictionary Views" on page 30-24 for job coordinator administration and "Using the Scheduler in Real Application Clusters Environments" on page 28-25 for Oracle RAC information

Maximum Number of Scheduler Job Processes

The coordinator automatically determines how many job slaves to start based on CPU load and the number of outstanding jobs. The JOB_QUEUE_PROCESSES initialization parameter can be used to limit the number of job slaves that the Scheduler can start.

How Jobs Execute

Job slaves actually execute the jobs you submit. They are awakened by the job coordinator when it is time for a job to be executed. They gather metadata to run the job from the job table.

When a job is picked for processing, the job slave does the following:

1. Gathers all the metadata needed to run the job, for example, program arguments and privilege information.
2. Starts a database session as the owner of the job, starts a transaction, and then starts executing the job.
3. Once the job is complete, the slave commits and ends the transaction.
4. Closes the session.

After Jobs Complete

When a job is done, the slaves do the following:

- Reschedule the job if required.
- Update the state in the job table to reflect whether the job has completed or is scheduled to run again.
- Insert an entry into the job log table.
- Update the run count, and if necessary, failure and retry counts.
- Clean up.
- Look for new work (if none, they go to sleep).

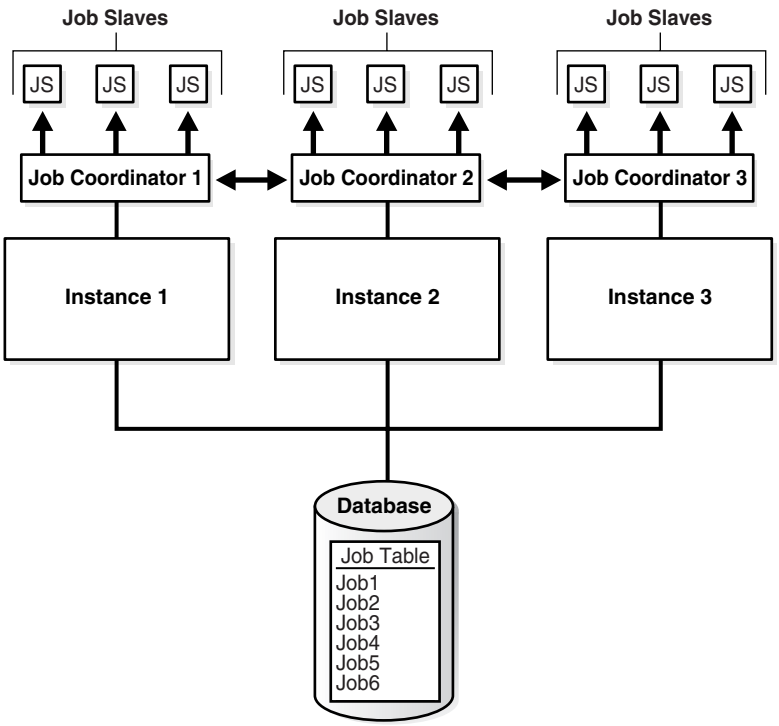
The Scheduler dynamically sizes the slave pool as required.

Using the Scheduler in Real Application Clusters Environments

In an Oracle Real Application Clusters (Oracle RAC) environment, the Scheduler uses one job table for each database and one job coordinator for each instance. The job coordinators communicate with each other to keep information current. The Scheduler attempts to balance the load of the jobs of a job class across all available instances when the job class has no service affinity, or across the instances assigned to a particular service when the job class does have service affinity.

Figure 28–7 illustrates a typical Oracle RAC architecture, with the job coordinator for each instance exchanging information with the others.

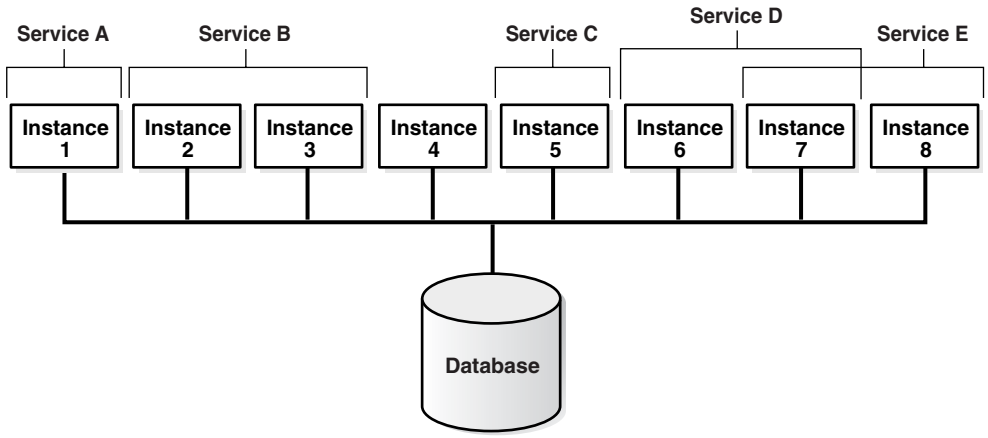
Figure 28–7 Oracle RAC Architecture and the Scheduler



Service Affinity when Using the Scheduler

The Scheduler enables you to specify the database service under which a job should be run (service affinity). This ensures better availability than instance affinity because it guarantees that other nodes can be dynamically assigned to the service if an instance goes down. Instance affinity does not have this capability, so, when an instance goes down, none of the jobs with an affinity to that instance can run until the instance comes back up. Figure 28–8 illustrates a typical example of how services and instances could be used.

Figure 28–8 Service Affinity and the Scheduler



In Figure 28–8, you could change the properties of the services and the Scheduler automatically recognizes the change.

Each job class can specify a database service. If a service is not specified, the job class belongs to an internal service that is guaranteed to be mapped to every running instance.

Scheduler Support for Oracle Data Guard

Beginning with Oracle Database 11*g* Release 1, the Scheduler can run jobs based on whether a database is a primary database or a logical standby in an Oracle Data Guard environment.

For a physical standby database, any changes made to Scheduler objects or any database changes made by Scheduler jobs on the primary database are applied to the physical standby like any other database changes.

For the primary database and logical standby databases, there is additional functionality that enables you to specify that a job can run only when the database is in the role of the primary database or a logical standby. You do this using the `DBMS_SCHEDULER.SET_ATTRIBUTE` procedure to set the `database_role` job attribute to one of two values: `'PRIMARY'` or `'LOGICAL STANDBY'`. (To run a job in both roles, you can make a copy of the job and set `database_role` to `'PRIMARY'` for one job and to `'LOGICAL STANDBY'` for the other). On switchover or failover, the Scheduler automatically switches to running jobs specific to the new role. DML is replicated to the job event log so that on failover, there is an available record of what ran successfully on the primary database until it failed.

See Also:

- "Examples of Setting Attributes" on page 30-18 for an example of setting the `database_role` attribute
- "Example of Creating a Job In an Oracle Data Guard Environment" on page 30-22
- *Oracle Data Guard Concepts and Administration*

Oracle Scheduler and Editions

Scheduler jobs always use the database default edition.

See Also: *Oracle Database Advanced Application Developer's Guide* for information on editions and edition-based redefinition

29

Scheduling Jobs with Oracle Scheduler

In this chapter:

- About Scheduler Objects and Their Naming
- Creating, Running, and Managing Jobs
- Creating and Managing Programs to Define Jobs
- Creating and Managing Schedules to Define Jobs
- Using Events to Start Jobs
- Creating and Managing Job Chains
- Prioritizing Jobs
- Monitoring Jobs

Note: This chapter describes how to use the `DBMS_SCHEDULER` package to work with Scheduler objects. You can accomplish the same tasks using Oracle Enterprise Manager.

See *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` information and the Oracle Enterprise Manager online help for information on Oracle Scheduler pages.

About Scheduler Objects and Their Naming

You operate Oracle Scheduler by creating and managing a set of Scheduler objects. Each Scheduler object is a complete database schema object of the form `[schema.]name`. Scheduler objects follow the naming rules for database objects exactly and share the SQL namespace with other database objects.

Follow SQL naming rules to name Scheduler objects in the `DBMS_SCHEDULER` package. By default, Scheduler object names are uppercase unless they are surrounded by double quotes. For example, when creating a job, `job_name => 'my_job'` is the same as `job_name => 'My_Job'` and `job_name => 'MY_JOB'`, but different from `job_name => '"my_job"'`. These naming rules are also followed in those cases where comma-delimited lists of Scheduler object names are used within the `DBMS_SCHEDULER` package.

See Also:

- *Oracle Database SQL Language Reference* for details regarding naming objects
- "About Jobs and Supporting Scheduler Objects" on page 28-3

Creating, Running, and Managing Jobs

A job is the combination of a schedule and a program, along with any additional arguments required by the program. This section introduces you to basic job tasks, and discusses the following topics:

- Job Tasks and Their Procedures
- Creating Jobs
- Altering Jobs
- Running Jobs
- Stopping Jobs
- Dropping Jobs
- Disabling Jobs
- Enabling Jobs
- Copying Jobs
- Viewing the Job Log
- Viewing stdout and stderr for External Jobs

See Also: "Jobs" on page 28-5 for an overview of jobs.

Job Tasks and Their Procedures

Table 29–1 illustrates common job tasks and their appropriate procedures and privileges:

Table 29–1 Job Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job	`CREATE_JOB` or `CREATE_JOBS`	`CREATE JOB` or `CREATE ANY JOB`
Alter a job	`SET_ATTRIBUTE` or `SET_JOB_ATTRIBUTES`	`ALTER` or `CREATE ANY JOB` or be the owner
Run a job	`RUN_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Copy a job	`COPY_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a job	`DROP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Stop a job	`STOP_JOB`	`ALTER` or `CREATE ANY JOB` or be the owner
Disable a job	`DISABLE`	`ALTER` or `CREATE ANY JOB` or be the owner
Enable a job	`ENABLE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Jobs

This section contains:

- Overview of Creating Jobs
- Specifying a Job Action and Job Schedule
- Specifying Job Credentials and Job Destinations

- Creating Multiple-Destination Jobs
- Setting Job Arguments
- Setting Additional Job Attributes
- Creating Detached Jobs
- Creating Multiple Jobs in a Single Transaction
- Techniques for External Jobs

Overview of Creating Jobs

You create one or more jobs using the DBMS_SCHEDULER.CREATE_JOB or DBMS_SCHEDULER.CREATE_JOBS procedures or Enterprise Manager. You use the CREATE_JOB procedure to create a single job. This procedure is overloaded to enable you to create different types of jobs that are based on different objects. You can create multiple jobs in a single transaction using the CREATE_JOBS procedure.

You must have the CREATE JOB privilege to create a job in your own schema, and the CREATE ANY JOB privilege to create a job in any schema except SYS.

For each job being created, you specify a job type, an action, and a schedule. You can also optionally specify a credential name, a destination or destination group name, a job class, and other attributes. As soon as you enable a job, it is automatically run by the Scheduler at its next scheduled date and time. By default, jobs are disabled when created and must be enabled with DBMS_SCHEDULER.ENABLE to run. You can also set the enabled argument of the CREATE_JOB procedure to TRUE, in which case the job is ready to be automatically run, according to its schedule, as soon as you create it.

Some job attributes cannot be set with CREATE_JOB, and instead must be set with DBMS_SCHEDULER.SET_ATTRIBUTE. For example, to set the logging_level attribute for a job, you must call SET_ATTRIBUTE after calling CREATE_JOB.

You can create a job in another schema by specifying schema.job_name. The creator of a job is, therefore, not necessarily the job owner. The job owner is the user in whose schema the job is created. The NLS environment of the job, when it runs, is the existing environment at the time the job was created.

Example 29–1 demonstrates creating a database job called update_sales, which calls a package procedure in the OPS schema that updates a sales summary table:

Example 29–1 Creating a Job

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'update_sales',
   job_type           =>  'STORED_PROCEDURE',
   job_action         =>  'OPS.SALES_PKG.UPDATE_SALES_SUMMARY',
   start_date         =>  '28-APR-08 07.00.00 PM Australia/Sydney',
   repeat_interval    =>  'FREQ=DAILY;INTERVAL=2', /* every other day */
   end_date           =>  '20-NOV-08 07.00.00 PM Australia/Sydney',
   auto_drop          =>   FALSE,
   job_class          =>  'batch_update_jobs',
   comments           =>  'My new job');
END;
/
```

Because no destination_name attribute is specified, the job runs on the originating (local) database. The job runs as the user who created the job.

The repeat_interval argument specifies that this job runs every other day until it reaches the end date and time. Another way to limit the number of times that a repeating job runs is to set its max_runs attribute to a positive number.

The job is disabled when it is created, by default. You must enable it with DBMS_SCHEDULER.ENABLE before the Scheduler will automatically run it.

Jobs are set to be automatically dropped by default after they complete. Setting the auto_drop attribute to FALSE causes the job to persist. Note that repeating jobs are not auto-dropped unless the job end date passes, the maximum number of runs (max_runs) is reached, or the maximum number of failures is reached (max_failures).

After a job is created, it can be queried using the *_SCHEDULER_JOBS views.

> **See Also:** "Specifying Job Credentials and Job Destinations" on page 29-5

Specifying a Job Action and Job Schedule

Because the CREATE_JOB procedure is overloaded, there are several different ways of using it. In addition to specifying the job action and job repeat interval as job attributes as shown in Example 29–1, known as specifying the job action and job schedule *inline*, you can create a job that points to a program object (program) to specify the job action, a schedule object (schedule) to specify the repeat interval, or both a program and schedule. This is discussed in the following sections:

- Creating Jobs Using a Named Program
- Creating Jobs Using a Named Schedule
- Creating Jobs Using Named Programs and Schedules

> **See Also:**
>
> - "Programs" on page 28-4
> - "Schedules" on page 28-4

Creating Jobs Using a Named Program You can create a job by pointing to a named program instead of inlining its action. To create a job using a named program, you specify the value for program_name in the CREATE_JOB procedure when creating the job and do not specify the values for job_type, job_action, and number_of_arguments.

To use an existing program when creating a job, the owner of the job must be the owner of the program or have EXECUTE privileges on it. The following PL/SQL block is an example of a CREATE_JOB procedure with a named program that creates a regular job called my_new_job1:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name          =>  'my_new_job1',
   program_name      =>  'my_saved_program',
   repeat_interval   =>  'FREQ=DAILY;BYHOUR=12',
   comments          =>  'Daily at noon');
END;
/
```

The following PL/SQL block creates a lightweight job. Lightweight jobs must reference a program, and the program type must be 'PLSQL_BLOCK' or 'STORED_PROCEDURE'. In addition, the program must be already enabled when you create the job.

```
BEGIN
```

```
  DBMS_SCHEDULER.CREATE_JOB (
   job_name         =>  'my_lightweight_job1',
   program_name     =>  'polling_prog_n2',
   repeat_interval  =>  'FREQ=SECONDLY;INTERVAL=10',
   end_date         =>  '30-APR-09 04.00.00 AM Australia/Sydney',
   job_style        => 'LIGHTWEIGHT',
   comments         => 'Job that polls device n2 every 10 seconds');
END;
/
```

Creating Jobs Using a Named Schedule You can also create a job by pointing to a named schedule instead of inlining its schedule. To create a job using a named schedule, you specify the value for `schedule_name` in the `CREATE_JOB` procedure when creating the job and do not specify the values for `start_date`, `repeat_interval`, and `end_date`.

You can use any named schedule to create a job because all schedules are created with access to `PUBLIC`. The following `CREATE_JOB` procedure has a named schedule and creates a regular job called `my_new_job2`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name                =>  'my_new_job2',
   job_type                =>  'PLSQL_BLOCK',
   job_action              =>  'BEGIN SALES_PKG.UPDATE_SALES_SUMMARY; END;',
   schedule_name           =>  'my_saved_schedule');
END;
/
```

Creating Jobs Using Named Programs and Schedules A job can also be created by pointing to both a named program and a named schedule. For example, the following `CREATE_JOB` procedure creates a regular job called `my_new_job3`, based on the existing program, `my_saved_program1`, and the existing schedule, `my_saved_schedule1`:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'my_new_job3',
   program_name        =>  'my_saved_program1',
   schedule_name       =>  'my_saved_schedule1');
END;
/
```

See Also:

- "Creating and Managing Programs to Define Jobs" on page 29-21
- "Creating and Managing Schedules to Define Jobs" on page 29-24
- "Using Events to Start Jobs" on page 29-30

Specifying Job Credentials and Job Destinations

For local external jobs, remote external jobs, and remote database jobs, you must specify the credentials under which the job runs. You do so by creating a credential object and assigning it to the `credential_name` job attribute.

For remote external jobs and remote database jobs, you specify the job destination by creating a destination object and assigning it to the `destination_name` job attribute. A job with a `NULL` `destination_name` attribute runs on the host where the job is created.

This section contains:

- Credential and Destination Tasks and Their Procedures

- Creating Credentials
- Creating Destinations
- Creating Destination Groups for Multiple-Destination Jobs
- Example: Creating a Remote Database Job

See Also:

- "Credentials" on page 28-7
- "Destinations" on page 28-6
- "Creating Multiple-Destination Jobs" on page 29-9

Credential and Destination Tasks and Their Procedures Table 29–2 illustrates credential and destination tasks and their procedures and privileges:

Table 29–2 Credential and Destination Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a credential	CREATE_CREDENTIAL	CREATE JOB or CREATE ANY JOB
Drop a credential	DROP_CREDENTIAL	CREATE ANY JOB or be the owner
Create an external destination	(none)	See "Creating Destinations" on page 29-7
Drop an external destination	DROP_AGENT_DESTINATION	MANAGE SCHEDULER
Create a database destination	CREATE_DATABASE_DESTINATION	CREATE JOB or CREATE ANY JOB
Drop a database destination	DROP_DATABASE_DESTINATION	CREATE ANY JOB or be the owner
Create a destination group	CREATE_GROUP	CREATE JOB or CREATE ANY JOB
Drop a destination group	DROP_GROUP	CREATE ANY JOB or be the owner
Add members to a destination group	ADD_GROUP_MEMBER	ALTER or CREATE ANY JOB or be the owner
Remove members from a destination group	REMOVE_GROUP_MEMBER	ALTER or CREATE ANY JOB or be the owner

Creating Credentials A **credential** is a user name and password pair stored in a dedicated database object. You assign a credential to a job so that it can authenticate with an Oracle database or the operating system before running.

To create a credential:

- Call the DBMS_SCHEDULER.CREATE_CREDENTIAL procedure.

You must have the CREATE JOB privilege to create a credential in your own schema, and the CREATE ANY JOB privilege to create a credential in any schema except SYS. A credential can be used only by a job whose owner has EXECUTE privileges on the credential or whose owner also owns the credential. Because a credential belongs to a schema like any other schema object, you use the GRANT SQL statement to grant privileges on a credential.

Example 29–2 Creating a Credential

```
BEGIN
 DBMS_SCHEDULER.CREATE_CREDENTIAL('DW_CREDENTIAL', 'dwuser', 'dW001515');
END;
/
```

```
GRANT EXECUTE ON DW_CREDENTIAL TO salesuser;
```

You can query the *_SCHEDULER_CREDENTIALS views to see a list of credentials in the database. Credential passwords are stored obfuscated and are not displayed in the *_SCHEDULER_CREDENTIALS views.

> **See Also:** "Credentials" on page 28-7 for more information about credentials

Creating Destinations A **destination** is a Scheduler object that defines a location for running a job. You designate the locations where a job runs by specifying either a single destination or a destination group in the destination_name attribute of the job. If you leave the destination_name attribute NULL, the job runs on the local host (the host where the job was created).

Use external destinations to specify locations where remote external jobs run. Use database destinations to specify locations where remote database jobs run.

You do not need object privileges to use a destination created by another user.

To create an external destination:

- Register a remote Scheduler agent with the database.

  See "Installing and Configuring the Scheduler Agent on a Remote Host" on page 30-6 for instructions.

> **Note:** There is no DBMS_SCHEDULER package procedure to create an external destination. You create an external destination implicitly by registering a remote agent.
>
> You can also register a local Scheduler agent if you have other database instances on the same host that are targets for remote jobs. This creates an external destination that references the local host.

The external destination name is automatically set to the agent name. To verify that the external destination was created, query the views DBA_SCHEDULER_EXTERNAL_DESTS or ALL_SCHEDULER_EXTERNAL_DESTS.

To create a database destination:

- Call the DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION procedure.

  You must specify the name of an external destination as a procedure argument. This designates the remote host that the database destination points to. You also specify a net service name or complete connect descriptor that identifies the database instance being connected to. If you specify a net service name, it must be resolved by the local tnsnames.ora file. If you do not specify a database instance, the remote Scheduler agent connects to its default database, which is specified in the agent configuration file.

To create a database destination, you must have the CREATE JOB system privilege. To create a database destination in a schema other than your own, you must have the CREATE ANY JOB privilege.

Example 29–3 Creating a Database Destination

The following example creates a database destination named DBHOST1_ORCLDW. For this example, assume the following:

- You installed a Scheduler agent on the remote host dbhost1.example.com, and you registered the agent with the local database.
- You did not modify the agent configuration file to set the agent name. Therefore the agent name and the external destination name default to DBHOST1.
- You used Net Configuration Assistant on the local host to create a connect descriptor in tnsnames.ora for the Oracle Database instance named orcldw, which resides on the remote host dbhost1.example.com. You assigned a net service name (alias) of ORCLDW to this connect descriptor.

```
BEGIN
 DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION (
  destination_name     => 'DBHOST1_ORCLDW',
  agent                => 'DBHOST1',
  tns_name             => 'ORCLDW',
  comments             => 'Instance named orcldw on host dbhost1.example.com');
END;
/
```

To verify that the database destination was created, query the views *_SCHEDULER_DB_DESTS.

See Also:

- "Destinations" on page 28-6 for more information about destinations
- "Jobs" on page 28-5 to learn about remote external jobs and remote database jobs

Creating Destination Groups for Multiple-Destination Jobs To create a job that runs on multiple destinations, you must create a destination group and assign that group to the destination_name attribute of the job. You can specify group members (destinations) when you create the group, or you can add group members at a later time.

To create a destination group:

- Call the DBMS_SCHEDULER.CREATE_GROUP procedure.

For remote external jobs you must specify a group of type 'EXTERNAL_DEST', and all group members must be external destinations. For remote database jobs, you must specify a group of type 'DB_DEST', and all members must be database destinations.

Members of destination groups have the following format:

```
[[schema.]credential@][schema.]destination
```

where:

- *credential* is the name of an existing credential.
- *destination* is the name of an existing database destination or external destination

The credential portion of a destination member is optional. If omitted, the job using this destination member uses its default credential.

You can include another group of the same type as a member of a destination group. Upon group creation, the Scheduler expands the included group into its members.

If you want the local host to be one of many destinations on which a job runs, you can include the keyword `LOCAL` as a group member for either type of destination group. `LOCAL` can be preceded by a credential only in an external destination group.

A group is owned by the user who creates it. You must have the `CREATE JOB` system privilege to create a group in your own schema, and the `CREATE ANY JOB` system privilege to create a group in another schema. You can grant object privileges on a group to other users by granting `SELECT` on the group.

Example 29–4 Creating a Database Destination Group

This example creates a database destination group. Because some members do not include a credential, a job using this destination group must have default credentials.

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP(
    GROUP_NAME    => 'all_dbs',
    GROUP_TYPE    => 'DB_DEST',
    MEMBER        => 'oltp_admin@orcl, orcldw1, LOCAL',
    COMMENTS      => 'All databases managed by me');
END;
/
```

The following code adds another member to the group.

```
BEGIN
  DBMS_SCHEDULER.ADD_GROUP_MEMBER(
    GROUP_NAME    => 'all_dbs',
    MEMBER        => 'dw_admin@orcldw2');
END;
/
```

See Also: "Groups" on page 28-14 for an overview of groups.

Example: Creating a Remote Database Job The following example creates a remote database job by specifying a database destination object in the `destination_name` object of the job. A credential must also be specified so the job can authenticate with the remote database. The example uses the credential created in Example 29–2 on page 29-6 and the database destination created in Example 29–3 on page 29-7.

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'SALES_SUMMARY1',
   job_type            =>  'STORED_PROCEDURE',
   job_action          =>  'SALES.SALES_REPORT1',
   start_date          =>  '15-JUL-09 11.00.00 PM Europe/Warsaw',
   repeat_interval     =>  'FREQ=DAILY',
   credential_name     =>  'DW_CREDENTIAL',
   destination_name    =>  'DBHOST1_ORCLDW');
END;
/
```

Creating Multiple-Destination Jobs

You can create a job that runs on multiple destinations, but that is managed from a single location. A typical reason to do this is to run a database maintenance job on all of the databases that you administer. Rather than create the job on each database, you create the job once and designate multiple destinations for the job. From the database where you created the job (the *local database*), you can monitor the state and results of all instances of the job at all locations.

To create a multiple-destination job:

- Call the `DBMS_SCHEDULER.CREATE_JOB` procedure and set the `destination_name` attribute of the job to the name of database destination group or external destination group.

  If not all destination group members include a credential prefix (the schema), assign a default credential to the job.

  To include the local host or local database as one of the destinations on which the job runs, ensure that the keyword `LOCAL` is one of the members of the destination group.

To obtain a list of destination groups, submit this query:

```
SELECT owner, group_name, group_type, number_of_members FROM all_scheduler_groups
  WHERE group_type = 'DB_DEST' or group_type = 'EXTERNAL_DEST';

OWNER           GROUP_NAME      GROUP_TYPE    NUMBER_OF_MEMBERS
--------------- --------------- ------------- -----------------
DBA1            ALL_DBS         DB_DEST                       4
DBA1            ALL_HOSTS       EXTERNAL_DEST                 4
```

The following example creates a multiple-destination database job, using the database destination group created in Example 29–4 on page 29-9. Because this is a system administration job, it uses a credential with system administrator privileges.

```
BEGIN
 DBMS_SCHEDULER.CREATE_CREDENTIAL('DBA_CREDENTIAL', 'dba1', 'sYs040533');
 DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'MAINT_SET1',
   job_type            =>  'STORED_PROCEDURE',
   job_action          =>  'MAINT_PROC1',
   start_date          =>  '15-JUL-09 11.00.00 PM Europe/Warsaw',
   repeat_interval     =>  'FREQ=DAILY',
   credential_name     =>  'DBA_CREDENTIAL',
   destination_name    =>  'ALL_DBS');
END;
/
```

See Also:

- "Multiple-Destination Jobs" on page 28-18
- "Monitoring Multiple Destination Jobs" on page 29-67
- "Groups" on page 28-14

Setting Job Arguments

After creating a job, you may need to set job arguments if:

- The inline job action is a stored procedure or other executable that requires arguments
- The job references a named program object and you want to override one or more default program arguments
- The job references a named program object and one or more of the program arguments were not assigned a default value

To set job arguments, use the `SET_JOB_ARGUMENT_VALUE` or `SET_JOB_ANYDATA_VALUE` procedures or Enterprise Manager. `SET_JOB_ANYDATA_VALUE` is used for complex data types that cannot be represented as a `VARCHAR2` string.

An example of a job that might need arguments is one that starts a reporting program that requires a start date and end date. The following code example sets the end date job argument, which is the second argument expected by the reporting program:

```
BEGIN
  DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE (
   job_name                => 'ops_reports',
   argument_position       => 2,
   argument_value          => '12-DEC-03');
END;
/
```

If you use this procedure on an argument whose value has already been set, it will be overwritten. You can set argument values using either the argument name or the argument position. To use argument name, the job must reference a named program object, and the argument must have been assigned a name in the program object. If a program is inlined, only setting by position is supported. Arguments are not supported for jobs of type 'PLSQL_BLOCK'.

To remove a value that has been set, use the RESET_JOB_ARGUMENT procedure. This procedure can be used for both regular and ANYDATA arguments.

SET_JOB_ARGUMENT_VALUE only supports arguments of SQL type. Therefore, argument values that are not of SQL type, such as booleans, are not supported as program or job arguments.

See Also: "Defining Program Arguments" on page 29-22

Setting Additional Job Attributes

After creating a job, you can set additional job attributes or change attribute values by using the SET_ATTRIBUTE or SET_JOB_ATTRIBUTES procedures. You can also set job attributes with Enterprise Manager. Although many job attributes can be set with the call to CREATE_JOB, some attributes, such as destination and credential_name, can be set only with SET_ATTRIBUTE or SET_JOB_ATTRIBUTES after the job is created.

Creating Detached Jobs

A detached job must point to a program object (program) that has its detached attribute set to TRUE.

Example 29–5 Creating a Detached Job That Performs a Cold Backup

This example for Linux and UNIX creates a nightly job that performs a cold backup of the database. It contains three steps.

Step 1—Create the Script That Invokes RMAN

Create a shell script that calls an RMAN script to perform a cold backup. The shell script is located in $ORACLE_HOME/scripts/coldbackup.sh. It must be executable by the user who installed Oracle Database (typically the user oracle).

```
#!/bin/sh

export ORACLE_HOME=/u01/app/oracle/product/11.2.0/db_1
export ORACLE_SID=orcl
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

$ORACLE_HOME/bin/rman TARGET / @$ORACLE_HOME/scripts/coldbackup.rman
  trace /u01/app/oracle/backup/coldbackup.out &
exit 0
```

Step 2—Create the RMAN Script

Create an RMAN script that performs the cold backup and then ends the job. The script is located in $ORACLE_HOME/scripts/coldbackup.rman.

```
run {
# Shut down database for backups and put into MOUNT mode
shutdown immediate
startup mount

# Perform full database backup
backup full format "/u01/app/oracle/backup/%d_FULL_%U" (database) ;

# Open database after backup
alter database open;

# Call notification routine to indicate job completed successfully
sql " BEGIN  DBMS_SCHEDULER.END_DETACHED_JOB_RUN(''sys.backup_job'', 0,
  null); END; ";
}
```

Step 3—Create the Job and Use a Detached Program

Submit the following PL/SQL block:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    program_name   => 'sys.backup_program',
    program_type   => 'executable',
    program_action => '?/scripts/coldbackup.sh',
    enabled        =>  TRUE);

  DBMS_SCHEDULER.SET_ATTRIBUTE('sys.backup_program', 'detached', TRUE);

  DBMS_SCHEDULER.CREATE_JOB(
    job_name        => 'sys.backup_job',
    program_name    => 'sys.backup_program',
    repeat_interval => 'FREQ=DAILY;BYHOUR=1;BYMINUTE=0');

  DBMS_SCHEDULER.ENABLE('sys.backup_job');
END;
/
```

See Also: "Detached Jobs" on page 28-19

Creating Multiple Jobs in a Single Transaction

If you must create many jobs, you may be able to reduce transaction overhead and experience a performance gain if you use the `CREATE_JOBS` procedure. Example 29–6 demonstrates how to use this procedure to create multiple jobs in a single transaction.

Example 29–6 Creating Multiple Jobs in a Single Transaction

```
DECLARE
 newjob sys.job_definition;
 newjobarr sys.job_definition_array;
BEGIN
 -- Create an array of JOB_DEFINITION object types
 newjobarr := sys.job_definition_array();

 -- Allocate sufficient space in the array
```

```
 newjobarr.extend(5);

 -- Add definitions for 5 jobs
 FOR i IN 1..5 LOOP
   -- Create a JOB_DEFINITION object type
   newjob := sys.job_definition(job_name => 'TESTJOB' || to_char(i),
                     job_style => 'REGULAR',
                     program_name => 'PROG1',
                     repeat_interval => 'FREQ=HOURLY',
                     start_date => systimestamp + interval '600' second,
                     max_runs => 2,
                     auto_drop => FALSE,
                     enabled => TRUE
                    );

   -- Add it to the array
   newjobarr(i) := newjob;
 END LOOP;

 -- Call CREATE_JOBS to create jobs in one transaction
 DBMS_SCHEDULER.CREATE_JOBS(newjobarr, 'TRANSACTIONAL');
END;
/

PL/SQL procedure successfully completed.

SELECT JOB_NAME FROM USER_SCHEDULER_JOBS;

JOB_NAME
------------------------------
TESTJOB1
TESTJOB2
TESTJOB3
TESTJOB4
TESTJOB5

5 rows selected.
```

See Also: "Lightweight Jobs" on page 28-20

Techniques for External Jobs

This section contains the following examples, which demonstrate some practical techniques for external jobs:

- Creating a Local External Job That Runs a DOS Command
- Creating a Local External Job and Retrieving stdout

Example 29–7 Creating a Local External Job That Runs a DOS Command

This example demonstrates how to create a local external job on Windows that runs a DOS built-in command (in this case, `mkdir`). The job runs `cmd.exe` with the `/c` option.

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
   job_name             => 'MKDIR_JOB',
   job_type             => 'EXECUTABLE',
   number_of_arguments  => 3,
   job_action           => '\windows\system32\cmd.exe',
   auto_drop            => FALSE,
   credential_name      => 'TESTCRED');
```

```
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',1,'/c');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',2,'mkdir');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('mkdir_job',3,'\temp\extjob_test_dir');
 DBMS_SCHEDULER.ENABLE('MKDIR_JOB');
END;
/
```

Example 29–8 Creating a Local External Job and Retrieving stdout

This example for Linux and UNIX shows how to create and run a local external job and then use the `GET_FILE` procedure to retrieve the job's stdout output. For local external jobs, stdout output is stored in a log file in *ORACLE_HOME*/scheduler/log. It is not necessary to supply this path to `GET_FILE`; you supply only the file name, which you generate by querying the log views for the job's external log ID and then appending "_stdout".

```
-- User scott must have CREATE JOB and CREATE EXTERNAL JOB privileges
grant create job, create external job to scott ;

connect scott/password
set serveroutput on

-- Create a credential for the job to use
exec dbms_scheduler.create_credential('my_cred','host_username','host_passwd')

-- Create a job that lists a directory. After running, the job is dropped.
begin
 DBMS_SCHEDULER.CREATE_JOB(
  job_name            => 'lsdir',
  job_type            => 'EXECUTABLE',
  job_action          => '/bin/ls',
  number_of_arguments => 1,
  enabled             => false,
  auto_drop           => true,
  credential_name     => 'my_cred');
 DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('lsdir',1,'/tmp');
 DBMS_SCHEDULER.ENABLE('lsdir');
end;
/

-- Wait a bit for the job to run, and then check the job results.
select job_name, status, error#, actual_start_date, additional_info
 from user_scheduler_job_run_details where job_name='LSDIR';

-- Now use the external log id from the additional_info column to
-- formulate the log file name and retrieve the output
declare
 my_clob clob;
 log_id varchar2(50);
begin
 select regexp_substr(additional_info,'job[_0-9]*') into log_id
   from user_scheduler_job_run_details where job_name='LSDIR';
 dbms_lob.createtemporary(my_clob, false);
 dbms_scheduler.get_file(
   source_file     => log_id ||'_stdout',
   credential_name => 'my_cred',
   file_contents   => my_clob,
   source_host     => null);
 dbms_output.put_line(my_clob);
```

```
end;
/
```

Note: For a remote external job, the method is the same, except that:

- You set the job's `destination_name` attribute.
- You designate a source host for the `GET_FILE` procedure.

`GET_FILE` automatically searches the correct host location for log files for both local and remote external jobs.

See Also:

- *Oracle Database Security Guide* for more information about external authentication
- "External Jobs" on page 28-16
- "Viewing stdout and stderr for External Jobs" on page 29-20
- "Stopping External Jobs" on page 29-17
- "Troubleshooting Remote Jobs" on page 30-16

Altering Jobs

You alter a job by modifying its attributes. You do so using the `SET_ATTRIBUTE`, `SET_ATTRIBUTE_NULL`, or `SET_JOB_ATTRIBUTES` package procedures or Enterprise Manager. See the `CREATE_JOB` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on job attributes.

All jobs can be altered, and, except for the job name, all job attributes can be changed. If there is a running instance of the job when the change is made, it is not affected by the call. The change is only seen in future runs of the job.

In general, you should not alter a job that was automatically created for you by the database. Jobs that were created by the database have the column `SYSTEM` set to `TRUE` in job views. The attributes of a job are available in the `*_SCHEDULER_JOBS` views.

It is valid for running jobs to alter their own job attributes. However, these changes do not take effect until the next scheduled run of the job.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE`, `SET_ATTRIBUTE_NULL`, and `SET_JOB_ATTRIBUTES` procedures.

The following example changes the `repeat_interval` of the job `update_sales` to once per week on Wednesday.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name         =>  'update_sales',
   attribute    =>  'repeat_interval',
   value        =>  'freq=weekly; byday=wed');
END;
/
```

Running Jobs

There are three ways in which a job can be run:

- According to the job schedule—In this case, provided that the job is enabled, the job is automatically picked up by the Scheduler job coordinator and run under the control of a job slave. The job runs as the user who is the job owner, or in the case of a local external job with a credential, as the user named in the credential. To find out whether the job succeeded, you must query the job views (`*_SCHEDULER_JOBS`) or the job log (`*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`). See "How Jobs Execute" on page 28-24 for more information job slaves and the Scheduler architecture.
- When an event occurs—Enabled event-based jobs start when a specified event is received on an event queue or when a file watcher raises a file arrival event. (See "Using Events to Start Jobs" on page 29-30.) Event-based jobs also run under the control of a job slave and run as the user who owns the job, or in the case of a local external job with a credential, as the user named in the credential. To find out whether the job succeeded, you must query the job views or the job log.
- By calling `DBMS_SCHEDULER.RUN_JOB`—You can use the `RUN_JOB` procedure to test a job or to run it outside of its specified schedule. You can run the job asynchronously, which is similar to the previous two methods of running a job, or synchronously, in which the job runs in the session that called `RUN_JOB`, and as the user logged in to that session. The `use_current_session` argument of `RUN_JOB` determines whether a job runs synchronously or asynchronously.

  `RUN_JOB` accepts a comma-delimited list of job names.

  The following example asynchronously runs two jobs:

```
BEGIN
  DBMS_SCHEDULER.RUN_JOB(
    JOB_NAME            => 'DSS.ETLJOB1, DSS.ETLJOB2',
    USE_CURRENT_SESSION => FALSE);
END;
/
```

> **Note:** It is not necessary to call `RUN_JOB` to run a job according to its schedule. Provided that job is enabled, the Scheduler runs it automatically.

Stopping Jobs

You stop one or more running jobs using the `STOP_JOB` procedure or Enterprise Manager. `STOP_JOB` accepts a comma-delimited list of jobs, job classes, and job destination IDs. A **job destination ID** is a number, assigned by the Scheduler, that represents a unique combination of a job, a credential, and a destination. It serves as a convenient method for identifying a particular child job of a multiple-destination job and for stopping just that child. You obtain the job destination ID for a child job from the `*_SCHEDULER_JOB_DESTS` views.

If a job class is supplied, all running jobs in the job class are stopped. For example, the following statement stops job `job1`, all jobs in the job class `dw_jobs`, and two child jobs of a multiple-destination job:

```
BEGIN
  DBMS_SCHEDULER.STOP_JOB('job1, sys.dw_jobs, 984, 1223');
END;
/
```

All instances of the designated jobs are stopped. After stopping a job, the state of a one-time job is set to `STOPPED`, and the state of a repeating job is set to `SCHEDULED`

(because the next run of the job is scheduled). In addition, an entry is made in the job log with `OPERATION` set to `'STOPPED'`, and `ADDITIONAL_INFO` set to `'REASON="Stop job called by user: username"'`.

By default, the Scheduler tries to gracefully stop a job using an interrupt mechanism. This method gives control back to the slave process, which can collect statistics of the job run. If the `force` option is set to `TRUE`, the job is abruptly terminated and certain run-time statistics might not be available for the job run.

Stopping a job that is running a chain automatically stops all running steps (by calling `STOP_JOB` with the `force` option set to `TRUE` on each step).

You can use the `commit_semantics` argument of `STOP_JOB` to control the outcome if multiple jobs are specified and errors occur when trying to stop one or more jobs. If you set this argument to `ABSORB_ERRORS`, the procedure may be able to continue after encountering an error and attempt to stop the remaining jobs. If the procedure indicates that errors occurred, you can query the view `SCHEDULER_BATCH_ERRORS` to determine the nature of the errors. See "Dropping Jobs" on page 29-17 for a more detailed discussion of commit semantics.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `STOP_JOB` procedure.

Caution: When a job is stopped, only the current transaction is rolled back. This can cause data inconsistency.

Stopping External Jobs

The Scheduler offers implementors of external jobs a mechanism to gracefully clean up after their external jobs when `STOP_JOB` is called with `force` set to `FALSE`. The following applies only to local external jobs created without credentials on any platform, and remote external jobs on the UNIX and Linux platforms.

On UNIX and Linux, a `SIGTERM` signal is sent to the process launched by the Scheduler. The implementor of the external job is expected to trap the `SIGTERM` in an interrupt handler, clean up whatever work the job has done, and exit. On Windows, `STOP_JOB` with `force` set to `FALSE` is supported only on Windows XP, Windows 2003, and later operating systems. On those platforms, the process launched by the Scheduler is a console process. To stop it, the Scheduler sends a `CTRL-BREAK` to the process. The `CTRL_BREAK` can be handled by registering a handler with the `SetConsoleCtrlHandler()` routine.

Stopping a Chain Job

If a job pointing to a chain is stopped, all steps of the running chain that are running are stopped.

See "Stopping Individual Chain Steps" on page 29-50 for information about stopping individual chain steps.

Dropping Jobs

You drop one or more jobs using the `DROP_JOB` procedure or Enterprise Manager. `DROP_JOB` accepts a comma-delimited list of jobs and job classes. If a job class is supplied, all jobs in the job class are dropped, although the job class itself is not dropped. (The `DROP_JOB_CLASS` procedure should be used to drop a job class. See "Dropping Job Classes" on page 29-55 for information about how to drop job classes.)

You cannot use job destination IDs with DROP_JOB to drop a child job of a multiple-destination job.

The following statement drops jobs job1 and job3, and all jobs in job classes jobclass1 and jobclass2:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB ('job1, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

If a job is running at the time of the procedure call, the attempt to drop the job fails. You can modify this default behavior by setting either the force or defer option.

When you set the force option to TRUE, the Scheduler first attempts to stop the running job by using an interrupt mechanism—calling STOP_JOB with the force option set to FALSE. If the job is successfully stopped, the job is then dropped. Alternatively, you can call STOP_JOB to first stop the job and then call DROP_JOB. If STOP_JOB fails, you can call STOP_JOB with the force option, provided you have the MANAGE SCHEDULER privilege. You can then drop the job. By default, force is set to FALSE for both the STOP_JOB and DROP_JOB procedures.

When you set the defer option to TRUE, the running job is allowed to complete and is then dropped. The force and defer options are mutually exclusive; setting both results in an error.

When you specify multiple jobs to drop, the commit_semantics argument determines the outcome when an error occurs on one of the jobs. The following are the possible values for this argument:

- STOP_ON_FIRST_ERROR, the default—The call returns on the first error and the previous drop operations that were successful are committed to disk.
- TRANSACTIONAL—The call returns on the first error and the previous drop operations before the error are rolled back. force must be FALSE.
- ABSORB_ERRORS—The call tries to absorb any errors, attempts to drop the rest of the jobs, and commits all the drops that were successful.

Setting commit_semantics is valid only when no job classes are included in the job_name list. When you include job classes, default commit semantics (STOP_ON_FIRST_ERROR) are in effect.

The following example drops the jobs myjob1 and myjob2 with the defer option and with transactional commit semantics:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB(
     job_name         => 'myjob1, myjob2',
     defer            => TRUE,
     commit_semantics => 'TRANSACTIONAL');
END;
/
```

This next example illustrates the ABSORB_ERRORS commit semantics. Assume that myjob1 is running when the procedure is called and that myjob2 is not.

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB(
     job_name         => 'myjob1, myjob2',
     commit_semantics => 'ABSORB_ERRORS');
END;
/
```

```
Error report:
ORA-27362: batch API call completed with errors
```

You can query the view SCHEDULER_BATCH_ERRORS to determine the nature of the errors.

```
SELECT object_name, error_code, error_message FROM scheduler_batch_errors;

OBJECT_NAME    ERROR CODE ERROR_MESSAGE
-------------- ---------- --------------------------------------------------
STEVE.MYJOB1        27478 "ORA-27478: job "STEVE.MYJOB1" is running
```

Checking USER_SCHEDULER_JOBS, you would find that myjob2 was successfully dropped and that myjob1 is still present.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DROP_JOB procedure.

Disabling Jobs

You disable one or more jobs using the DISABLE procedure or Enterprise Manager. A job can also become disabled for other reasons. For example, a job will be disabled when the job class it belongs to is dropped. A job is also disabled if either the program or the schedule that it points to is dropped. Note that if the program or schedule that the job points to is disabled, the job will not be disabled and will therefore result in an error when the Scheduler tries to run the job.

Disabling a job means that, although the metadata of the job is there, it should not run and the job coordinator will not pick up these jobs for processing. When a job is disabled, its state in the job table is changed to disabled.

When a job is disabled with the force option set to FALSE and the job is currently running, an error is returned. When force is set to TRUE, the job is disabled, but the currently running instance is allowed to finish.

If commit_semantics is set to STOP_ON_FIRST_ERROR, then the call returns on the first error and the previous disable operations that were successful are committed to disk. If commit_semantics is set to TRANSACTIONAL and force is set to FALSE, then the call returns on the first error and the previous disable operations before the error are rolled back. If commit_semantics is set to ABSORB_ERRORS, then the call tries to absorb any errors and attempts to disable the rest of the jobs and commits all the disable operations that were successful. If the procedure indicates that errors occurred, you can query the view SCHEDULER_BATCH_ERRORS to determine the nature of the errors.

By default, commit_semantics is set to STOP_ON_FIRST_ERROR.

You can also disable several jobs in one call by providing a comma-delimited list of job names or job class names to the DISABLE procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
  DBMS_SCHEDULER.DISABLE('job1, job2, job3, sys.jobclass1, sys.jobclass2');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DISABLE procedure.

Enabling Jobs

You enable one or more jobs by using the ENABLE procedure or Enterprise Manager. The effect of using this procedure is that the job will now be picked up by the job coordinator for processing. Jobs are created disabled by default, so you must enable them before they can run. When a job is enabled, a validity check is performed. If the check fails, the job is not enabled.

If you enable a disabled job, it begins to run immediately according to its schedule. Enabling a disabled job also resets the job RUN_COUNT, FAILURE_COUNT, and RETRY_COUNT attributes.

If commit_semantics is set to STOP_ON_FIRST_ERROR, then the call returns on the first error and the previous enable operations that were successful are committed to disk. If commit_semantics is set to TRANSACTIONAL, then the call returns on the first error and the previous enable operations before the error are rolled back. If commit_semantics is set to ABSORB_ERRORS, then the call tries to absorb any errors and attempts to enable the rest of the jobs and commits all the enable operations that were successful. If the procedure indicates that errors occurred, you can query the view SCHEDULER_BATCH_ERRORS to determine the nature of the errors.

By default, commit_semantics is set to STOP_ON_FIRST_ERROR.

You can enable several jobs in one call by providing a comma-delimited list of job names or job class names to the ENABLE procedure call. For example, the following statement combines jobs with job classes:

```
BEGIN
 DBMS_SCHEDULER.ENABLE ('job1, job2, job3,
   sys.jobclass1, sys.jobclass2, sys.jobclass3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the ENABLE procedure.

Copying Jobs

You copy a job using the COPY_JOB procedure or Enterprise Manager. This call copies all the attributes of the old job to the new job (except job name). The new job is created disabled.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the COPY_JOB procedure.

Viewing stdout and stderr for External Jobs

External jobs with credentials write stdout and stderr to log files. Local external jobs write to log files in the directory *ORACLE_HOME*/scheduler/log. Remote external jobs write to log files in the directory *AGENT_HOME*/data/log. You can retrieve the contents of these files with DBMS_SCHEDULER.GET_FILE. File names consist of the string "_stdout" or "_stderr" appended to a job log ID. You obtain the job log ID for a job by querying the ADDITIONAL_INFO column of the *_SCHEDULER_JOB_RUN_DETAILS views and parsing for a name/value pair that looks similar to this:

```
EXTERNAL_LOG_ID="job_71035_3158"
```

An example file name is job_71035_3158_stdout. Example 29–8, "Creating a Local External Job and Retrieving stdout" on page 29-14 illustrates how to retrieve stdout

output. Although this example is for a local external job, the method is the same for remote external jobs.

In addition, when a local external job or remote external job writes output to `stderr`, the first 200 bytes are recorded in the `ADDITIONAL_INFO` column of the `*_SCHEDULER_JOB_RUN_DETAILS` views. The information is in a name/value pair that looks like this:

```
STANDARD_ERROR="text"
```

Note: The `ADDITIONAL_INFO` column can have multiple name/value pairs. The order is indeterminate, so you must parse the field to locate the `STANDARD_ERROR` name/value pair.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about `DBMS_SCHEDULER.GET_FILE`

Creating and Managing Programs to Define Jobs

A program is a collection of metadata about a particular task. You optionally use a program to help define a job. This section introduces you to basic program tasks, and discusses the following topics:

- Program Tasks and Their Procedures
- Creating Programs
- Altering Programs
- Dropping Programs
- Disabling Programs
- Enabling Programs

See Also: "Programs" on page 28-4 for an overview of programs.

Program Tasks and Their Procedures

Table 29–3 illustrates common program tasks and their appropriate procedures and privileges:

Table 29–3 Program Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a program	`CREATE_PROGRAM`	`CREATE JOB` or `CREATE ANY JOB`
Alter a program	`SET_ATTRIBUTE`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a program	`DROP_PROGRAM`	`ALTER` or `CREATE ANY JOB` or be the owner
Disable a program	`DISABLE`	`ALTER` or `CREATE ANY JOB` or be the owner
Enable a program	`ENABLE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Programs

You create programs by using the `CREATE_PROGRAM` procedure or Enterprise Manager. By default, programs are created in the schema of the creator. To create a program in

another user's schema, you must qualify the program name with the schema name. For other users to use your programs, they must have EXECUTE privileges on the program, therefore, once a program has been created, you have to grant the EXECUTE privilege on it. An example of creating a program is the following, which creates a program called my_program1:

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM (
   program_name           => 'my_program1',
   program_action         => '/usr/local/bin/date',
   program_type           => 'EXECUTABLE',
   comments               => 'My comments here');
END;
/
```

Programs are created in the disabled state by default; you must enable them before you can enable jobs that point to them.

Do not attempt to enable a program that requires arguments before you define all program arguments, which you must do in a DEFINE_*XXX*_ARGUMENT procedure as described in "Defining Program Arguments" on page 29-22.

Defining Program Arguments

After creating a program, you can define program arguments. Arguments are defined by position in the calling sequence, with an optional argument name and optional default value. If no default value is defined for a program argument, the job that references the program must supply an argument value. (The job can also override a default value.) All argument values must be defined before the job can be enabled.

To set program argument values, use the DEFINE_PROGRAM_ARGUMENT or DEFINE_ANYDATA_ARGUMENT procedures. DEFINE_ANYDATA_ARGUMENT is used for complex types that must be encapsulated in an ANYDATA object. An example of a program that might need arguments is one that starts a reporting program that requires a start date and end date. The following code example sets the end date argument, which is the second argument expected by the reporting program. The example also assigns a name to the argument so that you can refer to the argument by name (instead of position) from other package procedures, including SET_JOB_ANYDATA_VALUE and SET_JOB_ARGUMENT_VALUE.

```
BEGIN
 DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_position       => 2,
   argument_name           => 'end_date',
   argument_type           => 'VARCHAR2',
   default_value           => '12-DEC-03');
END;
/
```

Valid values for the argument_type argument are only SQL data types, therefore booleans are not supported. For external executables, only string types such as CHAR or VARCHAR2 are permitted.

You can drop a program argument either by name or by position, as in the following:

```
BEGIN
 DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_position       => 2);
```

```
 DBMS_SCHEDULER.DROP_PROGRAM_ARGUMENT (
   program_name            => 'operations_reporting',
   argument_name           => 'end_date');
END;
/
```

In some special cases, program logic is dependent on the Scheduler environment. The Scheduler has some predefined metadata arguments that can be passed as an argument to the program for this purpose. For example, for some jobs whose schedule is a window name, it is useful to know how much longer the window will be open when the job is started. This is possible by defining the window end time as a metadata argument to the program.

If a program needs access to specific job metadata, you can define a special metadata argument using the `DEFINE_METADATA_ARGUMENT` procedure, so values will be filled in by the Scheduler when the program is executed.

See Also: "Setting Job Arguments" on page 29-10

Altering Programs

You alter a program by modifying its attributes. You can use Enterprise Manager or the `DBMS_SCHEDULER.SET_ATTRIBUTE` and `DBMS_SCHEDULER.SET_ATTRIBUTE_NULL` package procedures to alter programs. See the `DBMS_SCHEDULER.CREATE_PROGRAM` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on program attributes.

If any currently running jobs use the program that you altered, they continue to run with the program as defined before the alter operation.

The following example changes the executable that program `my_program1` runs:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           => 'my_program1',
   attribute      => 'program_action',
   value          => '/usr/local/bin/salesreports1');
END;
/
```

Dropping Programs

You drop one or more programs using the `DROP_PROGRAM` procedure or Enterprise Manager.

Running jobs that point to the program are not affected by the `DROP_PROGRAM` call, and are allowed to continue. Any arguments that pertain to the program are also dropped when the program is dropped. You can drop several programs in one call by providing a comma-delimited list of program names. For example, the following statement drops three programs:

```
BEGIN
  DBMS_SCHEDULER.DROP_PROGRAM('program1, program2, program3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DROP_PROGRAM` procedure.

Disabling Programs

You disable one or more programs using the `DISABLE` procedure or Enterprise Manager. When a program is disabled, the status is changed to `disabled`. A disabled program implies that, although the metadata is still there, jobs that point to this program cannot run.

Running jobs that point to the program are not affected by the `DISABLE` call, and are allowed to continue. Any argument that pertains to the program will not be affected when the program is disabled.

A program can also become disabled for other reasons. For example, if a program argument is dropped or `number_of_arguments` is changed so that all arguments are no longer defined.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DISABLE` procedure.

Enabling Programs

You enable one or more programs using the `ENABLE` procedure or Enterprise Manager. When a program is enabled, the enabled flag is set to `TRUE`. Programs are created disabled by default, therefore, you have to enable them before you can enable jobs that point to them. Before programs are enabled, validity checks are performed to ensure that the action is valid and that all arguments are defined.

You can enable several programs in one call by providing a comma-delimited list of program names to the `ENABLE` procedure call. For example, the following statement enables three programs:

```
BEGIN
  DBMS_SCHEDULER.ENABLE('program1, program2, program3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `ENABLE` procedure.

Creating and Managing Schedules to Define Jobs

You optionally use a schedule object (a schedule) to define when a job should be run. Schedules can be shared among users by creating and saving them as objects in the database.

This section introduces you to basic schedule tasks, and discusses the following topics:

- Schedule Tasks and Their Procedures
- Creating Schedules
- Altering Schedules
- Dropping Schedules
- Setting the Repeat Interval

See Also:

- "Schedules" on page 28-4 for an overview of schedules.
- "Managing Job Scheduling and Job Priorities with Windows" on page 29-55 and "Managing Job Scheduling and Job Priorities with Window Groups" on page 29-60 for a method of scheduling jobs while managing job resource utilization

Schedule Tasks and Their Procedures

Table 29–4 illustrates common schedule tasks and the procedures you use to handle them.

Table 29–4 Schedule Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a schedule	`CREATE_SCHEDULE`	`CREATE JOB` or `CREATE ANY JOB`
Alter a schedule	`SET_ATTRIBUTE`	`ALTER` or `CREATE ANY JOB` or be the owner
Drop a schedule	`DROP_SCHEDULE`	`ALTER` or `CREATE ANY JOB` or be the owner

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Schedules

You create schedules by using the `CREATE_SCHEDULE` procedure or Enterprise Manager. Schedules are created in the schema of the user creating the schedule, and are enabled when first created. You can create a schedule in another user's schema. Once a schedule has been created, it can be used by other users. The schedule is created with access to `PUBLIC`. Therefore, there is no need to explicitly grant access to the schedule. An example of creating a schedule is the following statement:

```
BEGIN
 DBMS_SCHEDULER.CREATE_SCHEDULE (
  schedule_name     => 'my_stats_schedule',
  start_date        => SYSTIMESTAMP,
  end_date          => SYSTIMESTAMP + INTERVAL '30' day,
  repeat_interval   => 'FREQ=HOURLY; INTERVAL=4',
  comments          => 'Every 4 hours');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_SCHEDULE` procedure.
- "Creating an Event Schedule" on page 29-33

Altering Schedules

You alter a schedule by using the `SET_ATTRIBUTE` and `SET_ATTRIBUTE_NULL` package procedures or Enterprise Manager. Altering a schedule changes the definition of the schedule. With the exception of schedule name, all attributes can be changed. The attributes of a schedule are available in the `*_SCHEDULER_SCHEDULES` views.

If a schedule is altered, the change will not affect running jobs and open windows that use this schedule. The change will only be in effect the next time the jobs runs or the window opens.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_ATTRIBUTE` procedure.

Dropping Schedules

You drop a schedule using the `DROP_SCHEDULE` procedure or Enterprise Manager. This procedure call will delete the schedule object from the database.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `DROP_SCHEDULE` procedure.

Setting the Repeat Interval

You control when and how often a job repeats by setting the `repeat_interval` attribute of the job itself or of the named schedule that the job references. You can set `repeat_interval` with `DBMS_SCHEDULER` package procedures or with Enterprise Manager.

The result of evaluating the `repeat_interval` is a set of timestamps. The Scheduler runs the job at each timestamp. Note that the start date from the job or schedule also helps determine the resulting set of timestamps. (See *Oracle Database PL/SQL Packages and Types Reference* for more information about `repeat_interval` evaluation.) If no value for `repeat_interval` is specified, the job runs only once at the specified start date.

Immediately after a job is started, the `repeat_interval` is evaluated to determine the next scheduled execution time of the job. The next scheduled execution time might arrive while the job is still running. A new instance of the job, however, will not be started until the current one completes.

There are two ways to specify the repeat interval:

- Using the Scheduler Calendaring Syntax
- Using a PL/SQL Expression

Using the Scheduler Calendaring Syntax

The primary method of setting how often a job will repeat is by setting the `repeat_interval` attribute with a Scheduler calendaring expression. See *Oracle Database PL/SQL Packages and Types Reference* for a detailed description of the calendaring syntax for `repeat_interval` as well as the `CREATE_SCHEDULE` procedure.

Examples of Calendaring Expressions

The following examples illustrate simple repeat intervals. For simplicity, it is assumed that there is no contribution to the evaluation results by the start date.

Run every Friday. (All three examples are equivalent.)

```
FREQ=DAILY; BYDAY=FRI;
FREQ=WEEKLY; BYDAY=FRI;
FREQ=YEARLY; BYDAY=FRI;
```

Run every other Friday.

```
FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI;
```

Run on the last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=-1;
```

Run on the next to last day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=-2;
```

Run on March 10th. (Both examples are equivalent)

```
FREQ=YEARLY; BYMONTH=MAR; BYMONTHDAY=10;
FREQ=YEARLY; BYDATE=0310;
```

Run every 10 days.

```
FREQ=DAILY; INTERVAL=10;
```

Run daily at 4, 5, and 6PM.

```
FREQ=DAILY; BYHOUR=16,17,18;
```

Run on the 15th day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=15;
```

Run on the 29th day of every month.

```
FREQ=MONTHLY; BYMONTHDAY=29;
```

Run on the second Wednesday of each month.

```
FREQ=MONTHLY; BYDAY=2WED;
```

Run on the last Friday of the year.

```
FREQ=YEARLY; BYDAY=-1FRI;
```

Run every 50 hours.

```
FREQ=HOURLY; INTERVAL=50;
```

Run on the last day of every other month.

```
FREQ=MONTHLY; INTERVAL=2; BYMONTHDAY=-1;
```

Run hourly for the first three days of every month.

```
FREQ=HOURLY; BYMONTHDAY=1,2,3;
```

Here are some more complex repeat intervals:

Run on the last workday of every month (assuming that workdays are Monday through Friday).

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; BYSETPOS=-1
```

Run on the last workday of every month, excluding company holidays. (This example references an existing named schedule called `Company_Holidays`.)

```
FREQ=MONTHLY; BYDAY=MON,TUE,WED,THU,FRI; EXCLUDE=Company_Holidays; BYSETPOS=-1
```

Run at noon every Friday and on company holidays.

```
FREQ=YEARLY;BYDAY=FRI;BYHOUR=12;INCLUDE=Company_Holidays
```

Run on these three holidays: July 4th, Memorial Day, and Labor Day. (This example references three existing named schedules—`JUL4`, `MEM`, and `LAB`—where each defines a single date corresponding to a holiday.)

```
JUL4,MEM,LAB
```

Examples of Calendaring Expression Evaluation

A repeat interval of "`FREQ=MINUTELY;INTERVAL=2;BYHOUR=17;BYMINUTE=2,4,5,50,51,7;`" with a start date of 28-FEB-2004 23:00:00 will generate the following schedule:

```
SUN 29-FEB-2004 17:02:00
SUN 29-FEB-2004 17:04:00
SUN 29-FEB-2004 17:50:00
MON 01-MAR-2004 17:02:00
MON 01-MAR-2004 17:04:00
MON 01-MAR-2004 17:50:00
...
```

A repeat interval of "`FREQ=MONTHLY;BYMONTHDAY=15,-1`" with a start date of 29-DEC-2003 9:00:00 will generate the following schedule:

```
WED 31-DEC-2003 09:00:00
THU 15-JAN-2004 09:00:00
SAT 31-JAN-2004 09:00:00
SUN 15-FEB-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 15-MAR-2004 09:00:00
WED 31-MAR-2004 09:00:00
...
```

A repeat interval of "`FREQ=MONTHLY;`" with a start date of 29-DEC-2003 9:00:00 will generate the following schedule. (Note that because there is no `BYMONTHDAY` clause, the day of month is retrieved from the start date.)

```
MON 29-DEC-2003 09:00:00
THU 29-JAN-2004 09:00:00
SUN 29-FEB-2004 09:00:00
MON 29-MAR-2004 09:00:00
...
```

Example of Using a Calendaring Expression

As an example of using the calendaring syntax, consider the following statement:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job1',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'FREQ=MINUTELY; INTERVAL=30;',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then run until September 15. The job is run every 30 minutes.

Using a PL/SQL Expression

When you need more complicated capabilities than the calendaring syntax provides, you can use PL/SQL expressions. You cannot, however, use PL/SQL expressions for windows or in named schedules. The PL/SQL expression must evaluate to a date or a timestamp. Other than this restriction, there are no limitations, so with sufficient programming, you can create every possible repeat interval. As an example, consider the following statement:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name             => 'scott.my_job2',
   start_date           => '15-JUL-04 01.00.00 AM Europe/Warsaw',
   repeat_interval      => 'SYSTIMESTAMP + INTERVAL '30' MINUTE',
   end_date             => '15-SEP-04 01.00.00 AM Europe/Warsaw',
   comments             => 'My comments here');
END;
/
```

This creates `my_job1` in `scott`. It will run for the first time on July 15th and then every 30 minutes until September 15. The job is run every 30 minutes because `repeat_interval` is set to `SYSTIMESTAMP + INTERVAL '30' MINUTE`, which returns a date 30 minutes into the future.

Differences Between PL/SQL Expression and Calendaring Syntax Behavior

The following are important differences in behavior between a calendaring expression and PL/SQL repeat interval:

- Start date

  Using the calendaring syntax, the start date is a reference date only. Therefore, the schedule is valid as of this date. It does not mean that the job will start on the start date.

  Using a PL/SQL expression, the start date represents the actual time that the job will start executing for the first time.

- Next run time

  Using the calendaring syntax, the next time the job will run is fixed.

  Using the PL/SQL expression, the next time the job will run depends on the actual start time of the current run of the job. As an example of the difference, if a job started at 2:00 PM and its schedule was to repeat every 2 hours, then, if the repeat interval was specified with the calendaring syntax, it would repeat at 4, 6 and so on. If PL/SQL was used and the job started at 2:10, then the job would repeat at 4:10, and if the next job actually started at 4:11, then the subsequent run would be at 6:11.

To illustrate these two points, consider a situation where you have a start date of 15-July-2003 1:45:00 and you want it to repeat every two hours. A calendar expression of "`FREQ=HOURLY; INTERVAL=2; BYMINUTE=0;`" will generate the following schedule:

```
TUE 15-JUL-2003  03:00:00
TUE 15-JUL-2003  05:00:00
TUE 15-JUL-2003  07:00:00
TUE 15-JUL-2003  09:00:00
TUE 15-JUL-2003  11:00:00
...
```

Note that the calendar expression repeats every two hours on the hour.

A PL/SQL expression of "`SYSTIMESTAMP + interval '2' hour`", however, might have a run time of the following:

```
TUE 15-JUL-2003  01:45:00
TUE 15-JUL-2003  03:45:05
TUE 15-JUL-2003  05:45:09
TUE 15-JUL-2003  07:45:14
TUE 15-JUL-2003  09:45:20
...
```

Repeat Intervals and Daylight Savings

For repeating jobs, the next time a job is scheduled to run is stored in a timestamp with time zone column. When using the calendaring syntax, the time zone is retrieved from `start_date`. For more information on what happens when `start_date` is not specified, see *Oracle Database PL/SQL Packages and Types Reference*.

In the case of repeat intervals that are based on PL/SQL expressions, the time zone is part of the timestamp that is returned by the PL/SQL expression. In both cases, it is important to use region names. For example, `"Europe/Istanbul"`, instead of absolute time zone offsets such as `"+2:00"`. Only when a time zone is specified as a region name will the Scheduler follow daylight savings adjustments that apply to that region.

Using Events to Start Jobs

This section contains:

- About Events
- Starting Jobs with Events Raised by Your Application
- Starting a Job When a File Arrives on a System

See Also:

- "Examples of Creating Jobs and Schedules Based on Events" on page 30-21
- "Creating and Managing Job Chains" on page 29-41 for information on how to use events with chains to achieve precise control over process flow

About Events

An **event** is a message sent by one application or system process to another to indicate that some action or occurrence has been detected. An event is **raised** (sent) by one application or process, and **consumed** (received) by one or more applications or processes.

There are two kinds of events consumed by the Scheduler:

- Events raised by your application

  An application can raise an event to be consumed by the Scheduler. The Scheduler reacts to the event by starting a job. For example, when an inventory tracking system notices that the inventory has gone below a certain threshold, it can raise an event that starts an inventory replenishment job.

  See "Starting Jobs with Events Raised by Your Application" on page 29-31.

- File arrival events raised by a file watcher

You can create a file watcher—a Scheduler object introduced in Oracle Database 11*g* Release 2—to watch for the arrival of a file on a system. You can then configure a job to start when the file watcher detects the presence of the file. For example, a data warehouse for a chain of stores loads data from end-of-day revenue reports uploaded from the point-of-sale systems in the stores. The data warehouse load job starts each time a new end-of-day report arrives.

See "Starting a Job When a File Arrives on a System" on page 29-35

See Also:

- *Oracle Database Advanced Queuing User's Guide* for more information on Advanced Queuing
- "Monitoring Job State with Events Raised by the Scheduler" on page 29-68 for information about how your application can consume job state change events raised by the Scheduler

Starting Jobs with Events Raised by Your Application

Your application can raise an event to notify the Scheduler to start a job. A job started in this way is referred to as an event-based job. You can create a named schedule that references an event instead of containing date, time, and recurrence information. If a job is given such a schedule (an **event schedule**), the job runs when the event is raised.

To raise an event to notify the Scheduler to start a job, your application enqueues a message onto an Oracle Streams Advanced Queuing queue that was specified when setting up the job. When the job starts, it can optionally retrieve the message content of the event.

To create an event-based job, you must set these two additional attributes:

- `queue_spec`

  A queue specification that includes the name of the queue where your application enqueues messages to raise job start events, or in the case of a secure queue, the queue name followed by a comma and the agent name.

- `event_condition`

  A conditional expression based on message properties that must evaluate to TRUE for the message to start the job. The expression must have the syntax of an Oracle Streams Advanced Queuing rule. Accordingly, you can include user data properties in the expression, provided that the message payload is an object type, and that you prefix object attributes in the expression with `tab.user_data`.

  For more information on rules, see the `DBMS_AQADM.ADD_SUBSCRIBER` procedure in *Oracle Database PL/SQL Packages and Types Reference*.

  The following example sets `event_condition` to select only low-inventory events that occur after midnight and before 9:00 a.m. Assume that the message payload is an object with two attributes called `event_type` and `event_timestamp`.

  ```
  event_condition = 'tab.user_data.event_type = ''LOW_INVENTORY'' and
  extract hour from tab.user_data.event_timestamp < 9'
  ```

You can specify `queue_spec` and `event_condition` as inline job attributes, or you can create an **event schedule** with these attributes and point to this schedule from the job.

Note: The Scheduler runs the event-based job for each occurrence of an event that matches `event_condition`. However, by default, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job. Beginning in Oracle Database 11*g* Release 1, you can change this default behavior by setting the job attribute `PARALLEL_INSTANCES` to `TRUE`. In this case, an instance of the job is started for every instance of the event, and all job instances are lightweight jobs. See the `SET_ATTRIBUTE` procedure in *Oracle Database PL/SQL Packages and Types Reference* for details.

Table 29–5 describes common administration tasks involving events raised by an application (and consumed by the Scheduler) and the procedures associated with them.

Table 29–5 Event Tasks and Their Procedures for Events Raised by an Application

Task	Procedure	Privilege Needed
Creating an Event-Based Job	`CREATE_JOB`	`CREATE JOB` or `CREATE ANY JOB`
Altering an Event-Based Job	`SET_ATTRIBUTE`	`CREATE ANY JOB` or ownership of the job being altered or ALTER privileges on the job
Creating an Event Schedule	`CREATE_EVENT_SCHEDULE`	`CREATE JOB` or `CREATE ANY JOB`
Altering an Event Schedule	`SET_ATTRIBUTE`	`CREATE ANY JOB` or ownership of the schedule being altered or ALTER privileges on the schedule

See Also: *Oracle Database Advanced Queuing User's Guide* for information on how to create queues and enqueue messages.

Creating an Event-Based Job

You use the `CREATE_JOB` procedure or Enterprise Manager to create an event-based job. The job can include event information inline as job attributes or can specify event information by pointing to an event schedule.

Like jobs based on time schedules, event-based jobs are not auto-dropped unless the job end date passes, `max_runs` is reached, or the maximum number of failures (`max_failures`) is reached.

Specifying Event Information as Job Attributes To specify event information as job attributes, you use an alternate syntax of `CREATE_JOB` that includes the `queue_spec` and `event_condition` attributes.

The following example creates a job that starts when an application signals the Scheduler that inventory levels for an item have fallen to a low threshold level:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  'process_lowinv_j1',
   program_name        =>  'process_lowinv_p1',
   event_condition     =>  'tab.user_data.event_type = ''LOW_INVENTORY''',
   queue_spec          =>  'inv_events_q, inv_agent1',
   enabled             =>  TRUE,
   comments            =>  'Start an inventory replenishment job');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the CREATE_JOB procedure.

Specifying Event Information in an Event Schedule To specify event information with an event schedule, you set the job's schedule_name attribute to the name of an event schedule, as shown in the following example:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name           =>  'process_lowinv_j1',
   program_name       =>  'process_lowinv_p1',
   schedule_name      =>  'inventory_events_schedule',
   enabled            =>  TRUE,
   comments           =>  'Start an inventory replenishment job');
END;
/
```

See "Creating an Event Schedule" on page 29-33 for more information.

Altering an Event-Based Job

You alter an event-based job by using the SET_ATTRIBUTE procedure. For jobs that specify the event inline, you cannot set the queue_spec and event_condition attributes individually with SET_ATTRIBUTE. Instead, you must set an attribute called event_spec, and pass an event condition and queue specification as the third and fourth arguments, respectively, to SET_ATTRIBUTE.

The following is an example of using the event_spec attribute:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE ('my_job', 'event_spec',
   'tab.user_data.event_type = ''LOW_INVENTORY''', 'inv_events_q, inv_agent1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the SET_ATTRIBUTE procedure.

Creating an Event Schedule

You can create a schedule that is based on an event. You can then reuse the schedule for multiple jobs. To do so, use the CREATE_EVENT_SCHEDULE procedure, or use Enterprise Manager. The following is an example of creating an event schedule:

```
BEGIN
  DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'inventory_events_schedule',
   start_date        =>  SYSTIMESTAMP,
   event_condition   =>  'tab.user_data.event_type = ''LOW_INVENTORY''',
   queue_spec        =>  'inv_events_q, inv_agent1');
END;
/
```

You can drop an event schedule using the DROP_SCHEDULE procedure. See *Oracle Database PL/SQL Packages and Types Reference* for more information on CREATE_EVENT_SCHEDULE.

Altering an Event Schedule

You alter the event information in an event schedule in the same way that you alter event information in a job. For more information, see "Altering an Event-Based Job" on page 29-33.

The following example demonstrates how to use the SET_ATTRIBUTE procedure and the event_spec attribute to alter event information in an event schedule.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE ('inventory_events_schedule', 'event_spec',
   'tab.user_data.event_type = ''LOW_INVENTORY''', 'inv_events_q, inv_agent1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the SET_ATTRIBUTE procedure.

Passing Event Messages into an Event-Based Job

Through a metadata argument, the Scheduler can pass to an event-based job the message content of the event that started the job. The following rules apply:

- The job must use a named program of type STORED_PROCEDURE.
- One of the named program's arguments must be a metadata argument with metadata_attribute set to EVENT_MESSAGE.
- The stored procedure that implements the program must have an argument at the position corresponding to the named program's metadata argument. The argument type must be the data type of the queue where your application queues the job-start event.

If you use the RUN_JOB procedure to manually run a job that has an EVENT_MESSAGE metadata argument, the value passed to that argument is NULL.

The following example shows how to construct an event-based job that can receive the event message content:

```
create or replace procedure my_stored_proc (event_msg IN event_queue_type)
as
begin
  -- retrieve and process message body
end;
/

begin
  dbms_scheduler.create_program (
      program_name => 'my_prog',
      program_action=> 'my_stored_proc',
      program_type => 'STORED_PROCEDURE',
      number_of_arguments => 1,
      enabled => FALSE) ;

  dbms_scheduler.define_metadata_argument (
      program_name => 'my_prog',
      argument_position => 1 ,
      metadata_attribute => 'EVENT_MESSAGE') ;

  dbms_scheduler.enable ('my_prog');
exception
  when others then raise ;
end ;
```

```
/

begin
  dbms_scheduler.create_job (
     job_name => 'my_evt_job' ,
     program_name => 'my_prog',
     schedule_name => 'my_evt_sch',
     enabled => true,
     auto_Drop => false) ;
exception
  when others then raise ;
end ;
/
```

Starting a Job When a File Arrives on a System

You can configure the Scheduler to start a job when a file arrives on the local system or a remote system. The job is an event-based job, and the file arrival event is raised by a file watcher, which is a Scheduler object introduced in Oracle Database 11*g* Release 2.

This section contains:

- About File Watchers
- Enabling File Arrival Events from Remote Systems
- Creating File Watchers and File Watcher Jobs
- File Arrival Example
- Managing File Watchers
- Viewing File Watcher Information

About File Watchers

A **file watcher** is a Scheduler object that defines the location, name, and other properties of a file whose arrival on a system causes the Scheduler to start a job. You create a file watcher and then create any number of event-based jobs or event schedules that reference the file watcher. When the file watcher detects the arrival of the designated file, it raises a file arrival event. The job started by the file arrival event can retrieve the event message to learn about the newly arrived file. The message contains the information required to find the file, open it, and process it.

A file watcher can watch for a file on the local system (the same host computer running Oracle Database) or a remote system. Remote systems must be running the Scheduler agent, and the agent must be registered with the database.

File watchers check for the arrival of files every 10 minutes. You can adjust this interval. See "Changing the File Arrival Detection Interval" on page 29-40 for details.

You must have the `CREATE JOB` system privilege to create a file watcher in your own schema. You require the `CREATE ANY JOB` system privilege to create a file watcher in a schema different from your own (except the `SYS` schema, which is disallowed). You can grant the `EXECUTE` object privilege on a file watcher so that jobs in different schemas can reference it. You can also grant the `ALTER` object privilege on a file watcher so that another user can modify it.

To use file watchers, the database Java virtual machine (JVM) component must be installed.

Enabling File Arrival Events from Remote Systems

To receive file arrival events from a remote system, you must install the Scheduler agent on that system, and you must register the agent with the database. The remote system does not require a running Oracle Database instance to generate file arrival events.

To enable the raising of file arrival events at remote systems:

1. Set up the local database to run remote external jobs.

   See "Setting up Databases for Remote Jobs" on page 30-5 for instructions.

2. Install, configure, register, and start the Scheduler agent on the first remote system.

   See "Installing and Configuring the Scheduler Agent on a Remote Host" on page 30-6 for instructions.

   This adds the remote host to the list of external destinations maintained on the local database.

3. Repeat the previous step for each additional remote system.

Creating File Watchers and File Watcher Jobs

You perform the following tasks to create a file watcher and create the event-based job that starts when the designated file arrives.

Task 1 - Create a Credential

The file watcher requires a Scheduler credential object (a credential) with which to authenticate with the host operating system for access to the file. See "Credentials" on page 28-7 for information on privileges required to create credentials.

Perform these steps:

1. Create a credential for the operating system user that must have access to the watched-for file.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CREDENTIAL('WATCH_CREDENTIAL', 'salesapps', 'sa324w1');
END;
/
```

2. Grant the `EXECUTE` object privilege on the credential to the schema that owns the event-based job that the file watcher will start.

```
GRANT EXECUTE ON WATCH_CREDENTIAL to DSSUSER;
```

Task 2 - Create a File Watcher

Perform these steps:

1. Create the file watcher, assigning attributes as described in the `DBMS_SCHEDULER.CREATE_FILE_WATCHER` procedure documentation in *Oracle Database PL/SQL Packages and Types Reference*. You can specify wildcard parameters in the file name. A '?' prefix in the `DIRECTORY_PATH` attribute denotes the path to the Oracle home directory. A `NULL destination` indicates the local host. To watch for the file on a remote host, provide a valid external destination name, which you can obtain from the view `ALL_SCHEDULER_EXTERNAL_DESTS`.

```
BEGIN
  DBMS_SCHEDULER.CREATE_FILE_WATCHER(
    FILE_WATCHER_NAME => 'EOD_FILE_WATCHER',
```

```
    DIRECTORY_PATH    => '?/eod_reports',
    FILE_NAME         => 'eod*.txt',
    CREDENTIAL_NAME   => 'WATCH_CREDENTIAL',
    DESTINATION       => NULL,
    ENABLED           => FALSE);
END;
/
```

2. Grant `EXECUTE` on the file watcher to any schema that owns an event-based job that references the file watcher.

```
GRANT EXECUTE ON EOD_FILE_WATCHER to DSSUSER;
```

Task 3 - Create a Program Object with a Metadata Argument

So that your application can retrieve the file arrival event message content, which includes file name, file size, and so on, create a Scheduler program object with a metadata argument that references the event message.

Perform these steps:

1. Create the program.

```
BEGIN
  DBMS_SCHEDULER.CREATE_PROGRAM(
    PROGRAM_NAME        => 'DSSUSER.EOD_PROGRAM',
    PROGRAM_TYPE        => 'STORED_PROCEDURE',
    PROGRAM_ACTION      => 'EOD_PROCESSOR',
    NUMBER_OF_ARGUMENTS => 1,
    ENABLED             => FALSE);
END;
/
```

2. Define the metadata argument using the `event_message` attribute.

```
BEGIN
  DBMS_SCHEDULER.DEFINE_METADATA_ARGUMENT(
    PROGRAM_NAME       => 'DSSUSER.EOD_PROGRAM',
    METADATA_ATTRIBUTE => 'event_message',
    ARGUMENT_POSITION  => 1);
END;
/
```

3. Create the stored procedure that the program invokes.

   The stored procedure that processes the file arrival event must have an argument of type `SYS.SCHEDULER_FILEWATCHER_RESULT`, which is the data type of the event message. The position of that argument must match the position of the defined metadata argument. The procedure can then access attributes of this abstract data type to learn about the arrived file.

   See Also:

   - *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DEFINE_METADATA_ARGUMENT` procedure
   - *Oracle Database PL/SQL Packages and Types Reference* for a description of the `SYS.SCHEDULER_FILEWATCHER_RESULT` type

Task 4 - Create an Event-Based Job That References the File Watcher

Create the event-based job as described in "Creating an Event-Based Job" on page 29-32, with the following exception: instead of providing a queue specification in

the queue_spec attribute, provide the name of the file watcher. You would typically leave the event_condition job attribute null, but you can provide a condition if desired.

As an alternative to setting the queue_spec attribute for the job, you can create an event schedule, reference the file watcher in the queue_spec attribute of the event schedule, and reference the event schedule in the schedule_name attribute of the job.

Perform these steps to prepare the event-based job:

1. Create the job.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    JOB_NAME        => 'DSSUSER.EOD_JOB',
    PROGRAM_NAME    => 'DSSUSER.EOD_PROGRAM',
    EVENT_CONDITION => NULL,
    QUEUE_SPEC      => 'EOD_FILE_WATCHER',
    AUTO_DROP       => FALSE,
    ENABLED         => FALSE);
END;
/
```

2. If you want the job to run for each instance of the file arrival event, even if the job is already processing a previous event, set the parallel_instances attribute to TRUE. With this setting, the job runs as a lightweight job so that multiple instances of the job can be started quickly. To discard file watcher events that occur while the event-based job is already processing another, leave the parallel_instances attribute FALSE (the default).

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE('DSSUSER.EOD_JOB','PARALLEL_INSTANCES',TRUE);
END;
/
```

For more information about this attribute, see the SET_ATTRIBUTE description in *Oracle Database PL/SQL Packages and Types Reference*.

See Also:

- "Creating an Event Schedule" on page 29-33
- "Creating Jobs Using Named Programs and Schedules" on page 29-5

Task 5 - Enable All Objects

Enable the file watcher, the program, and the job.

```
BEGIN
   DBMS_SCHEDULER.ENABLE('DSSUSER.EOD_PROGRAM,DSSUSER.EOD_JOB,EOD_FILE_WATCHER');
END;
/
```

File Arrival Example

In this example, an event-based job watches for the arrival of end-of-day sales reports onto the local host from various locations. As each report file arrives, a stored procedure captures information about the file and stores the information in a table called eod_reports. A regularly scheduled report aggregation job can then query this table, process all unprocessed files, and mark any newly processed files as processed.

It is assumed that the database user running the following code has been granted `EXECUTE` on the `SYS.SCHEDULER_FILEWATCHER_RESULT` data type.

```
begin
  dbms_scheduler.create_credential(
     credential_name => 'watch_credential',
     username        => 'pos1',
     password        => 'jk4545st');
end;
/

create table eod_reports (when timestamp, file_name varchar2(100),
   file_size number, processed char(1));

create or replace procedure q_eod_report
  (payload IN sys.scheduler_filewatcher_result) as
begin
  insert into eod_reports values
     (payload.file_timestamp,
      payload.directory_path || '/' || payload.actual_file_name,
      payload.file_size,
      'N');
end;
/

begin
  dbms_scheduler.create_program(
    program_name        => 'eod_prog',
    program_type        => 'stored_procedure',
    program_action      => 'q_eod_report',
    number_of_arguments => 1,
    enabled             => false);
  dbms_scheduler.define_metadata_argument(
    program_name        => 'eod_prog',
    metadata_attribute  => 'event_message',
    argument_position   => 1);
  dbms_scheduler.enable('eod_prog');
end;
/

begin
  dbms_scheduler.create_file_watcher(
    file_watcher_name => 'eod_reports_watcher',
    directory_path    => '?/eod_reports',
    file_name         => 'eod*.txt',
    credential_name   => 'watch_credential',
    destination       => null,
    enabled           => false);
end;
/

begin
  dbms_scheduler.create_job(
    job_name        => 'eod_job',
    program_name    => 'eod_prog',
    event_condition => 'tab.user_data.file_size > 10',
    queue_spec      => 'eod_reports_watcher',
    auto_drop       => false,
    enabled         => false);
  dbms_scheduler.set_attribute('eod_job','parallel_instances',true);
end;
```

```
/

exec dbms_scheduler.enable('eod_reports_watcher,eod_job');
```

Managing File Watchers

The DBMS_SCHEDULER PL/SQL package provides procedures for enabling, disabling, dropping, and setting attributes for file watchers.

The section contains:

- Enabling File Watchers
- Altering File Watchers
- Disabling and Dropping File Watchers
- Changing the File Arrival Detection Interval

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_SCHEDULER PL/SQL package

Enabling File Watchers If a file watcher is disabled, use DBMS_SCHEDULER.ENABLE to enable it, as shown in Task 5, "- Enable All Objects" on page 29-38.

You can enable a file watcher only if all of its attributes are set to legal values and the file watcher owner has EXECUTE privileges on the specified credential.

Altering File Watchers Use the DBMS_SCHEDULER.SET_ATTRIBUTE and DBMS_SCHEDULER.SET_ATTRIBUTE_NULL package procedures to modify the attributes of a file watcher. See the CREATE_FILE_WATCHER procedure description for information about file watcher attributes.

Disabling and Dropping File Watchers Use DBMS_SCHEDULER.DISABLE to disable a file watcher and DBMS_SCHEDULER.DROP_FILE_WATCHER to drop a file watcher. You cannot disable or drop a file watcher if there are jobs that depend on it. To force a disable or drop operation in this case, set the FORCE attribute to TRUE. If you force disabling or dropping a file watcher, jobs that depend on it become disabled.

Changing the File Arrival Detection Interval File watchers check for the arrival of files every ten minutes by default. You can change this interval.

To change the file arrival detection interval:

1. Connect to the database as the SYS user.
2. Change the REPEAT_INTERVAL attribute of the predefined schedule SYS.FILE_WATCHER_SCHEDULE. Use any valid calendaring syntax.

   The following example changes the file arrival detection frequency to every two minutes.

   ```
   BEGIN
     DBMS_SCHEDULER.SET_ATTRIBUTE('FILE_WATCHER_SCHEDULE', 'REPEAT_INTERVAL',
       'FREQ=MINUTELY;INTERVAL=2');
   END;
   /
   ```

Viewing File Watcher Information

You can view information about file watchers by querying the views *_SCHEDULER_FILE_WATCHERS.

```
SELECT file_watcher_name, destination, directory_path, file_name, credential_name
   FROM dba_scheduler_file_watchers;

FILE_WATCHER_NAME    DESTINATION          DIRECTORY_PATH       FILE_NAME  CREDENTIAL_NAME
-------------------- -------------------- -------------------- ---------- ----------------
MYFW                 dsshost.example.com  /tmp                 abc        MYFW_CRED
EOD_FILE_WATCHER                          ?/eod_reports        eod*.txt   WATCH_CREDENTIAL
```

See Also: *Oracle Database Reference* for details on the `*_SCHEDULER_FILE_WATCHERS` views

Creating and Managing Job Chains

A job chain ("chain") is a named series of tasks that are linked together for a combined objective. Chains are the means by which you can implement dependency based scheduling, in which jobs are started depending on the outcomes of one or more previous jobs.

To create and use a chain, you complete these tasks in order:

Task	See...
1. Create a chain object	Creating Chains
2. Define the steps in the chain	Defining Chain Steps
3. Add rules	Adding Rules to a Chain
4. Enable the chain	Enabling Chains
5. Create a job (the "chain job") that points to the chain	Creating Jobs for Chains

Additional topics discussed in this section include:

- Chain Tasks and Their Procedures
- Dropping Chains
- Running Chains
- Dropping Chain Rules
- Disabling Chains
- Dropping Chain Steps
- Stopping Chains
- Stopping Individual Chain Steps
- Pausing Chains
- Skipping Chain Steps
- Running Part of a Chain
- Monitoring Running Chains
- Handling Stalled Chains

See Also:

- "Chains" on page 28-8 for an overview of chains
- "Examples of Creating Chains" on page 30-19

Chain Tasks and Their Procedures

Table 29–6 illustrates common tasks involving chains and the procedures associated with them.

Table 29–6 Chain Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a chain	CREATE_CHAIN	CREATE JOB, CREATE EVALUATION CONTEXT, CREATE RULE, and CREATE RULE SET if the owner. CREATE ANY JOB, CREATE ANY RULE, CREATE ANY RULE SET, and CREATE ANY EVALUATION CONTEXT otherwise
Drop a chain	DROP_CHAIN	Ownership of the chain or ALTER privileges on the chain or CREATE ANY JOB privileges. If not owner, also requires DROP ANY EVALUATION CONTEXT and DROP ANY RULE SET
Alter a chain	SET_ATTRIBUTE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB
Alter a running chain	ALTER_RUNNING_CHAIN	Ownership of the job, or ALTER privileges on the job or CREATE ANY JOB
Run a chain	RUN_CHAIN	CREATE JOB or CREATE ANY JOB. In addition, the owner of the new job must have EXECUTE privileges on the chain or EXECUTE ANY PROGRAM
Add rules to a chain	DEFINE_CHAIN_RULE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB privileges. CREATE RULE if the owner of the chain, CREATE ANY RULE otherwise
Alter rules in a chain	DEFINE_CHAIN_RULE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB privileges. If not owner of the chain, requires ALTER privileges on the rule or ALTER ANY RULE
Drop rules from a chain	DROP_CHAIN_RULE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB privileges. DROP ANY RULE if not the owner of the chain
Enable a chain	ENABLE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB
Disable a chain	DISABLE	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB
Create steps	DEFINE_CHAIN_STEP	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB
Drop steps	DROP_CHAIN_STEP	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB
Alter steps (including assigning additional attribute values to steps)	ALTER_CHAIN	Ownership of the chain, or ALTER privileges on the chain or CREATE ANY JOB

Creating Chains

You create a chain by using the CREATE_CHAIN procedure. You must ensure that you have the required privileges first. See "Setting Chain Privileges" on page 30-2 for details.

After creating the chain object with CREATE_CHAIN, you define chain steps and chain rules separately.

The following is an example of creating a chain:

```
BEGIN
DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => NULL,
   comments            => 'My first chain');
END;
/
```

The `rule_set_name` and `evaluation_interval` arguments are typically left `NULL`. `evaluation_interval` can define a repeating interval at which chain rules get evaluated. `rule_set_name` refers to a rule set as defined within Oracle Streams.

See Also:

- "Adding Rules to a Chain" on page 29-44 for more information about the `evaluation_interval` attribute.
- See *Oracle Database PL/SQL Packages and Types Reference* for more information on `CREATE_CHAIN`
- See *Oracle Streams Concepts and Administration* for information on rules and rule sets

Defining Chain Steps

After creating a chain object, you define one or more chain steps. Each step can point to one of the following:

- A Scheduler program object (program)
- Another chain (a nested chain)
- An event schedule, inline event, or file watcher

You define a step that points to a program or nested chain by using the `DEFINE_CHAIN_STEP` procedure. An example is the following, which adds two steps to `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step1',
   program_name    =>  'my_program1');
  DBMS_SCHEDULER.DEFINE_CHAIN_STEP (
   chain_name      =>  'my_chain1',
   step_name       =>  'my_step2',
   program_name    =>  'my_chain2');
END;
/
```

The named program or chain does not have to exist when defining the step. However it must exist and be enabled when the chain runs, otherwise an error is generated.

You define a step that waits for an event to occur by using the `DEFINE_CHAIN_EVENT_STEP` procedure. Procedure arguments can point to an event schedule, can include an inline queue specification and event condition, or can include a file watcher name. This example creates a third chain step that waits for the event specified in the named event schedule:

```
BEGIN
  DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP (
   chain_name           =>  'my_chain1',
   step_name            =>  'my_step3',
   event_schedule_name  =>  'my_event_schedule');
END;
/
```

An event step does not wait for its event until the step is started.

Steps That Run Local External Executables

After defining a step that runs a local external executable, you must use the ALTER_CHAIN procedure to assign a credential to the step, as shown in the following example:

```
BEGIN
  DBMS_SCHEDULER.ALTER_CHAIN('chain1','step1','credential_name','MY_CREDENTIAL');
END;
/
```

Steps That Run on Remote Destinations

After defining a step that is to run an external executable on a remote host or a database program unit on a remote database, you must use the ALTER_CHAIN procedure to assign both a credential and a destination to the step, as shown in the following example:

```
BEGIN
 DBMS_SCHEDULER.ALTER_CHAIN('chain1','step2','credential_name','DW_CREDENTIAL');
 DBMS_SCHEDULER.ALTER_CHAIN('chain1','step2','destination_name','DBHOST1_ORCLDW');
END;
/
```

Making Steps Restartable

After a database recovery, by default steps that were running are marked as STOPPED and the chain continues. You can specify the chain steps to restart automatically after a database recovery by using ALTER_CHAIN to set the restart_on_recovery attribute to TRUE for those steps.

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the DEFINE_CHAIN_STEP, DEFINE_CHAIN_EVENT_STEP, and ALTER_CHAIN procedures.

See Also:

- "About Events" on page 29-30
- "About File Watchers" on page 29-35
- "Credentials" on page 28-7
- "Destinations" on page 28-6

Adding Rules to a Chain

You add a rule to a chain with the DEFINE_CHAIN_RULE procedure. You call this procedure once for each rule that you want to add to the chain.

Chain rules define when steps run, and define dependencies between steps. Each rule has a condition and an action. Whenever rules are evaluated, if a rule's condition evaluates to TRUE, its action is performed. The condition can contain Scheduler chain condition syntax or any syntax that is valid in a SQL WHERE clause. The syntax can include references to attributes of any chain step, including step completion status. A typical action is to run a specified step or to run a list of steps.

All chain rules work together to define the overall action of the chain. When the chain job starts and at the end of each step, all rules are evaluated to see what action or actions occur next. If more than one rule has a TRUE condition, multiple actions can occur. You can cause rules to also be evaluated at regular intervals by setting the evaluation_interval attribute of a chain.

Conditions are usually based on the outcome of one or more previous steps. For example, you might want one step to run if the two previous steps succeeded, and another to run if either of the two previous steps failed.

Scheduler chain condition syntax takes one of the following two forms:

```
stepname [NOT] {SUCCEEDED|FAILED|STOPPED|COMPLETED}
stepname ERROR_CODE {comparision_operator|[NOT] IN} {integer|list_of_integers}
```

You can combine conditions with boolean operators AND, OR, and NOT() to create conditional expressions. You can employ parentheses in your expressions to determine order of evaluation.

ERROR_CODE can be set with the RAISE_APPLICATION_ERROR PL/SQL statement within the program assigned to the step. Although the error codes that your program sets in this way are negative numbers, when testing ERROR_CODE in a chain rule, you test for positive numbers. For example, if your program contains the following statement:

```
RAISE_APPLICATION_ERROR(-20100, errmsg);
```

your chain rule condition must be the following:

```
stepname ERROR_CODE=20100
```

Step Attributes

The following is a list of step attributes that you can include in conditions when using SQL WHERE clause syntax:

```
completed
state
start_date
end_date
error_code
duration
```

The completed attribute is boolean and is TRUE when the state attribute is either SUCCEEDED, FAILED, or STOPPED.

Table 29–7 shows the possible values for the state attribute. These values are visible in the STATE column of the *_SCHEDULER_RUNNING_CHAINS views.

Table 29–7 Values for the State Attribute of a Chain Step

State Attribute Value	Meaning
NOT_STARTED	The step's chain is running, but the step has not yet started.
SCHEDULED	A rule started the step with an AFTER clause and the designated wait time has not yet expired.
RUNNING	The step is running. For an event step, the step was started and is waiting for an event.
PAUSED	The step's PAUSE attribute is set to TRUE and the step is paused. It must be unpaused before steps that depend on it can start.
SUCCEEDED	The step completed successfully. The step's ERROR_CODE is 0.
FAILED	The step completed with a failure. ERROR_CODE is nonzero.
STOPPED	The step was stopped with the STOP_JOB procedure.
STALLED	The step is a nested chain that has stalled.

See the DEFINE_CHAIN_RULE procedure in *Oracle Database PL/SQL Packages and Types Reference* for rules and examples for SQL WHERE clause syntax.

Condition Examples Using Scheduler Chain Condition Syntax

These examples use Scheduler chain condition syntax.

Steps started by rules containing the following condition are started when the step named form_validation_step is completed (SUCCEEDED, FAILED, or STOPPED).

```
form_validation_step COMPLETED
```

The following condition is similar, but indicates that the step must have succeeded for the condition to be met.

```
form_validation_step SUCCEEDED
```

The next condition tests for an error. It is TRUE if the step form_validation_step failed with any error code other than 20001.

```
form_validation_step FAILED AND form_validation_step ERROR_CODE != 20001
```

See the DEFINE_CHAIN_RULE procedure in *Oracle Database PL/SQL Packages and Types Reference* for more examples.

Condition Examples Using SQL WHERE Syntax

```
':step1.state=''SUCCEEDED'''
```

Starting the Chain

At least one rule must have a condition that always evaluates to TRUE so that the chain can start when the chain job starts. The easiest way to accomplish this is to just set the condition to 'TRUE' if you are using Schedule chain condition syntax, or '1=1' if you are using SQL syntax.

Ending the Chain

At least one chain rule must contain an action of 'END'. A chain job does not complete until one of the rules containing the END action evaluates to TRUE. Several different rules with different END actions are common, some with error codes, and some without.

If a chain has no more running steps or it is not waiting for an event to occur, and no rules containing the END action evaluate to TRUE (or there are no rules with the END action), the chain job enters the CHAIN_STALLED state. See "Handling Stalled Chains" on page 29-53 for more information.

Example of Defining Rules

The following example defines a rule that starts the chain at step step1 and a rule that starts step step2 when step1 completes. rule_name and comments are optional and default to NULL. If you do use rule_name, you can later redefine that rule with another call to DEFINE_CHAIN_RULE. The new definition overwrites the previous one.

```
BEGIN
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   condition    =>   'TRUE',
   action       =>   'START step1',
   rule_name    =>   'my_rule1',
   comments     =>   'start the chain');
```

```
DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   condition    =>   'step1 completed',
   action       =>   'START step2',
   rule_name    =>   'my_rule2');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information on the `DEFINE_CHAIN_RULE` procedure and Scheduler chain condition syntax
- "Examples of Creating Chains" on page 30-19

Setting an Evaluation Interval for Chain Rules

The Scheduler evaluates all chain rules at the start of the chain job and at the end of each chain step. You can configure a chain to also have its rules evaluated at a repeating time interval, such as once per hour. This capability is useful to start chain steps based on time of day or based on occurrences external to the chain. Here are some examples:

- A chain step is resource-intensive and must therefore run at off-peak hours. You could condition the step on both the completion of another step and on the time of day being after 6:00 p.m and before midnight. The Scheduler would then have to evaluate rules every so often to determine when this condition becomes `TRUE`.
- A step must wait for data to arrive in a table from some other process that is external to the chain. You could condition this step on both the completion of another step and on a particular table containing rows. The Scheduler would then have to evaluate rules every so often to determine when this condition becomes `TRUE`. The condition would use SQL `WHERE` clause syntax, and would be similar to the following:

  ```
  ':step1.state=''SUCCEEDED'' AND select count(*) from oe.sync_table > 0'
  ```

To set an evaluation interval for a chain, you set the `evaluation_interval` attribute when you create the chain. The data type for this attribute is `INTERVAL DAY TO SECOND`.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name          => 'my_chain1',
   rule_set_name       => NULL,
   evaluation_interval => INTERVAL '30' MINUTE,
   comments            => 'Chain with 30 minute evaluation interval');
END;
/
```

Enabling Chains

You enable a chain with the `ENABLE` procedure. A chain must be enabled before it can be run by a job. Enabling an already enabled chain does not return an error.

The following example enables chain `my_chain1`:

```
BEGIN
  DBMS_SCHEDULER.ENABLE ('my_chain1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the ENABLE procedure.

Note: Chains are automatically disabled by the Scheduler when:

- The program that one of the chain steps points to is dropped
- The nested chain that one of the chain steps points to is dropped
- The event schedule that one of the chain event steps points to is dropped

Creating Jobs for Chains

To run a chain, you must either use the RUN_CHAIN procedure or create and schedule a job of type 'CHAIN' (a **chain job**). The job action must refer to the chain name, as shown in the following example:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

For every step of a chain job that is running, the Scheduler creates a **step job** with the same job name and owner as the chain job. Each step job additionally has a job subname to uniquely identify it. The job subname is included as a column in the views *_SCHEDULER_RUNNING_JOBS, *_SCHEDULER_JOB_LOG, and *_SCHEDULER_JOB_RUN_DETAILS. The job subname is normally the same as the step name except in the following cases:

- For nested chains, the current step name may have already been used as a job subname. In this case, the Scheduler appends '_*N*' to the step name, where *N* is an integer that results in a unique job subname.
- If there is a failure when creating a step job, the Scheduler logs a FAILED entry in the job log views (*_SCHEDULER_JOB_LOG and *_SCHEDULER_JOB_RUN_DETAILS) with the job subname set to '*step_name*_0'.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information on the CREATE_JOB procedure.
- "Running Chains" on page 29-49 for another way to run a chain without creating a chain job.

Dropping Chains

You drop a chain, including its steps and rules, by using the DROP_CHAIN procedure. An example of dropping a chain is the following, which drops my_chain1:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN (
   chain_name   => 'my_chain1',
   force        => TRUE);
END;
```

```
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the DROP_CHAIN procedure.

Running Chains

You can use the following two procedures to run a chain immediately:

- RUN_JOB
- RUN_CHAIN

If you already created a chain job for a chain, you can use the RUN_JOB procedure to run that job (and thus run the chain), but you must set the use_current_session argument of RUN_JOB to FALSE.

You can use the RUN_CHAIN procedure to run a chain without having to first create a chain job for the chain. You can also use RUN_CHAIN to run only part of a chain.

RUN_CHAIN creates a temporary job to run the specified chain. If you supply a job name, the job is created with that name, otherwise a default job name is assigned.

If you supply a list of *start steps*, only those steps are started when the chain begins running. (Steps that would normally have started do not run if they are not in the list.) If no list of start steps is given, the chain starts normally—that is, an initial evaluation is done to see which steps to start running. An example is the following, which immediately runs the chain my_chain1:

```
BEGIN
  DBMS_SCHEDULER.RUN_CHAIN (
   chain_name    =>  'my_chain1',
   job_name      =>  'partial_chain_job',
   start_steps   =>  'my_step2, my_step4');
END;
/
```

See Also:

- "Running Part of a Chain" on page 29-52
- *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the RUN_CHAIN procedure.

Dropping Chain Rules

You drop a rule from a chain by using the DROP_CHAIN_RULE procedure. An example is the following, which drops my_rule1:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN_RULE (
   chain_name   =>   'my_chain1',
   rule_name    =>   'my_rule1',
   force        =>   TRUE);
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the DROP_CHAIN_RULE procedure.

Disabling Chains

You disable a chain by using the DISABLE procedure. An example is the following, which disables my_chain1:

```
BEGIN
  DBMS_SCHEDULER.DISABLE ('my_chain1');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the DISABLE procedure.

Note: Chains are automatically disabled by the Scheduler when:

- The program that one of the chain steps points to is dropped
- The nested chain that one of the chain steps points to is dropped
- The event schedule that one of the chain event steps points to is dropped

---

Dropping Chain Steps

You drop a step from a chain by using the DROP_CHAIN_STEP procedure. An example is the following, which drops my_step2 from my_chain2:

```
BEGIN
  DBMS_SCHEDULER.DROP_CHAIN_STEP (
   chain_name   =>   'my_chain2',
   step_name    =>   'my_step2',
   force        =>    TRUE);
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the DROP_CHAIN_STEP procedure.

Stopping Chains

To stop a running chain, you call DBMS_SCHEDULER.STOP_JOB, passing the name of the chain job (the job that started the chain). When you stop a chain job, all steps of the chain that are running are stopped and the chain ends.

See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the STOP_JOB procedure.

Stopping Individual Chain Steps

There are two ways to stop individual chain steps:

- By creating a chain rule that stops one or more steps when the rule condition is met.
- By calling the STOP_JOB procedure.

  For each step to stop, you must specify the schema name, chain job name, and step job subname.

  ```
  BEGIN
    DBMS_SCHEDULER.STOP_JOB('oe.chainrunjob.stepa');
  ```

```
END;
/
```

In this example, `chainrunjob` is the chain job name and `stepa` is the step job subname. The step job subname is typically the same as the step name, but not always. You can obtain the step job subname from the `STEP_JOB_SUBNAME` column of the `*_SCHEDULER_RUNNING_CHAINS` views.

When you stop a chain step, its `state` is set to `STOPPED` and the chain rules are evaluated to determine the steps to run next.

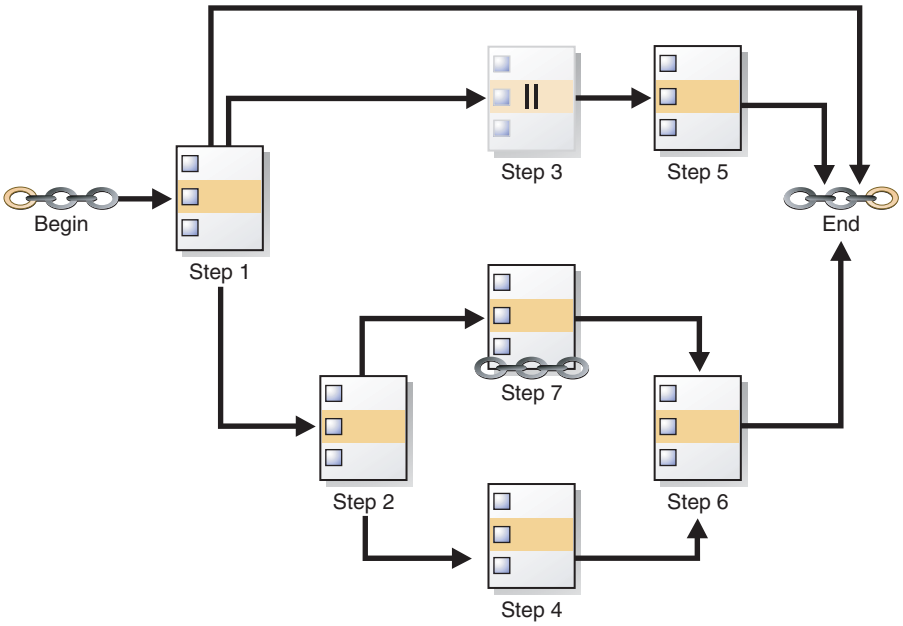
See *Oracle Database PL/SQL Packages and Types Reference* for more information regarding the `STOP_JOB` procedure.

Pausing Chains

You can pause an entire chain or individual branches of a chain. You do so by setting the `PAUSE` attribute of one or more steps to `TRUE` with `DBMS_SCHEDULER.ALTER_CHAIN` or `ALTER_RUNNING_CHAIN`. Pausing chain steps enables you to suspend the running of the chain after those steps run.

When you pause a step, after the step runs, its `state` attribute changes to `PAUSED`, and its `completed` attribute remains `FALSE`. Steps that depend on the completion of the paused step are therefore not run. If you reset the `PAUSE` attribute to `FALSE` for a paused step, its `state` attribute is set to its completion state (`SUCCEEDED`, `FAILED`, or `STOPPED`), and steps that are awaiting the completion of the paused step can then run.

Figure 29–1 Chain with Step 3 Paused



In Figure 29–1, Step 3 is paused. Until Step 3 is unpaused, Step 5 will not run. If you were to pause only Step 2, then Steps 4, 6, and 7 would not run. However Steps 1, 3, and 5 could run. In either case, you are suspending only one branch of the chain.

To pause an entire chain, you pause all steps of the chain. To unpause a chain, you unpause one, many, or all of the chain steps. With the chain in Figure 29–1, pausing Step 1 would also achieve the pausing of the entire chain after Step 1 runs.

> **See Also:** The DBMS_SCHEDULER.ALTER_CHAIN and DBMS_SCHEDULER.ALTER_RUNNING_CHAIN procedures in *Oracle Database PL/SQL Packages and Types Reference*

Skipping Chain Steps

You can skip one or more steps in a chain. You do so by setting the SKIP attribute of one or more steps to TRUE with DBMS_SCHEDULER.ALTER_CHAIN or ALTER_RUNNING_CHAIN. If a step's SKIP attribute is TRUE, then when a chain condition to run that step is met, instead of being run, the step is treated as if it has immediately succeeded. Setting SKIP to TRUE has no effect on a step that is running, that is scheduled to run after a delay, or that has already run.

Skipping steps is especially useful when testing chains. For example, when testing the chain shown in Figure 29–1 on page 29-51, skipping Step 7 could shorten testing time considerably, because this step is a nested chain.

Running Part of a Chain

There are two ways to run only a part of a chain:

- Use the ALTER_CHAIN procedure to set the PAUSE attribute to TRUE for one or more steps, and then either start the chain job with RUN_JOB or start the chain with RUN_CHAIN. Any steps that depend on the paused steps do not run. (However, the paused steps do run.)

  The disadvantage of this method is that you must set the PAUSE attribute back to FALSE for the affected steps for future runs of the chain.

- Use the RUN_CHAIN procedure to start only certain steps of the chain, skipping those steps that you do not want to run.

  This is a more straightforward approach and also enables you to set the initial state of steps before starting them.

You may have to use both of these methods to skip steps both at the beginning and end of a chain.

See the discussion of the RUN_CHAIN procedure in *Oracle Database PL/SQL Packages and Types Reference* for more information.

> **See Also:** "Skipping Chain Steps" on page 29-52

Monitoring Running Chains

You can view the status of running chains with the following two views:

```
*_SCHEDULER_RUNNING_JOBS
*_SCHEDULER_RUNNING_CHAINS
```

The *_SCHEDULER_RUNNING_JOBS views contain one row for the chain job and one row for each running step. The *_SCHEDULER_RUNNING_CHAINS views contain one row for each chain step (including any nested chains) and include run status for each step (NOT_STARTED, RUNNING, STOPPED, SUCCEEDED, and so on).

See *Oracle Database Reference* for details on these views.

Handling Stalled Chains

At the completion of a step, the chain rules are always evaluated to determine the next steps to run. If none of the rules cause another step to start, none cause the chain to end, and the `evaluation_interval` for the chain is `NULL`, the chain enters the **stalled** state. When a chain is stalled, no steps are running, no steps are scheduled to run (after waiting a designated time interval), and no event steps are waiting for an event. The chain can make no further progress unless you manually intervene. In this case, the state of the job that is running the chain is set to `CHAIN_STALLED`. (However, the job is still listed in the `*_SCHEDULER_RUNNING_JOBS` views.)

You can troubleshoot a stalled chain with the views `ALL_SCHEDULER_RUNNING_CHAINS`, which shows the state of all steps in the chain (including any nested chains), and `ALL_SCHEDULER_CHAIN_RULES`, which contains all the chain rules.

You can enable the chain to continue by altering the `state` of one of its steps with the `ALTER_RUNNING_CHAIN` procedure. For example, if step 11 is waiting for step 9 to succeed before it can start, and if it makes sense to do so, you can set the `state` of step 9 to `'SUCCEEDED'`.

Alternatively, if one or more rules are incorrect, you can use the `DEFINE_CHAIN_RULE` procedure to replace them (using the same rule names), or to create new rules. The new and updated rules apply to the running chain and all future chain runs. After adding or updating rules, you must run `EVALUATE_RUNNING_CHAIN` on the stalled chain job to trigger any required actions.

Prioritizing Jobs

You prioritize Oracle Scheduler jobs using three Scheduler objects: job classes, windows, and window groups. These objects prioritize jobs by associating jobs with database resource manager consumer groups. This in turn controls the amount of resources allocated to these jobs. In addition, job classes enable you to set relative priorities among a group of jobs if all jobs in the group are allocated identical resource levels.

This section contains:

- Managing Job Priorities with Job Classes
- Setting Relative Job Priorities Within a Job Class
- Managing Job Scheduling and Job Priorities with Windows
- Managing Job Scheduling and Job Priorities with Window Groups
- Allocating Resources Among Jobs Using Resource Manager
- Example of Resource Allocation for Jobs

> **See Also:** Chapter 27, "Managing Resources with Oracle Database Resource Manager"

Managing Job Priorities with Job Classes

Job classes provide a way to group jobs for prioritization. They also provide a way to easily assign a set of attribute values to member jobs. Job classes influence the priorities of their member jobs through job class attributes that relate to the database resource manager. See "Allocating Resources Among Jobs Using Resource Manager" on page 29-63 for details.

There is a default job class that is created with the database. If you create a job without specifying a job class, the job will be assigned to this default job class (DEFAULT_JOB_CLASS). The default job class has the EXECUTE privilege granted to PUBLIC so any database user who has the privilege to create a job can create a job in the default job class.

This section introduces you to basic job class tasks, and discusses the following topics:

- Job Class Tasks and Their Procedures
- Creating Job Classes
- Altering Job Classes
- Dropping Job Classes

See Also: "Job Classes" on page 28-10 for an overview of job classes.

Job Class Tasks and Their Procedures

Table 29–8 illustrates common job class tasks and their appropriate procedures and privileges:

Table 29–8 Job Class Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a job class	CREATE_JOB_CLASS	MANAGE SCHEDULER
Alter a job class	SET_ATTRIBUTE	MANAGE SCHEDULER
Drop a job class	DROP_JOB_CLASS	MANAGE SCHEDULER

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Job Classes

You create a job class using the CREATE_JOB_CLASS procedure or Enterprise Manager. Job classes are always created in the SYS schema.

The following statement creates a job class for all finance jobs:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group');
END;
/
```

All jobs in this job class are assigned to the finance_group resource consumer group.

To query job classes, use the *_SCHEDULER_JOB_CLASSES views.

See Also: "About Resource Consumer Groups" on page 27-3

Altering Job Classes

You alter a job class by using the SET_ATTRIBUTE procedure or Enterprise Manager. Other than the job class name, all the attributes of a job class can be altered. The attributes of a job class are available in the *_SCHEDULER_JOB_CLASSES views.

When a job class is altered, running jobs that belong to the class are not affected. The change only takes effect for jobs that have not started running yet.

Dropping Job Classes

You drop one or more job classes using the DROP_JOB_CLASS procedure or Enterprise Manager. Dropping a job class means that all the metadata about the job class is removed from the database.

You can drop several job classes in one call by providing a comma-delimited list of job class names to the DROP_JOB_CLASS procedure call. For example, the following statement drops three job classes:

```
BEGIN
  DBMS_SCHEDULER.DROP_JOB_CLASS('jobclass1, jobclass2, jobclass3');
END;
/
```

Setting Relative Job Priorities Within a Job Class

You can change the relative priorities of jobs within the same job class by using the SET_ATTRIBUTE procedure. Job priorities must be in the range of 1-5, where 1 is the highest priority. For example, the following statement changes the job priority for my_job1 to a setting of 1:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'job_priority',
   value          =>   1);
END;
/
```

You can verify that the attribute was changed by issuing the following statement:

```
SELECT JOB_NAME, JOB_PRIORITY FROM DBA_SCHEDULER_JOBS;

JOB_NAME                       JOB_PRIORITY
------------------------------ ------------
MY_EMP_JOB                                3
MY_EMP_JOB1                               1
MY_NEW_JOB1                               3
MY_NEW_JOB2                               3
MY_NEW_JOB3                               3
```

Overall priority of a job within the system is determined first by the combination of the resource consumer group that the job's job class is assigned to and the current resource plan, and then by relative priority within the job class.

See Also:

- "Allocating Resources Among Jobs Using Resource Manager" on page 29-63
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE procedure

Managing Job Scheduling and Job Priorities with Windows

Windows provide a way to automatically activate different resource plans at different times. Running jobs can then see a change in the resources that are allocated to them when there is a change in resource plan. A job can name a window in its schedule_name attribute. The Scheduler then starts the job with the window "opens." A window

has a schedule associated with it, so a window can open at various times during your workload cycle.

The key attributes of a window are its:

- Schedule

  This controls when the window is in effect.

- Duration

  This controls how long the window is open.

- Resource plan

  This names the resource plan that activates when the window opens.

Only one window can be in effect at any given time. Windows belong to the `SYS` schema.

All window activity is logged in the `*_SCHEDULER_WINDOW_LOG` views, otherwise known as the **window logs**. See "Window Log" on page 30-12 for examples of window logging.

This section introduces you to basic window tasks, and discusses the following topics:

- Window Tasks and Their Procedures
- Creating Windows
- Dropping Windows
- Opening Windows
- Closing Windows
- Dropping Windows
- Disabling Windows
- Enabling Windows

See Also: "Windows" on page 28-11 for an overview of windows.

Window Tasks and Their Procedures

Table 29–9 illustrates common window tasks and the procedures you use to handle them.

Table 29–9 Window Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window	`CREATE_WINDOW`	`MANAGE SCHEDULER`
Open a window	`OPEN_WINDOW`	`MANAGE SCHEDULER`
Close a window	`CLOSE_WINDOW`	`MANAGE SCHEDULER`
Alter a window	`SET_ATTRIBUTE`	`MANAGE SCHEDULER`
Drop a window	`DROP_WINDOW`	`MANAGE SCHEDULER`
Disable a window	`DISABLE`	`MANAGE SCHEDULER`
Enable a window	`ENABLE`	`MANAGE SCHEDULER`

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Windows

You can use Enterprise Manager or the DBMS_SCHEDULER.CREATE_WINDOW package procedure to create windows. When using the package procedure, you can leave the resource_plan parameter NULL. In this case, when the window opens, the current plan remains in effect.

You must have the MANAGE SCHEDULER privilege to create windows.

When you specify a schedule for a window, the Scheduler does not check if there is already a window defined for that schedule. Therefore, this may result in windows that overlap. Also, using a named schedule that has a PL/SQL expression as its repeat interval is not supported for windows

See the CREATE_WINDOW procedure in *Oracle Database PL/SQL Packages and Types Reference* for details on window attributes.

The following example creates a window named daytime that enables the mixed_workload_plan resource plan during office hours:

```
BEGIN
   DBMS_SCHEDULER.CREATE_WINDOW (
     window_name      => 'daytime',
     resource_plan    => 'mixed_workload_plan',
     start_date       => '28-APR-09 08.00.00 AM',
     repeat_interval  => 'freq=daily; byday=mon,tue,wed,thu,fri',
     duration         => interval '9' hour,
     window_priority  => 'low',
     comments         => 'OLTP transactions have priority');
END;
/
```

To verify that the window was created properly, query the view DBA_SCHEDULER_WINDOWS. As an example, issue the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN, DURATION, REPEAT_INTERVAL FROM DBA_SCHEDULER_WINDOWS;

WINDOW_NAME   RESOURCE_PLAN         DURATION       REPEAT_INTERVAL
-----------   -------------------   -------------  ---------------
DAYTIME       MIXED_WORKLOAD_PLAN   +000 09:00:00  freq=daily; byday=mon,tue,wed,thu,fri
```

Altering Windows

You alter a window by modifying its attributes. You do so with the SET_ATTRIBUTE and SET_ATTRIBUTE_NULL procedures or Enterprise Manager. With the exception of WINDOW_NAME, all the attributes of a window can be changed when it is altered. See the CREATE_WINDOW procedure in *Oracle Database PL/SQL Packages and Types Reference* for window attribute details.

When a window is altered, it does not affect an active window. The changes only take effect the next time the window opens.

All windows can be altered. If you alter a window that is disabled, it will remain disabled after it is altered. An enabled window will be automatically disabled, altered, and then reenabled, if the validity checks performed during the enable process are successful.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the SET_ATTRIBUTE and SET_ATTRIBUTE_NULL procedures.

Opening Windows

When a window opens, the Scheduler switches to the resource plan that has been associated with it during its creation. If there are jobs running when the window opens, the resources allocated to them might change due to the switch in resource plan.

There are two ways a window can open:

- According to the window's schedule
- Manually, using the `OPEN_WINDOW` procedure

  This procedure opens the window independent of its schedule. This window will open and the resource plan associated with it will take effect immediately. Only an enabled window can be manually opened.

  In the `OPEN_WINDOW` procedure, you can specify the time interval that the window should be open for, using the `duration` attribute. The duration is of type interval day to second. If the duration is not specified, then the window will be opened for the regular duration as stored with the window.

  Opening a window manually has no impact on regular scheduled runs of the window.

  When a window that was manually opened closes, the rules about overlapping windows are applied to determine which other window should be opened at that time if any at all.

  You can force a window to open even if there is one already open by setting the `force` option to `TRUE` in the `OPEN_WINDOW` call or Enterprise Manager.

  When the `force` option is set to `TRUE`, the Scheduler automatically closes any window that is open at that time, even if it has a higher priority. For the duration of this manually opened window, the Scheduler does not open any other scheduled windows even if they have a higher priority. You can open a window that is already open. In this case, the window stays open for the duration specified in the call, from the time the `OPEN_WINDOW` command was issued.

  Consider an example to illustrate this. `window1` was created with a duration of four hours. It has how been open for two hours. If at this point you reopen `window1` using the `OPEN_WINDOW` call and do not specify a duration, then `window1` will be open for another four hours because it was created with that duration. If you specified a duration of 30 minutes, the window will close in 30 minutes.

When a window opens, an entry is made in the window log.

A window can fail to switch resource plans if the current resource plan has been manually switched using the `ALTER SYSTEM` statement with the `FORCE` option, or using the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure with the `allow_scheduler_plan_switches` argument set to `FALSE`. In this case, the failure to switch resource plans is written to the window log.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `OPEN_WINDOW` procedure and the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` procedure.

Closing Windows

There are two ways a window can close:

- Based on a schedule

  A window will close based on the schedule defined at creation time.

- Manually, using the CLOSE_WINDOW procedure

  The CLOSE_WINDOW procedure will close an open window prematurely.

A closed window means that it is no longer in effect. When a window is closed, the Scheduler will switch the resource plan to the one that was in effect outside the window or in the case of overlapping windows to another window. If you try to close a window that does not exist or is not open, an error is generated.

A job that is running will not close when the window it is running in closes unless the attribute stop_on_window_close was set to TRUE when the job was created. However, the resources allocated to the job may change because the resource plan may change.

When a running job has a window group as its schedule, the job will not be stopped when its window is closed if another window that is also a member of the same window group then becomes active. This is the case even if the job was created with the attribute stop_on_window_close set to TRUE.

When a window is closed, an entry will be added to the window log DBA_SCHEDULER_WINDOW_LOG.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the CLOSE_WINDOW procedure.

Dropping Windows

You drop one or more windows using the DROP_WINDOW procedure or Enterprise Manager. When a window is dropped, all metadata about the window is removed from the *_SCHEDULER_WINDOWS views. All references to the window are removed from window groups.

You can drop several windows in one call by providing a comma-delimited list of window names or window group names to the DROP_WINDOW procedure. For example, the following statement drops both windows and window groups:

```
BEGIN
  DBMS_SCHEDULER.DROP_WINDOW ('window1, window2, window3,
   windowgroup1, windowgroup2');
END;
/
```

Note that if a window group name is provided, then the windows in the window group are dropped, but the window group is not dropped. To drop the window group, you must use the DROP_WINDOW_GROUP procedure.

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DROP_WINDOW procedure.

Disabling Windows

You disable one or more windows using the DISABLE procedure or with Enterprise Manager. Therefore, the window will not open. However, the metadata of the window is still there, so it can be reenabled. Because the DISABLE procedure is used for several Scheduler objects, when disabling windows, they must be preceded by SYS.

A window can also become disabled for other reasons. For example, a window will become disabled when it is at the end of its schedule. Also, if a window points to a schedule that no longer exists, it becomes disabled.

If there are jobs that have the window as their schedule, you will not be able to disable the window unless you set force to TRUE in the procedure call. By default, force is set

to FALSE. When the window is disabled, those jobs that have the window as their schedule will not be disabled.

You can disable several windows in one call by providing a comma-delimited list of window names or window group names to the DISABLE procedure call. For example, the following statement disables both windows and window groups:

```
BEGIN
  DBMS_SCHEDULER.DISABLE ('sys.window1, sys.window2,
   sys.window3, sys.windowgroup1, sys.windowgroup2');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the DISABLE procedure.

Enabling Windows

You enable one or more windows using the ENABLE procedure or Enterprise Manager. An enabled window is one that can be opened. Windows are, by default, created enabled. When a window is enabled using the ENABLE procedure, a validity check is performed and only if this is successful will the window be enabled. When a window is enabled, it is logged in the window log table. Because the ENABLE procedure is used for several Scheduler objects, when enabling windows, they must be preceded by SYS.

You can enable several windows in one call by providing a comma-delimited list of window names. For example, the following statement enables three windows:

```
BEGIN
  DBMS_SCHEDULER.ENABLE ('sys.window1, sys.window2, sys.window3');
END;
/
```

See *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the ENABLE procedure.

Managing Job Scheduling and Job Priorities with Window Groups

Window groups provide an easy way to schedule jobs that must run during multiple time periods throughout the day, week, and so on. If you create a window group, add windows to it, and then name this window group in a job's schedule_name attribute, the job runs during all the windows in the window group.

Window groups reside in the SYS schema. This section introduces you to basic window group tasks, and discusses the following topics:

- Window Group Tasks and Their Procedures
- Creating Window Groups
- Dropping Window Groups
- Adding a Member to a Window Group
- Removing a Member from a Window Group
- Enabling a Window Group
- Disabling a Window Group

> **See Also:** "Window Groups" on page 28-14 for an overview of window groups.

Window Group Tasks and Their Procedures

Table 29–10 illustrates common window group tasks and the procedures you use to handle them.

Table 29–10 Window Group Tasks and Their Procedures

Task	Procedure	Privilege Needed
Create a window group	CREATE_GROUP	MANAGE SCHEDULER
Drop a window group	DROP_GROUP	MANAGE SCHEDULER
Add a member to a window group	ADD_GROUP_MEMBER	MANAGE SCHEDULER
Drop a member from a window group	REMOVE_GROUP_MEMBER	MANAGE SCHEDULER
Enable a window group	ENABLE	MANAGE SCHEDULER
Disable a window group	DISABLE	MANAGE SCHEDULER

See "Scheduler Privileges" on page 30-23 for further information regarding privileges.

Creating Window Groups

You create a window group by using the DBMS_SCHEDULER.CREATE_GROUP procedure, specifying a group type of 'WINDOW'. You can specify the member windows of the group when you create the group, or you can add them later using the ADD_GROUP_MEMBER procedure. A window group cannot be a member of another window group. You can, however, create a window group that has no members.

If you create a window group and you specify a member window that does not exist, an error is generated and the window group is not created. If a window is already a member of a window group, it is not added again.

Window groups are created in the SYS schema. Window groups, like windows, are created with access to PUBLIC, therefore, no privileges are required to access window groups.

The following statement creates a window group called downtime and adds two windows (weeknights and weekends) to it:

```
BEGIN
  DBMS_SCHEDULER.CREATE_GROUP (
   group_name   =>  'downtime',
   group_type   =>  'WINDOW',
   member       =>  'weeknights, weekends');
END;
/
```

To verify the window group contents, issue the following queries as a user with the MANAGE SCHEDULER privilege:

```
SELECT group_name, enabled, number_of_members FROM dba_scheduler_groups
  WHERE group_type = 'WINDOW';

GROUP_NAME     ENABLED  NUMBER_OF_MEMBERS
-------------- -------- -----------------
DOWNTIME       TRUE                     2

SELECT group_name, member_name FROM dba_scheduler_group_members;

GROUP_NAME      MEMBER_NAME
--------------- --------------------
```

```
DOWNTIME        "SYS"."WEEKENDS"
DOWNTIME        "SYS"."WEEKNIGHTS"
```

Dropping Window Groups

You drop one or more window groups by using the DROP_GROUP procedure. This call will drop the window group but not the windows that are members of this window group. To drop all the windows that are members of this group but not the window group itself, you can use the DROP_WINDOW procedure and provide the name of the window group to the call.

You can drop several window groups in one call by providing a comma-delimited list of window group names to the DROP_GROUP procedure call. You must precede each window group name with the SYS schema. For example, the following statement drops three window groups:

```
BEGIN
DBMS_SCHEDULER.DROP_GROUP('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

Adding a Member to a Window Group

You add windows to a window group by using the ADD_GROUP_MEMBER procedure.

You can add several members to a window group in one call, by specifying a comma-delimited list of windows. For example, the following statement adds two windows to the window group window_group1:

```
BEGIN
  DBMS_SCHEDULER.ADD_GROUP_MEMBER ('sys.windowgroup1','window2, window3');
END;
/
```

If an already open window is added to a window group, the Scheduler will not start jobs that point to this window group until the next window in the window group opens.

Removing a Member from a Window Group

You can remove one or more windows from a window group by using the REMOVE_GROUP_MEMBER procedure. Jobs with the stop_on_window_close flag set will only be stopped when a window closes. Dropping an open window from a window group has no impact on this.

You can remove several members from a window group in one call by specifying a comma-delimited list of windows. For example, the following statement drops two windows:

```
BEGIN
  DBMS_SCHEDULER.REMOVE_GROUP_MEMBER('sys.window_group1', 'window2, window3');
END;
/
```

Enabling a Window Group

You enable one or more window groups using the ENABLE procedure. By default, window groups are created ENABLED. For example:

```
BEGIN
  DBMS_SCHEDULER.ENABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
```

```
/
```

Disabling a Window Group

You disable a window group using the `DISABLE` procedure. A job with a disabled window group as its schedule does not run when the member windows open. Disabling a window group does not disable its member windows.

You can also disable several window groups in one call by providing a comma-delimited list of window group names. For example, the following statement disables three window groups:

```
BEGIN
  DBMS_SCHEDULER.DISABLE('sys.windowgroup1, sys.windowgroup2, sys.windowgroup3');
END;
/
```

Allocating Resources Among Jobs Using Resource Manager

The Database Resource Manager (Resource Manager) controls how resources are allocated among database sessions. It not only controls asynchronous sessions like Scheduler jobs, but also synchronous sessions like user sessions. It groups all "units of work" in the database into resource consumer groups and uses a resource plan to specify how the resources are allocated among the various consumer groups. The primary system resource that the Resource Manager allocates is CPU.

For Scheduler jobs, resources are allocated by first assigning each job to a job class, and then associating a job class with a consumer group. Resources are then distributed among the Scheduler jobs and other sessions within the consumer group. You can also assign relative priorities to the jobs in a job class, and resources are distributed to those jobs accordingly.

You can manually change the current resource plan at any time. Another way to change the current resource plan is by creating Scheduler windows. Windows have a resource plan attribute. When a window opens, the current plan is switched to the window's resource plan.

The Scheduler tries to limit the number of jobs that are running simultaneously so that at least some jobs can complete, rather than running a lot of jobs concurrently but without enough resources for any of them to complete.

The Scheduler and the Resource Manager are tightly integrated. The job coordinator obtains database resource availability from the Resource Manager. Based on that information, the coordinator determines how many jobs to start. It will only start jobs from those job classes that will have enough resources to run. The coordinator will keep starting jobs in a particular job class that maps to a consumer group until the Resource Manager determines that the maximum resource allocated for that consumer group has been reached. Therefore, there might be jobs in the job table that are ready to run but will not be picked up by the job coordinator because there are no resources to run them. Therefore, there is no guarantee that a job will run at the exact time that it was scheduled. The coordinator picks up jobs from the job table on the basis of which consumer groups still have resources available.

The Resource Manager continues to manage the resources that are assigned to each running job based on the specified resource plan. Keep in mind that the Resource Manager can only manage database processes. The active management of resources does not apply to external jobs.

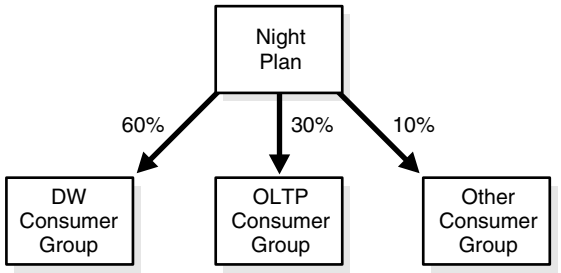
Note: The Resource Manager is active only when CPU utilization approaches 100%.

See Also: Chapter 27, "Managing Resources with Oracle Database Resource Manager"

Example of Resource Allocation for Jobs

The following example illustrates how resources are allocated for jobs. Assume that the active resource plan is called "Night Plan" and that there are three job classes: `JC1`, which maps to consumer group `DW`; `JC2`, which maps to consumer group `OLTP`; and `JC3`, which maps to the default consumer group. Figure 29–2 offers a simple graphical illustration of this scenario.

Figure 29–2 Sample Resource Plan



This resource plan clearly gives priority to jobs that are part of job class `JC1`. Consumer group `DW` gets 60% of the resources, thus jobs that belong to job class `JC1` will get 60% of the resources. Consumer group `OLTP` has 30% of the resources, which implies that jobs in job class `JC2` will get 30% of the resources. The consumer group `Other` specifies that all other consumer groups will be getting 10% of the resources. Therefore, all jobs that belong in job class `JC3` will share 10% of the resources and can get a maximum of 10% of the resources.

Note that resources that remain unused by one consumer group are available from use by the other consumer groups. So if the jobs in job class JC1 do not fully use the allocated 60%, the unused portion is available for use by jobs in classes JC2 and JC3. Note also that the Resource Manager does not begin to restrict resource usage at all until CPU usage reaches 100%. See Chapter 27, "Managing Resources with Oracle Database Resource Manager" for more information.

Monitoring Jobs

There are several ways to monitor Scheduler jobs:

- Viewing the job log

  The job log includes the data dictionary views `*_SCHEDULER_JOB_LOG` and `*_SCHEDULER_JOB_RUN_DETAILS`, where:

  `* = {DBA | ALL | USER}`

  See "Viewing the Job Log" on page 29-65.

- Querying additional data dictionary views

Query views such as DBA_SCHEDULER_RUNNING_JOBS and DBA_SCHEDULER_RUNNING_CHAINS to show the status and details of running jobs and chains.

- Writing applications that receive job state events from the Scheduler

  See "Monitoring Job State with Events Raised by the Scheduler" on page 29-68

- Configuring jobs to send e-mail notifications upon a state change

  See "Monitoring Job State with E-mail Notifications" on page 29-70

Viewing the Job Log

You can view information about job runs, job state changes, and job failures in the job log. The job log shows results for both local and remote jobs. The job log is implemented as the following two data dictionary views:

- *_SCHEDULER_JOB_LOG
- *_SCHEDULER_JOB_RUN_DETAILS

Depending on the logging level that is in effect, the Scheduler can make job log entries whenever a job is run and when a job is created, dropped, enabled, and so on. For a job that has a repeating schedule, the Scheduler makes multiple entries in the job log—one for each job instance. Each log entry provides information about a particular run, such as the job completion status.

The following example shows job log entries for a repeating job that has a value of 4 for the max_runs attribute:

```
SELECT job_name, job_class, operation, status FROM USER_SCHEDULER_JOB_LOG;

JOB_NAME         JOB_CLASS            OPERATION       STATUS
---------------- -------------------- --------------- ----------
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               RUN             SUCCEEDED
JOB1             CLASS1               COMPLETED
```

You can control how frequently information is written to the job log by setting the logging_level attribute of either a job or a job class. Table 29–11 shows the possible values for logging_level.

Table 29–11 Job Logging Levels

Logging Level	Description
DBMS_SCHEDULER.LOGGING_OFF	No logging is performed.
DBMS_SCHEDULER.LOGGING_FAILED_RUNS	A log entry is made only if the job fails.
DBMS_SCHEDULER.LOGGING_RUNS	A log entry is made each time the job is run.
DBMS_SCHEDULER.LOGGING_FULL	A log entry is made every time the job runs and for every operation performed on a job, including create, enable/disable, update (with SET_ATTRIBUTE), stop, and drop.

Log entries for job runs are not made until after the job run completes successfully, fails, or is stopped.

The following example shows job log entries for a complete job lifecycle. In this case, the logging level for the job class is LOGGING_FULL, and the job is a non-repeating job.

After the first successful run, the job is enabled again, so it runs once more. It is then stopped and dropped.

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MI:SS') TIMESTAMP, job_name,
  job_class, operation, status FROM USER_SCHEDULER_JOB_LOG
  WHERE job_name = 'JOB2' ORDER BY log_date;

TIMESTAMP            JOB_NAME  JOB_CLASS  OPERATION  STATUS
-------------------- --------- ---------- ---------- ---------
18-DEC-07 23:10:56   JOB2      CLASS1     CREATE
18-DEC-07 23:12:01   JOB2      CLASS1     UPDATE
18-DEC-07 23:12:31   JOB2      CLASS1     ENABLE
18-DEC-07 23:12:41   JOB2      CLASS1     RUN        SUCCEEDED
18-DEC-07 23:13:12   JOB2      CLASS1     ENABLE
18-DEC-07 23:13:18   JOB2                 RUN        STOPPED
18-DEC-07 23:19:36   JOB2      CLASS1     DROP
```

Run Details

For every row in `*_SCHEDULER_JOB_LOG` for which the operation is `RUN`, `RETRY_RUN`, or `RECOVERY_RUN`, there is a corresponding row in the `*_SCHEDULER_JOB_RUN_DETAILS` view. Rows from the two different views are correlated with their `LOG_ID` columns. You can consult the run details views to determine why a job failed or was stopped.

```
SELECT to_char(log_date, 'DD-MON-YY HH24:MI:SS') TIMESTAMP, job_name, status,
   SUBSTR(additional_info, 1, 40) ADDITIONAL_INFO
   FROM user_scheduler_job_run_details ORDER BY log_date;

TIMESTAMP            JOB_NAME   STATUS    ADDITIONAL_INFO
-------------------- ---------- --------- ----------------------------------------
18-DEC-07 23:12:41   JOB2       SUCCEEDED
18-DEC-07 23:12:18   JOB2       STOPPED   REASON="Stop job called by user:'SYSTEM'
19-DEC-07 14:12:20   REMOTE_16  FAILED    ORA-29273: HTTP request failed ORA-06512
```

The run details views also contain actual job start times and durations.

Precedence of Logging Levels in Jobs and Job Classes

Both jobs and job classes have a `logging_level` attribute, with possible values listed in Table 29–11 on page 29-65. The default logging level for job classes is `LOGGING_RUNS`, and the default level for individual jobs is `LOGGING_OFF`. If the logging level of the job class is higher than that of a job in the class, then the logging level of the job class takes precedence. Thus, by default, all job runs are recorded in the job log.

For job classes that have very short and highly frequent jobs, the overhead of recording every single run might be too much and you might choose to turn the logging off or set logging to occur only when jobs fail. However, you might prefer to have a complete audit trail of everything that happens with jobs in a specific class, in which case you would enable full logging for that class.

To ensure that there is an audit trail for all jobs, the individual job creator must not be able to turn logging off. The Scheduler supports this by making the class-specified level the minimum level at which job information is logged. A job creator can only enable more logging for an individual job, not less. Thus, leaving all individual job logging levels set to `LOGGING_OFF` ensures that all jobs in a class get logged as specified in the class.

This functionality is provided for debugging purposes. For example, if the class-specific level is set to record job runs and logging is turned off at the job level, the Scheduler still logs job runs. If, however, the job creator turns on full logging and the class-specific level is set to record runs only, the higher logging level of the job takes

precedence and all operations on this individual job are logged. This way, an end user can test his job by turning on full logging.

To set the logging level of an individual job, you must use the SET_ATTRIBUTE procedure on that job. For example, to turn on full logging for a job called mytestjob, issue the following statement:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'mytestjob', 'logging_level', DBMS_SCHEDULER.LOGGING_FULL);
END;
/
```

Only a user with the MANAGE SCHEDULER privilege can set the logging level of a job class.

> **See Also:** "Monitoring and Managing Window and Job Logs" on page 30-11 for more information about setting the job class logging level

Monitoring Multiple Destination Jobs

For multiple-destination jobs, the overall parent job state depends on the outcome of the child jobs. For example, if all child jobs succeed, the parent job state is set to SUCCEEDED. If all fail, the parent job state is set to FAILED. If some fail and some succeed, the parent job state is set to SOME FAILED.

Due to situations that might arise on some destinations that delay the start of child jobs, there might be a significant delay before the parent job state is finalized. For repeating multiple-destination jobs, there might even be a situation in which some child jobs are on their next scheduled run while others are still working on the previous scheduled run. In this case, the parent job state is set to INCOMPLETE. Eventually, however, lagging child jobs may catch up to their siblings, in which case the final state of the parent job can be determined.

Table Table 29–12 lists the contents of the job monitoring views for multiple-destination jobs.

Table 29–12 Scheduler Data Dictionary View Contents for Multiple-Destination Jobs

View Name	Contents
*_SCHEDULER_JOBS	One entry for the parent job
*_SCHEDULER_RUNNING_JOBS	One entry for the parent job when it starts and an entry for each running child job
*_SCHEDULER_JOB_LOG	One entry for the parent job when it starts (operation = 'MULTIDEST_START'), one entry for each child job when the child job completes, and one entry for the parent job when the last child job completes and thus the parent completes (operation = 'MULTIDEST_RUN')
*_SCHEDULER_JOB_RUN_DETAILS	One entry for each child job when the child job completes, and one entry for the parent job when the last child job completes and thus the parent completes
*_SCHEDULER_JOB_DESTS	One entry for each destination of the parent job

In the *_SCHEDULER_JOB_DESTS views, you can determine the unique job destination ID (job_dest_id) that is assigned to each child job. This ID represents the unique combination of a job, a credential, and a destination. You can use this ID with the

STOP_JOB procedure. You can also monitor the job state of each child job with the *_SCHEDULER_JOB_DESTS views.

See Also:

- "Multiple-Destination Jobs" on page 28-18
- "Creating Multiple-Destination Jobs" on page 29-9
- "Scheduler Data Dictionary Views" on page 30-24

Monitoring Job State with Events Raised by the Scheduler

This section contains:

- About Job State Events
- Altering a Job to Raise Job State Events
- Consuming Job State Events with your Application

About Job State Events

You can configure a job so that the Scheduler raises an event when the job changes state. The Scheduler can raise an event when a job starts, when a job completes, when a job exceeds its allotted run time, and so on. The consumer of the event is your application, which takes some action in response to the event. For example, if due to a high system load, a job is still not started 30 minutes after its scheduled start time, the Scheduler can raise an event that causes a handler application to stop lower priority jobs to free up system resources. The Scheduler can raise job state events for local (regular) jobs, remote database jobs, local external jobs, and remote external jobs.

Table 29–13 describes the job state event types raised by the Scheduler.

Table 29–13 Job State Event Types Raised by the Scheduler

Event Type	Description
job_all_events	Not an event, but a constant that provides an easy way for you to enable all events
job_broken	The job has been disabled and has changed to the BROKEN state because it exceeded the number of failures defined by the max_failures job attribute
job_chain_stalled	A job running a chain was put into the CHAIN_STALLED state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain evaluation_interval is set to NULL. No progress will be made in the chain unless there is manual intervention.
job_completed	The job completed because it reached its max_runs or end_date
job_disabled	The job was disabled by the Scheduler or by a call to SET_ATTRIBUTE
job_failed	The job failed, either by throwing an error or by abnormally terminating
job_over_max_dur	The job exceeded the maximum run duration specified by its max_run_duration attribute.
job_run_completed	A job run either failed, succeeded, or was stopped
job_sch_lim_reached	The job's schedule limit was reached. The job was not started because the delay in starting the job exceeded the value of the schedule_limit job attribute.

Table 29–13 (Cont.) Job State Event Types Raised by the Scheduler

Event Type	Description
job_started	The job started
job_stopped	The job was stopped by a call to STOP_JOB
job_succeeded	The job completed successfully

You enable the raising of job state events by setting the raise_events job attribute. By default, a job does not raise any job state events.

The Scheduler uses Oracle Streams Advanced Queuing to raise events. When raising a job state change event, the Scheduler enqueues a message onto a default event queue. Your applications subscribe to this queue, dequeue event messages, and take appropriate actions.

After you enable job state change events for a job, the Scheduler raises these events by enqueuing messages onto the Scheduler event queue SYS.SCHEDULER$_EVENT_QUEUE. This queue is a secure queue, so depending on your application, you may have to configure the queue to enable certain users to perform operations on it. See *Oracle Streams Concepts and Administration* for information on secure queues.

To prevent unlimited growth of the Scheduler event queue, events raised by the Scheduler expire in 24 hours by default. (Expired events are deleted from the queue.) You can change this expiry time by setting the event_expiry_time Scheduler attribute with the SET_SCHEDULER_ATTRIBUTE procedure. See *Oracle Database PL/SQL Packages and Types Reference* for more information.

Altering a Job to Raise Job State Events

To enable job state events to be raised for a job, you use the SET_ATTRIBUTE procedure to turn on bit flags in the raise_events job attribute. Each bit flag represents a different job state to raise an event for. For example, turning on the least significant bit enables job started events to be raised. To enable multiple state change event types in one call, you add the desired bit flag values together and supply the result as an argument to SET_ATTRIBUTE.

The following example enables multiple state change events for job dw_reports. It enables the following event types, both of which indicate some kind of error.

- JOB_FAILED
- JOB_SCH_LIM_REACHED

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE('dw_reports', 'raise_events',
   DBMS_SCHEDULER.JOB_FAILED + DBMS_SCHEDULER.JOB_SCH_LIM_REACHED);
END;
/
```

Note: You do not need to enable the JOB_OVER_MAX_DUR event with the raise_events job attribute; it is always enabled.

See Also: The discussion of DBMS_SCHEDULER.SET_ATTRIBUTE in *Oracle Database PL/SQL Packages and Types Reference* for the names and values of job state bit flags

Consuming Job State Events with your Application

To consume job state events, your application must subscribe to the Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE`. This queue is a secure queue and is owned by `SYS`. To create a subscription to this queue for a user, do the following:

1. Log in to the database as the `SYS` user or as a user with the `MANAGE ANY QUEUE` privilege.
2. Subscribe to the queue using a new or existing agent.
3. Run the package procedure `DBMS_AQADM.ENABLE_DB_ACCESS` as follows:

   ```
   DBMS_AQADM.ENABLE_DB_ACCESS(agent_name, db_username);
   ```

   where *agent_name* references the agent that you used to subscribe to the events queue, and *db_username* is the user for whom you want to create a subscription.

There is no need to grant dequeue privileges to the user. The dequeue privilege is granted on the Scheduler event queue to `PUBLIC`.

As an alternative, the user can subscribe to the Scheduler event queue using the `ADD_EVENT_QUEUE_SUBSCRIBER` procedure, as shown in the following example:

```
DBMS_SCHEDULER.ADD_EVENT_QUEUE_SUBSCRIBER(subscriber_name);
```

where *subscriber_name* is the name of the Oracle Streams Advanced Queuing (AQ) agent to be used to subscribe to the Scheduler event queue. (If it is `NULL`, an agent is created whose name is the user name of the calling user.) This call both creates a subscription to the Scheduler event queue and grants the user permission to dequeue using the designated agent. The subscription is rule-based. The rule permits the user to see only events raised by jobs that the user owns, and filters out all other messages. After the subscription is in place, the user can either poll for messages at regular intervals or register with AQ for notification.

See *Oracle Database Advanced Queuing User's Guide* for more information.

Scheduler Event Queue

The Scheduler event queue `SYS.SCHEDULER$_EVENT_QUEUE` is of type `scheduler$_event_info`. See *Oracle Database PL/SQL Packages and Types Reference* for details on this type.

Monitoring Job State with E-mail Notifications

This section contains:

- About E-mail Notifications
- Adding E-mail Notifications for a Job
- Removing E-mail Notifications for a Job
- Viewing Information About E-mail Notifications

About E-mail Notifications

You can configure a job to send e-mail notifications when it changes state. The job state events for which e-mails can be sent are listed in Table 29–13 on page 29-68. E-mail notifications can be sent to multiple recipients, and can be triggered by any event in a list of job state events that you specify. You can also provide a filter condition, and only job state events that match the filter condition generate notifications. You can include variables like job owner, job name, event type, error code, and error message

in both the subject and body of the message. The Scheduler automatically sets values for these variables before sending the e-mail notification.

You can configure many job state e-mail notifications for a single job. The notifications can differ by job state event list, recipients, and filter conditions.

For example, you can configure a job to send an e-mail to both the principle DBA and one of the senior DBAs whenever the job fails with error code 600 or 700. You can also configure the same job to send a notification to only the principle DBA if the job fails to start at its scheduled time.

Before you can configure jobs to send e-mail notifications, you must set the Scheduler attribute `email_server` to the address of the SMTP server to use to send the e-mail. You may also optionally set the Scheduler attribute `email_sender` to a default sender e-mail address for those jobs that do not specify a sender.

The Scheduler includes support for the SSL and TLS protocols when communicating with the SMTP server. The Scheduler also supports SMTP servers that require authentication.

> **See Also:** "Setting Scheduler Preferences" on page 30-2 for details about setting e-mail notification–related attributes

Adding E-mail Notifications for a Job

You use the `DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION` package procedure to add e-mail notifications for a job.

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name   =>  'EOD_JOB',
  recipients =>  'jsmith@example.com, rjones@example.com',
  sender     =>  'do_not_reply@example.com',
  subject    =>  'Scheduler Job Notification-%job_owner%.%job_name%-%event_type%',
  body       =>   '%event_type% occurred at %event_timestamp%. %error_message%',
  events     =>  'JOB_FAILED, JOB_BROKEN, JOB_DISABLED, JOB_SCH_LIM_REACHED');
END;
/
```

Note the variables, enclosed in the '%' character, used in the `subject` and `body` arguments. When you specify multiple recipients and multiple events, each recipient is notified when any of the specified events is raised. You can verify this by querying the view `USER_SCHEDULER_NOTIFICATIONS`.

```
SELECT JOB_NAME, RECIPIENT, EVENT FROM USER_SCHEDULER_NOTIFICATIONS;

JOB_NAME    RECIPIENT            EVENT
----------- -------------------- -------------------
EOD_JOB     jsmith@example.com   JOB_FAILED
EOD_JOB     jsmith@example.com   JOB_BROKEN
EOD_JOB     jsmith@example.com   JOB_SCH_LIM_REACHED
EOD_JOB     jsmith@example.com   JOB_DISABLED
EOD_JOB     rjones@example.com   JOB_FAILED
EOD_JOB     rjones@example.com   JOB_BROKEN
EOD_JOB     rjones@example.com   JOB_SCH_LIM_REACHED
EOD_JOB     rjones@example.com   JOB_DISABLED
```

You call `ADD_JOB_EMAIL_NOTIFICATION` once for each different set of notifications that you want to configure for a job. You must specify `job_name` and `recipients`. All other arguments have defaults. The default `sender` is defined by a Scheduler attribute, as described in the previous section. See the `ADD_JOB_EMAIL_NOTIFICATION` procedure in

Oracle Database PL/SQL Packages and Types Reference for defaults for the `subject`, `body`, and `events` arguments.

The following example configures an additional e-mail notification for the same job for a different event. This example accepts the defaults for the `sender`, `subject`, and `body` arguments.

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name         =>  'EOD_JOB',
  recipients       =>  'jsmith@example.com',
  events           =>  'JOB_OVER_MAX_DUR');
END;
/
```

This example could have also omitted the `events` argument to accept event defaults.

The next example is similar to the first, except that it uses a filter condition to specify that an e-mail notification is to be sent only when the error number that causes the job to fail is 600 or 700.

```
BEGIN
 DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name         => 'EOD_JOB',
  recipients       => 'jsmith@example.com, rjones@example.com',
  sender           => 'do_not_reply@example.com',
  subject          => 'Job Notification-%job_owner%.%job_name%-%event_type%',
  body             =>  '%event_type% at %event_timestamp%. %error_message%',
  events           => 'JOB_FAILED',
  filter_condition => ':event.error_code=600 or :event.error_code=700');
END;
/
```

> **See Also:** The `ADD_JOB_EMAIL_NOTIFICATION` procedure in *Oracle Database PL/SQL Packages and Types Reference*

Removing E-mail Notifications for a Job

You use the `DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION` package procedure to remove e-mail notifications for a job.

```
BEGIN
 DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (
  job_name   =>  'EOD_JOB',
  recipients =>  'jsmith@example.com, rjones@example.com',
  events     =>  'JOB_DISABLED, JOB_SCH_LIM_REACHED');
END;
/
```

When you specify multiple recipients and multiple events, the notification for each specified event is removed for each recipient. Running the same query as that of the previous section, the results are now the following:

```
SELECT JOB_NAME, RECIPIENT, EVENT FROM USER_SCHEDULER_NOTIFICATIONS;

JOB_NAME    RECIPIENT            EVENT
----------- -------------------- --------------------
EOD_JOB     jsmith@example.com   JOB_FAILED
EOD_JOB     jsmith@example.com   JOB_BROKEN
EOD_JOB     rjones@example.com   JOB_FAILED
EOD_JOB     rjones@example.com   JOB_BROKEN
```

Additional rules for specifying `REMOVE_JOB_EMAIL_NOTIFICATION` arguments are as follows:

- If you leave the `events` argument `NULL`, notifications for all events for the specified recipients are removed.
- If you leave `recipients` `NULL`, notifications for all recipients for the specified events are removed.
- If you leave both `recipients` and `events` `NULL`, then all notifications for the job are removed.
- If you include a recipient and event for which you did not previously create a notification, no error is generated.

> **See Also:** The `REMOVE_JOB_EMAIL_NOTIFICATION` procedure in *Oracle Database PL/SQL Packages and Types Reference*

Viewing Information About E-mail Notifications

As demonstrated in the previous sections, you can view information about current e-mail notifications by querying the views `*_SCHEDULER_NOTIFICATIONS`.

> **See Also:** *Oracle Database Reference* for details on these views

30

Administering Oracle Scheduler

In this chapter:

- Configuring Oracle Scheduler
- Monitoring and Managing the Scheduler
- Import/Export and the Scheduler
- Troubleshooting the Scheduler
- Examples of Using the Scheduler
- Scheduler Reference

Note: This chapter describes how to use the `DBMS_SCHEDULER` package to administer Oracle Scheduler. You can accomplish many of the same tasks using Oracle Enterprise Manager.

See *Oracle Database PL/SQL Packages and Types Reference* for `DBMS_SCHEDULER` information and the Oracle Enterprise Manager online help for information on Oracle Scheduler pages.

Configuring Oracle Scheduler

This section contains:

- Setting Oracle Scheduler Privileges
- Setting Scheduler Preferences
- Using the Oracle Scheduler Agent to Run Remote Jobs

Setting Oracle Scheduler Privileges

You must have the `SCHEDULER_ADMIN` role to perform all Oracle Scheduler administration tasks. Typically, database administrators already have this role with the `ADMIN` option as part of the `DBA` role. For example, users `SYS` and `SYSTEM` are granted the `DBA` role. You can grant this role to another administrator by issuing the following statement:

```
GRANT SCHEDULER_ADMIN TO username;
```

Because the `SCHEDULER_ADMIN` role is a powerful role allowing a grantee to execute code as any user, you should consider granting individual Scheduler system privileges instead. Object and system privileges are granted using regular SQL grant syntax. An example is if the database administrator issues the following statement:

```
GRANT CREATE JOB TO scott;
```

After this statement is executed, `scott` can create jobs, schedules, programs, file watchers, and credentials in his schema. Another example is if the database administrator issues the following statement:

```
GRANT MANAGE SCHEDULER TO adam;
```

After this statement is executed, `adam` can create, alter, or drop windows, job classes, or window groups. He will also be able to set and retrieve Scheduler attributes and purge Scheduler logs.

Setting Chain Privileges

Scheduler chains use underlying Oracle Streams Rules Engine objects along with their associated privileges. To create a chain in their own schema, users must have the `CREATE JOB` privilege in addition to the Rules Engine privileges required to create rules, rule sets, and evaluation contexts in their own schema. These can be granted by issuing the following statement:

```
GRANT CREATE RULE, CREATE RULE SET, CREATE EVALUATION CONTEXT TO user;
```

To create a chain in a different schema, users must have the `CREATE ANY JOB` privilege in addition to the privileges required to create rules, rule sets, and evaluation contexts in schemas other than their own. These can be granted by issuing the following statement:

```
GRANT CREATE ANY RULE, CREATE ANY RULE SET,
   CREATE ANY EVALUATION CONTEXT TO user;
```

Altering or dropping chains in schemas other than the users's schema require corresponding system Rules Engine privileges for rules, rule sets, and evaluation contexts.

> **See Also:** "Chain Tasks and Their Procedures" on page 29-42 for more information regarding chain privileges.

Setting Scheduler Preferences

There are several systemwide Scheduler preferences that you can set. You set these preferences by setting Scheduler attributes with the `SET_SCHEDULER_ATTRIBUTE` procedure. Setting these attributes requires the `MANAGE SCHEDULER` privilege. The attributes are:

- `default_timezone`

  It is very important that you set this attribute. Repeating jobs and windows that use the calendaring syntax need to know which time zone to use for their repeat intervals. See "Using the Scheduler Calendaring Syntax" on page 29-26. They normally retrieve the time zone from `start_date`, but if no `start_date` is provided (which is not uncommon), they retrieve the time zone from the `default_timezone` Scheduler attribute.

  The Scheduler derives the value of `default_timezone` from the operating system environment. If the Scheduler can find no compatible value from the operating system, it sets `default_timezone` to `NULL`.

  It is crucial that you verify that `default_timezone` is set properly, and if not, that you set it. To verify it, run this query:

  ```
  SELECT DBMS_SCHEDULER.STIME FROM DUAL;
  ```

```
STIME
-------------------------------------------------------------------------------
14-OCT-04 02.56.03.206273000 PM US/PACIFIC
```

To ensure that daylight savings adjustments are followed, it is recommended that you set `default_timezone` to a region name instead of an absolute time zone offset like '-8:00'. For example, if your database resides in Miami, Florida, USA, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('default_timezone','US/Eastern');
```

Similarly, if your database resides in Paris, you would set this attribute to `'Europe/Warsaw'`. To see a list of valid region names, run this query:

```
SELECT DISTINCT TZNAME FROM V$TIMEZONE_NAMES;
```

If you do not properly set `default_timezone`, the default time zone for repeating jobs and windows will be the absolute offset retrieved from `SYSTIMESTAMP` (the time zone of the operating system environment of the database), which means that repeating jobs and windows that do not have their `start_date` set will not follow daylight savings adjustments.

- `email_server`

  This attribute specifies an SMTP server address that the Scheduler uses to send e-mail notifications for job state events. It takes the following format:

  host[:*port*]

  where:

  - *host* is the host name or IP address of the SMTP server.
  - *port* is the TCP port on which the SMTP server listens. If not specified, the default port of 25 is used.

  If this attribute is not specified, set to `NULL`, or set to an invalid SMTP server address, the Scheduler cannot send job state e-mail notifications.

- `email_sender`

  This attribute specifies the default e-mail address of the sender for job state e-mail notifications. It must be a valid e-mail address. If this attribute is not set or set to `NULL`, then job state e-mail notifications that do not specify a sender address do not have a FROM address in the e-mail header.

- `email_server_credential`

  This attribute specifies the schema and name of an existing credential object. The default is `NULL`.

  When an e-mail notification goes out, the Scheduler determines if the `email_server_credential` points to a valid credential object that `SYS` has execute object privileges on. If the SMTP server specified in the `email_server` attribute requires authentication, then the Scheduler uses the user name and password stored in the specified credential object to authenticate with the e-mail server.

  If the `email_server_credential` is specified, then the `email_server` attribute must specify an SMTP server that requires authentication.

  If the `email_server_credential` is not specified, then the Scheduler supports sending notification e-mails through an SMTP server for which authentication is not configured.

> **Note:** This functionality is available with Oracle Database 11*g* release 2 (11.2.0.2).

- `email_server_encryption`

  This attribute indicates whether encryption is enabled for this SMTP server connection, and if so, at what point encryption starts, and with which protocol.

  Values for `email_server_encryption` are:

  `NONE`: The default, indicates no encryption.

  `SSL_TLS`: Indicates that either `SSL` or `TLS` are used, from the beginning of the connection. The two sides determine which protocol is most secure. This is the most common setting for this parameter.

  `STARTTLS`: Indicates that the connection starts in an unencrypted state, but then the command `STARTTLS` directs the e-mail server to start encryption using `TLS`.

  > **Note:** This functionality is available with Oracle Database 11*g* release 2 (11.2.0.2).

- `event_expiry_time`

  This attribute enables you to set the time in seconds before a job state event generated by the Scheduler expires (is automatically purged from the Scheduler event queue). If `NULL`, job state events expire after 24 hours.

- `log_history`

  This attribute controls the number of days that log entries for both the job log and the window log are retained. It helps prevent logs from growing indiscriminately. The range of valid values is 0 through 1000000. If set to 0, no history is kept. Default value is 30. You can override this value at the job class level by setting a value for the `log_history` attribute of the job class.

See *Oracle Database PL/SQL Packages and Types Reference* for the syntax for the `SET_SCHEDULER_ATTRIBUTE` procedure.

Using the Oracle Scheduler Agent to Run Remote Jobs

Using the Oracle Scheduler agent, the Scheduler can schedule and run two types of remote jobs:

- Remote database jobs: Remote database jobs must be run through an Oracle Scheduler agent. Oracle recommends that an agent be installed on the same host as the remote database.

  If you intend to run remote database jobs, the Scheduler agent must be release 11.2 or later.

- Remote external jobs: Remote external jobs run on the same host that the Scheduler agent is installed on.

  If you intend to run only remote external jobs, release 11.1 of the Scheduler agent is sufficient.

You must install Scheduler agents on all hosts that remote external jobs will run on. You should install Scheduler agents on all hosts running remote databases that remote database jobs will be run on.

Each database that runs remote jobs requires an initial setup to enable secure communications between databases and remote Scheduler agents, as described in "Setting up Databases for Remote Jobs" on page 30-5.

Enabling remote jobs involves the following steps:

1. Enabling and Disabling Databases for Remote Jobs
2. Installing and Configuring the Scheduler Agent on a Remote Host
3. Performing Tasks with the Scheduler Agent

See Also:

- "About Remote External Jobs" on page 28-18
- "Database Jobs" on page 28-15 for more information on remote database jobs

Enabling and Disabling Databases for Remote Jobs

This section covers these topics:

- Setting up Databases for Remote Jobs
- Disabling Remote Jobs

Setting up Databases for Remote Jobs Before a database can run jobs using a remote Scheduler agent, the database must be properly configured, and the agent must be registered with the database. This section describes the configuration, including the required agent registration password in the database. You will later register the database, as shown in "Registering Scheduler Agents with Databases" on page 30-9.

You can limit the number of Scheduler agents that can register, and you can set the password to expire after a specified duration.

Complete the following steps once for each database that creates and runs remote jobs.

To set up a database to create and run remote jobs:

1. Ensure that shared server is enabled.

   See "Enabling Shared Server" on page 5-7.

2. Using SQL*Plus, connect to the database as the `SYS` user.
3. Enter the following command to verify that the XML DB option is installed:

   ```
   SQL> DESC RESOURCE_VIEW
   ```

   If XML DB is not installed, this command returns an "object does not exist" error.

   > **Note:** If XML DB is not installed, you must install it before continuing.

4. Enable HTTP connections to the database as follows:

   a. Determine whether or not the Oracle XML DBM HTTP Server is enabled:

      Issue the following command:

```
SQL> SELECT DBMS_XDB.GETHTTPPORT() FROM DUAL;
```

If this statement returns `0`, Oracle XML DBM HTTP Server is disabled.

b. Enable Oracle XML DB HTTP Server on a nonzero port by logging in as `SYS` and issuing the following commands:

```
SQL> EXEC DBMS_XDB.SETHTTPPORT (port);
SQL> COMMIT;
```

where *`port`* is the TCP port number on which you want the database to listen for HTTP connections.

`port` must be an integer between 1 and 65536, and for UNIX and Linux must be greater than 1023. Choose a port number that is not already in use.

Note: This enables HTTP connections on all instances of an Oracle Real Application Clusters database.

---

5. Run the script `prvtrsch.plb` with following command:

```
SQL> @?/rdbms/admin/prvtrsch.plb
```

6. Set a registration password for the Scheduler agents using the `SET_AGENT_REGISTRATION_PASS` procedure.

The following example sets the agent registration password to `mypassword`.

```
BEGIN
  DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('mypassword');
END;
/
```

Note: You must have the `MANAGE SCHEDULER` privilege to set an agent registration password. See *Oracle Database PL/SQL Packages and Types Reference* for more information on the `SET_AGENT_REGISTRATION_PASS` procedure.

You will do the actual registration further on, in "Registering Scheduler Agents with Databases" on page 30-9.

Disabling Remote Jobs You can disable remote jobs on a database by dropping the `REMOTE_SCHEDULER_AGENT` user.

To disable remote jobs:

- Submit the following SQL statement:

```
DROP USER REMOTE_SCHEDULER_AGENT CASCADE;
```

Registration of new scheduler agents and execution of remote jobs is disabled until you run `prvtrsch.plb` again.

Installing and Configuring the Scheduler Agent on a Remote Host

Before you can run remote jobs on a particular host, you must install and configure the Scheduler agent, described in this section, and then register and start the Scheduler agent on the host, described in "Performing Tasks with the Scheduler Agent" on page 30-8. The Scheduler agent must also be installed in its own Oracle home.

To install and configure the Scheduler agent on a remote host:

1. Download or retrieve the Scheduler agent software, which is available on the Oracle Database Client media included in the Database Media Pack, and online at:

   http://www.oracle.com/technology/software/products/database

2. Ensure that you have first properly set up any database on which you want to register the agent.

   See "Enabling and Disabling Databases for Remote Jobs" on page 30-5 for instructions.

3. Log in to the host you want to install the Scheduler agent on. This host runs remote jobs.

   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user that you want the Scheduler agent to run as. This user requires no special privileges.

4. Run the Oracle Universal Installer (OUI) from the installation media for Oracle Database Client.

   - For Windows, run `setup.exe`.
   - For UNIX and Linux, use the following command:

     ```
     /directory_path/runInstaller
     ```

   where *`directory_path`* is the path to the Oracle Database Client installation media.

5. On the Select Installation Type page, select **Custom**, and then click **Next**.

6. On the Select Product Languages page, select the desired languages, and click **Next**.

7. On the Specify Install Location page, enter the path for a new Oracle home for the agent, and then click **Next**.

8. On the Available Product Components page, select **Oracle Scheduler Agent**, and click **Next**.

9. On the Oracle Database Scheduler Agent page:

   a. In the Scheduler Agent Hostname field, enter the host name of the computer that the Scheduler agent is installed on.

   b. In the Scheduler Agent Port Number field, enter the TCP port number that the Scheduler agent is to listen on for connections, or accept the default, and then click **Next**.

      Choose an integer between 1 and 65535. On UNIX and Linux, the number must be greater than 1023. Ensure that the port number is not already in use.

      OUI performs a series of prerequisite checks. If any of the prerequisite checks fail, resolve the problems, and then click **Next**.

10. On the Summary page, click **Finish**.

11. (UNIX and Linux only) When OUI prompts you to run the script `root.sh`, enter the following command as the `root` user:

    ```
    script_path/root.sh
    ```

    The script is located in the directory that you chose for agent installation.

When the script completes, click **OK** in the Execute Configuration Scripts dialog box.

12. Click **Close** to exit OUI when installation is complete.
13. Use a text editor to review the agent configuration parameter file `schagent.conf`, which is located in the Scheduler agent home directory, and verify the port number in the `PORT=` directive.
14. Ensure that any firewall software on the remote host or any other firewall that protects that host has an exception to accommodate the Scheduler agent.

Performing Tasks with the Scheduler Agent

The Scheduler agent is a standalone program that enables you to schedule and run external and database jobs on remote hosts. You start and stop the Scheduler agent using the `schagent` utility on UNIX and Linux, and the `OracleSchedulerExecutionAgent` service on Windows.

This section covers these topics:

- About the schagent utility
- Using the Scheduler Agent on Windows
- Starting the Scheduler Agent
- Stopping the Scheduler Agent
- Registering Scheduler Agents with Databases

About the schagent utility The executable utility `schagent` performs certain tasks for the agent on Windows, UNIX and Linux, as indicated by the options in Table 30–1.

Use `schagent` with the appropriate syntax and options as follows:

For example:

UNIX and Linux: *AGENT_HOME*`/bin/schagent -status`

Windows: *AGENT_HOME*`/bin/schagent.exe -status`

Table 30–1 schagent options

Option	Description
`-start`	Starts the Scheduler Agent. *UNIX and Linux only*
`-stop`	Prompts the Scheduler agent to stop all the currently running jobs and then stop execution gracefully. *UNIX and Linux only*
`-abort`	Stops the Scheduler agent forcefully, that is, without stopping jobs first. From Oracle Database Release 11.2. *UNIX and Linux only*
`-status`	Returns this information about the Scheduler Agent running locally: version, uptime, total number of jobs run since the agent started, number of jobs currently running, and their descriptions.
`-registerdatabase`	Register the Scheduler agent with the base database or additional databases that are to run remote jobs on the agent's host computer.

Table 30–1 (Cont.) schagent options

Option	Description
`-unregisterdatabase`	Unregister an agent from a database.

Using the Scheduler Agent on Windows The Windows Scheduler agent service is automatically created and started during installation. The name of the service ends with `OracleSchedulerExecutionAgent`.

> **Note:** Do not confuse this service with the `OracleJobScheduler` service, which runs on a Windows computer on which an Oracle database is installed, and manages the running of local external jobs without credentials.

Starting the Scheduler Agent Start the Scheduler agent with the following command:

To start the Scheduler agent:

- Do one of the following:
  - On UNIX and Linux, run the following command:

    ```
    AGENT_HOME/bin/schagent -start
    ```

  - On Windows, start the service whose name ends with `OracleSchedulerExecutionAgent`.

Stopping the Scheduler Agent Stopping the Scheduler agent prevents the host on which it resides from running remote jobs.

To stop the Scheduler agent:

- Do one of the following:
  - On UNIX and Linux, run the `schagent` utility with either the `-stop` or `-abort` option as described in Table 30–1:

    ```
    AGENT_HOME/bin/schagent -stop
    ```

  - On Windows, stop the service whose name ends with `OracleSchedulerExecutionAgent`. This is equivalent to the `-abort` option.

Registering Scheduler Agents with Databases As soon as you have finished configuring the Scheduler Agent, you can register the Agent on one or more databases that are to run remote jobs. You can also log in later on and register the agent with additional databases.

1. If you have already logged out, then log in to the host that is running the Scheduler agent, as follows:
   - For Windows, log in as an administrator.
   - For UNIX and Linux, log in as the user with which you installed the Scheduler agent.
2. Use the following command for each database that you want to register the Scheduler agent on:
   - On UNIX and Linux, run this command:

     ```
     AGENT_HOME/bin/schagent -registerdatabase db_host db_http_port
     ```

- On Windows, run this command:

  ```
  AGENT_HOME/bin/schagent.exe -registerdatabase db_host db_http_port
  ```

where:

- *db_host* is the host name or IP address of the host on which the database resides. In an Oracle Real Application Clusters environment, you can specify any node.
- *db_http_port* is the port number that the database listens on for HTTP connections. You set this parameter previously in "Enabling and Disabling Databases for Remote Jobs" on page 30-5. You can check the port number by submitting the following SQL statement to the database:

  ```
  SELECT DBMS_XDB.GETHTTPPORT() FROM DUAL;
  ```

  A port number of 0 means that HTTP connections are disabled.

The agent prompts you to enter the agent registration password that you set in "Enabling and Disabling Databases for Remote Jobs" on page 30-5.

3. Repeat the previous steps for any additional databases to run remote jobs on the agent's host.

Monitoring and Managing the Scheduler

The following sections discuss how to monitor and manage the Scheduler:

- Viewing the Currently Active Window and Resource Plan
- Finding Information About Currently Running Jobs
- Monitoring and Managing Window and Job Logs
- Managing Scheduler Security

Viewing the Currently Active Window and Resource Plan

You can view the currently active window and the plan associated with it by issuing the following statement:

```
SELECT WINDOW_NAME, RESOURCE_PLAN FROM DBA_SCHEDULER_WINDOWS
WHERE ACTIVE='TRUE';

WINDOW_NAME                    RESOURCE_PLAN
------------------------------ --------------------------
MY_WINDOW10                    MY_RESOURCEPLAN1
```

If there is no window active, you can view the active resource plan by issuing the following statement:

```
SELECT * FROM V$RSRC_PLAN;
```

Finding Information About Currently Running Jobs

You can check a job's state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME = 'MY_EMP_JOB1';

JOB_NAME                       STATE
```

```
------------------------------ ---------
MY_EMP_JOB1                    DISABLED
```

In this case, you could enable the job using the ENABLE procedure. Table 30–2 shows the valid values for job state.

Table 30–2 Job States

Job State	Description
disabled	The job is disabled.
scheduled	The job is scheduled to be executed.
running	The job is currently running.
completed	The job has completed, and is not scheduled to run again.
stopped	The job was scheduled to run once and was stopped while it was running.
broken	The job is broken.
failed	The job was scheduled to run once and failed.
retry scheduled	The job has failed at least once and a retry has been scheduled to be executed.
succeeded	The job was scheduled to run once and completed successfully.
chain_stalled	The job is of type chain and has no steps running, no steps scheduled to run, and no event steps waiting on an event, and the chain evaluation_interval is set to NULL. No progress will be made in the chain unless there is manual intervention.

You can check the progress of currently running jobs by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_JOBS;
```

Note that, for the column CPU_USED to show valid data, the initialization parameter RESOURCE_LIMIT must be set to true.

You can check the status of all jobs at all remote and local destinations by issuing the following statement:

```
SELECT * FROM DBA_SCHEDULER_JOB_DESTS;
```

You can find out information about a job that is part of a running chain by issuing the following statement:

```
SELECT * FROM ALL_SCHEDULER_RUNNING_CHAINS WHERE JOB_NAME='MY_JOB1';
```

You can check whether the job coordinator is running by searching for a process of the form cjqNNN.

See Also:

- *Oracle Database Reference* for details regarding the *_SCHEDULER_RUNNING_JOBS and DBA_SCHEDULER_JOBS views
- "Multiple-Destination Jobs" on page 28-18

Monitoring and Managing Window and Job Logs

The Scheduler supports two kinds of logs: the job log and the window log.

Job Log

You can view information about job runs, job state changes, and job failures in the job log. The job log is implemented as the following two data dictionary views:

- `*_SCHEDULER_JOB_LOG`
- `*_SCHEDULER_JOB_RUN_DETAILS`

You can control the amount of logging that the Scheduler performs on jobs at both the job class and individual job level. Normally, you control logging at the class level, as this offers you more control over logging for the jobs in the class.

See "Viewing the Job Log" on page 29-65 for definitions of the various logging levels and for information about logging level precedence between jobs and their job class. By default, the logging level of job classes is `LOGGING_RUNS`, which causes all job runs to be logged.

You can set the `logging_level` attribute when you create the job class, or you can use the `SET_ATTRIBUTE` procedure to change the logging level at a later time. The following example sets the logging level of jobs in the `myclass1` job class to `LOGGING_FAILED_RUNS`, which means that only failed runs are logged. Note that all job classes are in the `SYS` schema.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   'sys.myclass1', 'logging_level', DBMS_SCHEDULER.LOGGING_FAILED_RUNS);
END;
/
```

You must be granted the `MANAGE SCHEDULER` privilege to set the logging level of a job class.

See Also:

- "Viewing the Job Log" on page 29-65 for more detailed information about the job log and for examples of queries against the job log views
- *Oracle Database Reference* for details on the job log views.
- *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_JOB_CLASS` and `SET_ATTRIBUTE` procedures
- "Setting Scheduler Preferences" on page 30-2 for information about setting retention for log entries

Window Log

The Scheduler makes an entry in the window log each time that:

- You create or drop a window
- A window opens
- A window closes
- Windows overlap
- You enable or disable a window

There are no logging levels for window activity logging.

To see the contents of the window log, query the `DBA_SCHEDULER_WINDOW_LOG` view. The following statement shows sample output from this view:

```
SELECT log_id, to_char(log_date, 'DD-MON-YY HH24:MI:SS') timestamp,
   window_name, operation FROM DBA_SCHEDULER_WINDOW_LOG;

    LOG_ID TIMESTAMP            WINDOW_NAME       OPERATION
---------- -------------------- ----------------- --------
         4 10/01/2004 15:29:23  WEEKEND_WINDOW    CREATE
         5 10/01/2004 15:33:01  WEEKEND_WINDOW    UPDATE
        22 10/06/2004 22:02:48  WEEKNIGHT_WINDOW  OPEN
        25 10/07/2004 06:59:37  WEEKNIGHT_WINDOW  CLOSE
        26 10/07/2004 22:01:37  WEEKNIGHT_WINDOW  OPEN
        29 10/08/2004 06:59:51  WEEKNIGHT_WINDOW  CLOSE
```

The DBA_SCHEDULER_WINDOWS_DETAILS view provides information about every window that was active and is now closed (completed). The following statement shows sample output from that view:

```
SELECT LOG_ID, WINDOW_NAME, ACTUAL_START_DATE, ACTUAL_DURATION
  FROM DBA_SCHEDULER_WINDOW_DETAILS;

    LOG_ID WINDOW_NAME      ACTUAL_START_DATE                    ACTUAL_DURATION
---------- ---------------- ------------------------------------ ---------------
        25 WEEKNIGHT_WINDOW 06-OCT-04 10:02.48.832438 PM PST8PDT +000 01:02:32
        29 WEEKNIGHT_WINDOW 07-OCT-04 10.01.37.025704 PM PST8PDT +000 03:02:00
```

Notice that log IDs correspond in both of these views, and that in this case the rows in the DBA_SCHEDULER_WINDOWS_DETAILS view correspond to the CLOSE operations in the DBA_SCHEDULER_WINDOW_LOG view.

See Also:

- *Oracle Database Reference* for details on the window log views.

Purging Logs

To prevent job and window logs from growing indiscriminately, use the SET_SCHEDULER_ATTRIBUTE procedure to specify how much history (in days) to keep. Once per day, the Scheduler automatically purges all log entries that are older than the specified history period from both the job log and the window log. The default history period is 30 days. For example, to change the history period to 90 days, issue the following statement:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','90');
```

Some job classes are more important than others. Because of this, you can override this global history setting by using a class-specific setting. For example, suppose that there are three job classes (class1, class2, and class3), and that you want to keep 10 days of history for the window log, class1, and class3, but 30 days for class2. To achieve this, issue the following statements:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
DBMS_SCHEDULER.SET_ATTRIBUTE('class2','log_history','30');
```

You can also set the class-specific history when creating the job class.

Note that log entries pertaining to steps of a chain run are not purged until the entries for the main chain job are purged.

Purging Logs Manually

The PURGE_LOG procedure enables you to manually purge logs. As an example, the following statement purges all entries from both the job and window logs:

```
DBMS_SCHEDULER.PURGE_LOG();
```

Another example is the following, which purges all entries from the jog log that are older than three days. The window log is not affected by this statement.

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 3, which_log => 'JOB_LOG');
```

The following statement purges all window log entries older than 10 days and all job log entries older than 10 days that relate to `job1` and to the jobs in `class2`:

```
DBMS_SCHEDULER.PURGE_LOG(log_history => 10, job_name => 'job1, sys.class2');
```

Managing Scheduler Security

You should grant the `CREATE JOB` system privilege to regular users who need to be able to use the Scheduler to schedule and run jobs. You should grant `MANAGE SCHEDULER` to any database administrator who must be able to manage system resources. Granting any other Scheduler system privilege or role should not be done without great caution. In particular, the `CREATE ANY JOB` system privilege and the `SCHEDULER_ADMIN` role, which includes it, are very powerful because they allow execution of code as any user. They should only be granted to very powerful roles or users.

A particularly important issue from a security point of view is handling external jobs. Only users that need to run jobs outside of the database should be allowed to do so. You must grant the `CREATE EXTERNAL JOB` system privilege to those users. Security for the Scheduler has no other special requirements. See *Oracle Database Security Guide* for details regarding security.

> **Note:** When upgrading from Oracle Database 10g Release 1 to 10g Release 2 or later, `CREATE EXTERNAL JOB` is automatically granted to all users and roles that have the `CREATE JOB` privilege. Oracle recommends that you revoke this privilege from users that do not need it.

Import/Export and the Scheduler

You must use the Data Pump utilities (`impdp` and `expdp`) to export Scheduler objects. You cannot use the earlier import/export utilities (`IMP` and `EXP`) with the Scheduler. Also, Scheduler objects cannot be exported while the database is in read-only mode.

An export generates the DDL that was used to create the Scheduler objects. All attributes are exported. When an import is done, all the database objects are re-created in the new database. All schedules are stored with their time zones, which are maintained in the new database. For example, schedule "Monday at 1 PM PST in a database in San Francisco" would be the same if it was exported and imported to a database in Germany.

Although Scheduler credentials are exported, for security reasons, the passwords in these credentials are not exported. After you import Scheduler credentials, you must reset the passwords using the `SET_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package.

> **See Also:** *Oracle Database Utilities* for details on Data Pump

Troubleshooting the Scheduler

This section contains the following troubleshooting topics:

- A Job Does Not Run

- A Program Becomes Disabled
- A Window Fails to Take Effect

A Job Does Not Run

A job may fail to run for several reasons. To begin troubleshooting a job that you suspect did not run, check the job state by issuing the following statement:

```
SELECT JOB_NAME, STATE FROM DBA_SCHEDULER_JOBS;
```

Typical output will resemble the following:

```
JOB_NAME                       STATE
------------------------------ ---------
MY_EMP_JOB                     DISABLED
MY_EMP_JOB1                    FAILED
MY_NEW_JOB1                    DISABLED
MY_NEW_JOB2                    BROKEN
MY_NEW_JOB3                    COMPLETED
```

About Job States

There a four states that a job could be in if it does not run:

- Failed Jobs
- Broken Jobs
- Disabled Jobs
- Completed Jobs

Failed Jobs If a job has the status of `FAILED` in the job table, it was scheduled to run once but the execution has failed. If the job was specified as restartable, all retries have failed.

If a job fails in the middle of execution, only the last transaction of that job is rolled back. If your job executes multiple transactions, then you must be careful about setting `restartable` to `TRUE`. You can query failed jobs by querying the `*_SCHEDULER_JOB_RUN_DETAILS` views.

Broken Jobs A broken job is one that has exceeded a certain number of failures. This number is set in `max_failures`, and can be altered. In the case of a broken job, the entire job is broken, and it will not be run until it has been fixed. For debugging and testing, you can use the `RUN_JOB` procedure.

You can query broken jobs by querying the `*_SCHEDULER_JOBS` and `*_SCHEDULER_JOB_LOG` views.

Disabled Jobs A job can become disabled for the following reasons:

- The job was manually disabled
- The job class it belongs to was dropped
- The program, chain, or schedule that it points to was dropped
- A window or window group is its schedule and the window or window group is dropped

Completed Jobs A job will be completed if `end_date` or `max_runs` is reached. (If a job recently completed successfully but is scheduled to run again, the job state is `SCHEDULED`.)

Viewing the Job Log

An important troubleshooting tool is the job log. For details and instructions, see "Viewing the Job Log" on page 29-65.

Troubleshooting Remote Jobs

Remote jobs must successfully communicate with a Scheduler agent on the remote host. If a remote job does not run, check the `DBA_SCHEDULER_JOBS` view and the job log first. Then perform the following tasks:

1. Check that the remote system is reachable over the network with tools such as `nslookup` and `ping`.
2. Check the status of the Scheduler agent on the remote host by calling the `GET_AGENT_VERSION` package procedure.

   ```
   DECLARE
     versionnum VARCHAR2(30);
   BEGIN
     versionnum := DBMS_SCHEDULER.GET_AGENT_VERSION('remote_host.example.com');
     DBMS_OUTPUT.PUT_LINE(versionnum);
   END;
   /
   ```

   If an error is generated, the agent may not be installed or may not be registered with your local database. See "Enabling and Disabling Databases for Remote Jobs" on page 30-5 for instructions for installing, registering, and starting the Scheduler agent.

About Job Recovery After a Failure

The Scheduler attempts to recover jobs that are interrupted when:

- The database abnormally shuts down
- A job slave process is killed or otherwise fails
- For an external job, the external job process that starts the executable or script is killed or otherwise fails. (The external job process is `extjob` on UNIX. On Windows, it is the external job service.)
- For an external job, the process that runs the end-user executable or script is killed or otherwise fails.

Job recovery proceeds as follows:

- The Scheduler adds an entry to the job log for the instance of the job that was running when the failure occurred. In the log entry, the `OPERATION` is `'RUN'`, the `STATUS` is `'STOPPED'`, and `ADDITIONAL_INFO` contains one of the following:
  - `REASON="Job slave process was terminated"`
  - `REASON="ORA-01014: ORACLE shutdown in progress"`
- If `restartable` is set to `TRUE` for the job, the job is restarted.
- If `restartable` is set to `FALSE` for the job:

- If the job is a run-once job and `auto_drop` is set to `TRUE`, the job run is done and the job is dropped.
- If the job is a run-once job and `auto_drop` is set to `FALSE`, the job is disabled and the job `state` is set to '`STOPPED`'.
- If the job is a repeating job, the Scheduler schedules the next job run and the job `state` is set to '`SCHEDULED`'.

When a job is restarted as a result of this recovery process, the new run is entered into the job log with the operation '`RECOVERY_RUN`'.

A Program Becomes Disabled

A program can become disabled if a program argument is dropped or `number_of_arguments` is changed so that all arguments are no longer defined.

See "Creating and Managing Programs to Define Jobs" on page 29-21 for more information regarding programs.

A Window Fails to Take Effect

A window can fail to take effect for the following reasons:

- A window becomes disabled when it is at the end of its schedule
- A window that points to a schedule that no longer exists is disabled

See "Managing Job Scheduling and Job Priorities with Windows" on page 29-55 for more information regarding windows.

Examples of Using the Scheduler

This section discusses the following topics:

- Examples of Creating Job Classes
- Examples of Setting Attributes
- Examples of Creating Chains
- Examples of Creating Jobs and Schedules Based on Events
- Example of Creating a Job In an Oracle Data Guard Environment

Examples of Creating Job Classes

This section contains several examples of creating job classes. To create a job class, you use the `CREATE_JOB_CLASS` procedure.

Example 30–1 Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name              =>  'my_class1',
   service                     =>  'my_service1',
   comments                    =>  'This is my first job class');
END;
/
```

This creates my_class1 in SYS. It uses a service called my_service1. To verify that the job class was created, issue the following statement:

```
SELECT JOB_CLASS_NAME FROM DBA_SCHEDULER_JOB_CLASSES
WHERE JOB_CLASS_NAME = 'MY_CLASS1';

JOB_CLASS_NAME
------------------------------
MY_CLASS1
```

Example 30–2 Creating a Job Class

The following statement creates a job class:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB_CLASS (
   job_class_name             =>  'finance_jobs',
   resource_consumer_group    =>  'finance_group',
   service                    =>  'accounting',
   comments                   =>  'All finance jobs');
END;
/
```

This creates finance_jobs in SYS. It assigns a resource consumer group called finance_group, and designates service affinity for the accounting service. Note that if the accounting service is mapped to a resource consumer group other than finance_group, jobs in this class run under the finance_group consumer group, because the resource_consumer_group attribute takes precedence.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the CREATE_JOB_CLASS procedure and "Creating Job Classes" on page 29-54 for further information

Examples of Setting Attributes

This section contains several examples of setting attributes. To set attributes, you use SET_ATTRIBUTE and SET_SCHEDULER_ATTRIBUTE procedures.

Example 30–3 Setting the Repeat Interval Attribute

The following example resets the frequency my_emp_job1 will run to daily:

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE (
   name           =>   'my_emp_job1',
   attribute      =>   'repeat_interval',
   value          =>   'FREQ=DAILY');
END;
/
```

To verify the change, issue the following statement:

```
SELECT JOB_NAME, REPEAT_INTERVAL FROM DBA_SCHEDULER_JOBS
WHERE JOB_NAME =  'MY_EMP_JOB1';

JOB_NAME             REPEAT_INTERVAL
----------------     ---------------
MY_EMP_JOB1          FREQ=DAILY
```

Example 30–4 Setting Multiple Job Attributes for a Set of Jobs

The following example sets four different attributes for each of five jobs:

```
DECLARE
 newattr sys.jobattr;
 newattrarr sys.jobattr_array;
 j number;
BEGIN
 -- Create new JOBATTR array
 newattrarr := sys.jobattr_array();

 -- Allocate enough space in the array
 newattrarr.extend(20);
 j := 1;
 FOR i IN 1..5 LOOP
   -- Create and initialize a JOBATTR object type
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'MAX_FAILURES',
                          attr_value => 5);
   -- Add it to the array.
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'COMMENTS',
                          attr_value => 'Test job');
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'END_DATE',
                          attr_value => systimestamp + interval '24' hour);
   newattrarr(j) := newattr;
   j := j + 1;
   newattr := sys.jobattr(job_name => 'TESTJOB' || to_char(i),
                          attr_name => 'SCHEDULE_LIMIT',
                          attr_value => interval '1' hour);
   newattrarr(j) := newattr;
   j := j + 1;
 END LOOP;

 -- Call SET_JOB_ATTRIBUTES to set all 20 set attributes in one transaction
 DBMS_SCHEDULER.SET_JOB_ATTRIBUTES(newattrarr, 'TRANSACTIONAL');
END;
/
```

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `SET_SCHEDULER_ATTRIBUTE` procedure and "Setting Scheduler Preferences" on page 30-2

Examples of Creating Chains

This section contains examples of creating chains. To create chains, you use the `CREATE_CHAIN` procedure. After creating a chain, you add steps to the chain with the `DEFINE_CHAIN_STEP` or `DEFINE_CHAIN_EVENT_STEP` procedures and define the rules with the `DEFINE_CHAIN_RULE` procedure.

Example 30–5 Creating a Chain

The following example creates a chain where `my_program1` runs before `my_program2` and `my_program3`. `my_program2` and `my_program3` run in parallel after `my_program1` has completed.

The user for this example must have the `CREATE EVALUATION CONTEXT`, `CREATE RULE`, and `CREATE RULE SET` privileges. See "Setting Chain Privileges" on page 30-2 for more information.

```
BEGIN
  DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name            =>  'my_chain1',
   rule_set_name         =>  NULL,
   evaluation_interval   =>  NULL,
   comments              =>  NULL);
END;
/

--- define three steps for this chain. Referenced programs must be enabled.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepA', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepB', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain1', 'stepC', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE('my_chain1', 'TRUE', 'START stepA');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepA COMPLETED', 'Start stepB, stepC');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain1', 'stepB COMPLETED AND stepC COMPLETED', 'END');
END;
/

--- enable the chain
BEGIN
 DBMS_SCHEDULER.ENABLE('my_chain1');
END;
/

--- create a chain job to start the chain daily at 1:00 p.m.
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
   job_name        => 'chain_job_1',
   job_type        => 'CHAIN',
   job_action      => 'my_chain1',
   repeat_interval => 'freq=daily;byhour=13;byminute=0;bysecond=0',
   enabled         => TRUE);
END;
/
```

Example 30–6 Creating a Chain

The following example creates a chain where first `my_program1` runs. If it succeeds, `my_program2` runs; otherwise, `my_program3` runs.

```
BEGIN
 DBMS_SCHEDULER.CREATE_CHAIN (
   chain_name            => 'my_chain2',
```

```
   rule_set_name         => NULL,
   evaluation_interval   => NULL,
   comments              => NULL);
END;
/

--- define three steps for this chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step1', 'my_program1');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step2', 'my_program2');
 DBMS_SCHEDULER.DEFINE_CHAIN_STEP('my_chain2', 'step3', 'my_program3');
END;
/

--- define corresponding rules for the chain.
BEGIN
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE ('my_chain2', 'TRUE', 'START step1');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 SUCCEEDED', 'Start step2');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step1 COMPLETED AND step1 NOT SUCCEEDED', 'Start step3');
 DBMS_SCHEDULER.DEFINE_CHAIN_RULE (
   'my_chain2', 'step2 COMPLETED OR step3 COMPLETED', 'END');
END;
/
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_CHAIN`, `DEFINE_CHAIN_STEP`, and `DEFINE_CHAIN_RULE` procedures and "Setting Scheduler Preferences" on page 30-2

Examples of Creating Jobs and Schedules Based on Events

This section contains examples of creating event-based jobs and event schedules. To create event-based jobs, you use the `CREATE_JOB` procedure. To create event-based schedules, you use the `CREATE_EVENT_SCHEDULE` procedure.

These examples assume the existence of an application that, when it detects the arrival of a file on a system, enqueues an event onto the queue `my_events_q`.

Example 30–7 Creating an Event-Based Schedule

The following example illustrates creating a schedule that can be used to start a job whenever the Scheduler receives an event indicating that a file arrived on the system before 9AM:

```
BEGIN
  DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE (
   schedule_name     =>  'scott.file_arrival',
   start_date        =>  systimestamp,
   event_condition   =>  'tab.user_data.object_owner = ''SCOTT''
      and tab.user_data.event_name = ''FILE_ARRIVAL''
      and extract hour from tab.user_data.event_timestamp < 9',
   queue_spec        =>  'my_events_q');
END;
/
```

Example 30–8 Creating an Event-Based Job

The following example creates a job that starts when the Scheduler receives an event indicating that a file arrived on the system:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
   job_name            =>  my_job,
   program_name        =>  my_program,
   start_date          =>  '15-JUL-04 1.00.00AM US/Pacific',
   event_condition     =>  'tab.user_data.event_name = ''LOW_INVENTORY''',
   queue_spec          =>  'my_events_q'
   enabled             =>  TRUE,
   comments            =>  'my event-based job');
END;
/
```

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the `CREATE_JOB` and `CREATE_EVENT_SCHEDULE` procedures

Example of Creating a Job In an Oracle Data Guard Environment

In an Oracle Data Guard environment, the Scheduler includes additional support for two database roles: primary and logical standby. You can configure a job to run only when the database is in the primary role or only when the database is in the logical standby role. To do so, you set the `database_role` attribute. This example explains how to enable a job to run in both database roles. The method used is to create two copies of the job and assign a different `database_role` attribute to each.

By default, a job runs when the database is in the role that it was in when the job was created. You can run the same job in both roles using the following steps:

1. Copy the job
2. Enable the new job
3. Change the `database_role` attribute of the new job to the required role

The example starts by creating a job called `primary_job` on the primary database. It then makes a copy of this job and sets its `database_role` attribute to `'LOGICAL STANDBY'`. If the primary database then becomes a logical standby, the job continues to run according to its schedule.

When you copy a job, the new job is disabled, so you must enable the new job.

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB (
     job_name       => 'primary_job',
     program_name   => 'my_prog',
     schedule_name  => 'my_sched');

 DBMS_SCHEDULER.COPY_JOB('primary_job','standby_job');
 DBMS_SCHEDULER.ENABLE(name=>'standby_job', commit_semantics=>'ABSORB_ERRORS');
 DBMS_SCHEDULER.SET_ATTRIBUTE('standby_job','database_role','LOGICAL STANDBY');
END;
/
```

After you execute this example, the data in the `DBA_SCHEDULER_JOB_ROLES` view is as follows:

```
SELECT JOB_NAME, DATABASE_ROLE FROM DBA_SCHEDULER_JOB_ROLES
```

```
  WHERE JOB_NAME IN ('PRIMARY_JOB','STANDBY_JOB');

JOB_NAME               DATABASE_ROLE
--------               ----------------
PRIMARY_JOB            PRIMARY
STABDBY_JOB            LOGICAL STANDBY
```

Note: For a physical standby database, any changes made to Scheduler objects or any database changes made by Scheduler jobs on the primary database are applied to the physical standby like any other database changes.

Scheduler Reference

This section contains reference information for Oracle Scheduler. It contains the following topics:

- Scheduler Privileges
- Scheduler Data Dictionary Views

Scheduler Privileges

Table 30–3, " Scheduler System Privileges" and Table 30–4, " Scheduler Object Privileges" describe the various Scheduler privileges.

Table 30–3 Scheduler System Privileges

Privilege Name	Operations Authorized
CREATE JOB	This privilege enables you to create jobs, chains, schedules, programs, file watchers, credentials, destinations, and groups in your own schema. You can always alter and drop these objects in your own schema, even if you do not have the CREATE JOB privilege. In this case, the object would have been created in your schema by another user with the CREATE ANY JOB privilege.
CREATE ANY JOB	This privilege enables you to create, alter, and drop jobs, chains, schedules, programs, file watchers, credentials, destinations, and groups in any schema except SYS. This privilege is extremely powerful and should be used with care because it allows the grantee to execute any PL/SQL code as any other database user.
CREATE EXTERNAL JOB	This privilege is required to create jobs that run outside of the database. Owners of jobs of type 'EXECUTABLE' or jobs that point to programs of type 'EXECUTABLE' require this privilege. To run a job of type 'EXECUTABLE', you must have this privilege and the CREATE JOB privilege. This privilege is also required to retrieve files from a remote host and to save files to one or more remote hosts.
EXECUTE ANY PROGRAM	This privilege enables your jobs to use programs or chains from any schema.
EXECUTE ANY CLASS	This privilege enables your jobs to run under any job class.
MANAGE SCHEDULER	This is the most important privilege for administering the Scheduler. It enables you to create, alter, and drop job classes, windows, and window groups, and to stop jobs with the force option. It also enables you to set and retrieve Scheduler attributes, purge Scheduler logs, and set the agent password for a database.

Table 30–4 Scheduler Object Privileges

Privilege Name	Operations Authorized
SELECT	You can grant object privileges on a group to other users by granting SELECT on the group.
EXECUTE	You can grant this privilege only on programs, chains, file watchers, credentials, and job classes. The EXECUTE privilege enables you to reference the object in a job. It also enables you to view the object if the object is was not created in your schema.
ALTER	This privilege enables you to alter or drop the object it is granted on. Altering includes such operations as enabling, disabling, defining or dropping program arguments, setting or resetting job argument values and running a job. Certain restricted attributes of jobs of job type EXECUTABLE cannot be altered using the ALTER object privilege. These include job_type, job_action, number_of_arguments, event_spec, and setting PL/SQL date functions as schedules. For programs, jobs, chains, file watchers, and credentials, this privilege also enables schemas that do not own these objects to view them. This privilege can be granted on jobs, chains, programs, schedules, file watchers, and credentials. For other types of Scheduler objects, you must grant the MANAGE SCHEDULER system privilege.
ALL	This privilege authorizes operations allowed by all other object privileges possible for a given object. It can be granted on jobs, programs, chains, schedules, file watchers, credentials, and job classes.

Note: No object privileges are required to use a destination object created by another user.

The SCHEDULER_ADMIN role is created with all of the system privileges shown in Table 30–3 (with the ADMIN option). The SCHEDULER_ADMIN role is granted to DBA (with the ADMIN option).

The DBMS_SCHEDULER package ignores privileges granted on scheduler objects, such as jobs or chains, through roles. Object privileges must be granted directly to the user.

The following object privileges are granted to PUBLIC: SELECT ALL_SCHEDULER_* views, SELECT USER_SCHEDULER_* views, SELECT SYS.SCHEDULER$_JOBSUFFIX_S (for generating a job name), and EXECUTE SYS.DEFAULT_JOB_CLASS.

Scheduler Data Dictionary Views

You can check Scheduler information by using many views. An example is the following, which shows information for completed instances of my_job1:

```
SELECT JOB_NAME, STATUS, ERROR#
FROM DBA_SCHEDULER_JOB_RUN_DETAILS WHERE JOB_NAME = 'MY_JOB1';

JOB_NAME     STATUS           ERROR#
--------     --------------   ------
MY_JOB1      FAILURE           20000
```

Table 30–5 contains views associated with the Scheduler. The *_SCHEDULER_JOBS, *_SCHEDULER_SCHEDULES, *_SCHEDULER_PROGRAMS, *_SCHEDULER_RUNNING_JOBS, *_SCHEDULER_JOB_LOG, *_SCHEDULER_JOB_RUN_DETAILS views are particularly useful for managing jobs. See *Oracle Database Reference* for details regarding Scheduler views.

Note: In the following table, the asterisk at the beginning of a view name can be replaced with DBA, ALL, or USER.

Table 30–5 Scheduler Views

View	Description
*_SCHEDULER_CHAIN_RULES	These views show all rules for all chains.
*_SCHEDULER_CHAIN_STEPS	These views show all steps for all chains.
*_SCHEDULER_CHAINS	These views show all chains.
*_SCHEDULER_CREDENTIALS	These views show all credentials.
*_SCHEDULER_DB_DESTS	These views show all database destinations.
*_SCHEDULER_DESTS	These views show all destinations, both database and external.
*_SCHEDULER_EXTERNAL_DESTS	These views show all external destinations.
*_SCHEDULER_FILE_WATCHERS	These views show all file watchers.
*_SCHEDULER_GLOBAL_ATTRIBUTE	These views show the current values of Scheduler attributes.
*_SCHEDULER_GROUP_MEMBERS	These views show all group members in all groups.
*_SCHEDULER_GROUPS	These views show all groups.
*_SCHEDULER_JOB_ARGS	These views show all set argument values for all jobs.
*_SCHEDULER_JOB_CLASSES	These views show all job classes.
*_SCHEDULER_JOB_DESTS	These views show the state of both local jobs and jobs at remote destinations, including child jobs of multiple-destination jobs. You obtain job destination IDs (job_dest_id) from these views.
*_SCHEDULER_JOB_LOG	These views show job runs and state changes, depending on the logging level set.
*_SCHEDULER_JOB_ROLES	These views show all jobs by Oracle Data Guard database role.
*_SCHEDULER_JOB_RUN_DETAILS	These views show all completed (failed or successful) job runs.
*_SCHEDULER_JOBS	These views show all jobs, enabled as well as disabled.
*_SCHEDULER_NOTIFICATIONS	These views show all job state e-mail notifications.
*_SCHEDULER_PROGRAM_ARGS	These views show all arguments defined for all programs as well as the default values if they exist.
*_SCHEDULER_PROGRAMS	These views show all programs.
*_SCHEDULER_RUNNING_CHAINS	These views show all chains that are running.
*_SCHEDULER_RUNNING_JOBS	These views show state information on all jobs that are currently being run.
*_SCHEDULER_SCHEDULES	These views show all schedules.
*_SCHEDULER_WINDOW_DETAILS	These views show all completed window runs.
*_SCHEDULER_WINDOW_GROUPS	These views show all window groups.
*_SCHEDULER_WINDOW_LOG	These views show all state changes made to windows.
*_SCHEDULER_WINDOWS	These views show all windows.
*_SCHEDULER_WINGROUP_MEMBERS	These views show the members of all window groups, one row for each group member.

Part V

Distributed Database Management

Part V discusses the management of a distributed database environment. It contains the following chapters:

- Chapter 31, "Distributed Database Concepts"
- Chapter 32, "Managing a Distributed Database"
- Chapter 33, "Developing Applications for a Distributed Database System"
- Chapter 34, "Distributed Transactions Concepts"
- Chapter 35, "Managing Distributed Transactions"

31

Distributed Database Concepts

In this chapter:

- 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 not an 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

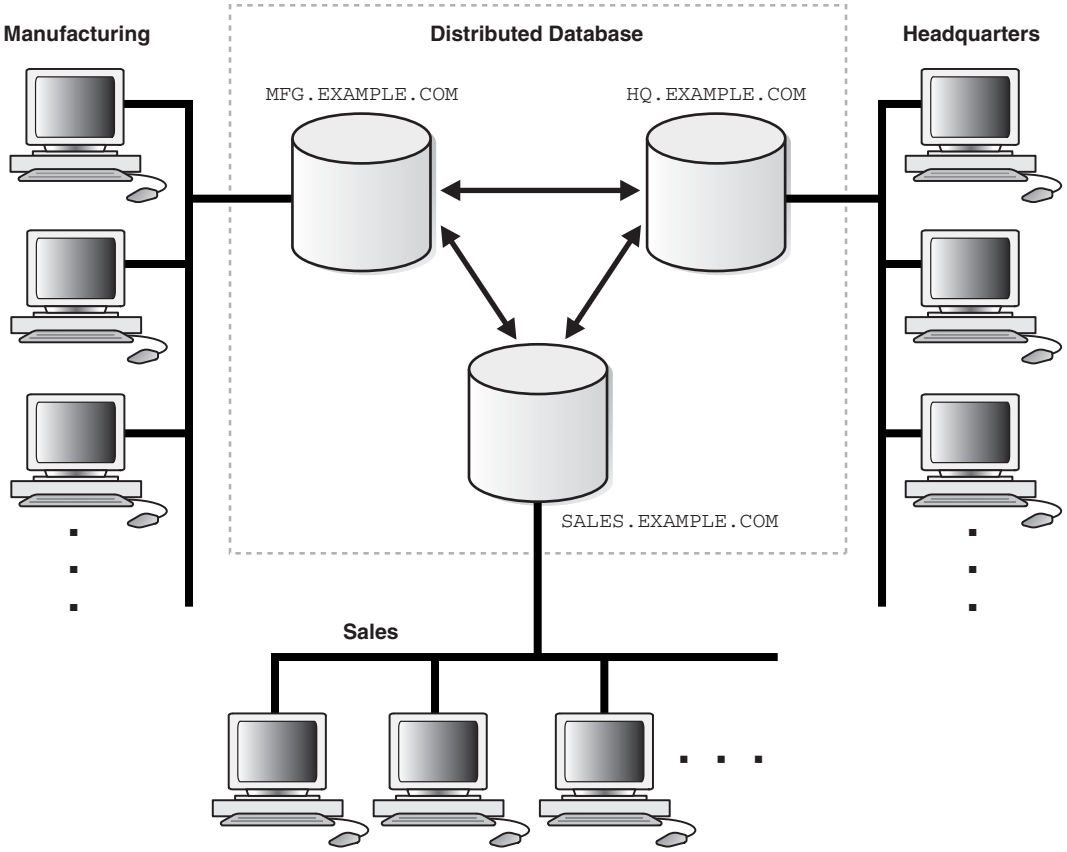
A homogenous distributed database system is a network of two or more Oracle Databases that reside on one or more systems. Figure 31–1 illustrates a distributed system that connects three databases: `hq`, `mfg`, and `sales`. An application can simultaneously access or modify the data in several databases 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 31–1 Homogeneous Distributed Database



An Oracle Database distributed database system can incorporate Oracle Databases of different versions. All supported releases of Oracle Database 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 Oracle Database.

Distributed Databases Versus Distributed Processing

The terms **distributed database** and **distributed processing** are closely related, yet have distinct meanings. There 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.

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:

- *Oracle Database Advanced Replication* for more information about Oracle Database replication features
- *Oracle Streams Concepts and Administration* for information about Oracle Streams, another method of sharing information between databases

Heterogeneous Distributed Database Systems

In a heterogeneous distributed database system, at least one of the databases is a non-Oracle Database 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 Database system using Oracle Heterogeneous Services with an **agent**. If you access the non-Oracle Database 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 Database distributed system, then you must obtain a Sybase-specific transparent gateway so that the Oracle Database in the system can communicate with it.

Alternatively, you can use **generic connectivity** to access non-Oracle Database data stores so long as the non-Oracle Database 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 *Oracle Database Heterogeneous Connectivity User'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 Database 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 Database system that you access, Heterogeneous Services can use a transparent gateway agent to interface with the specified non-Oracle Database system. The agent is specific to the non-Oracle Database system, so each type of system requires a different agent.

The transparent gateway agent facilitates communication between Oracle Database and non-Oracle Database systems and uses the Heterogeneous Services component in the Oracle Database server. The agent executes SQL and transactional requests at the non-Oracle Database 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 Database 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.

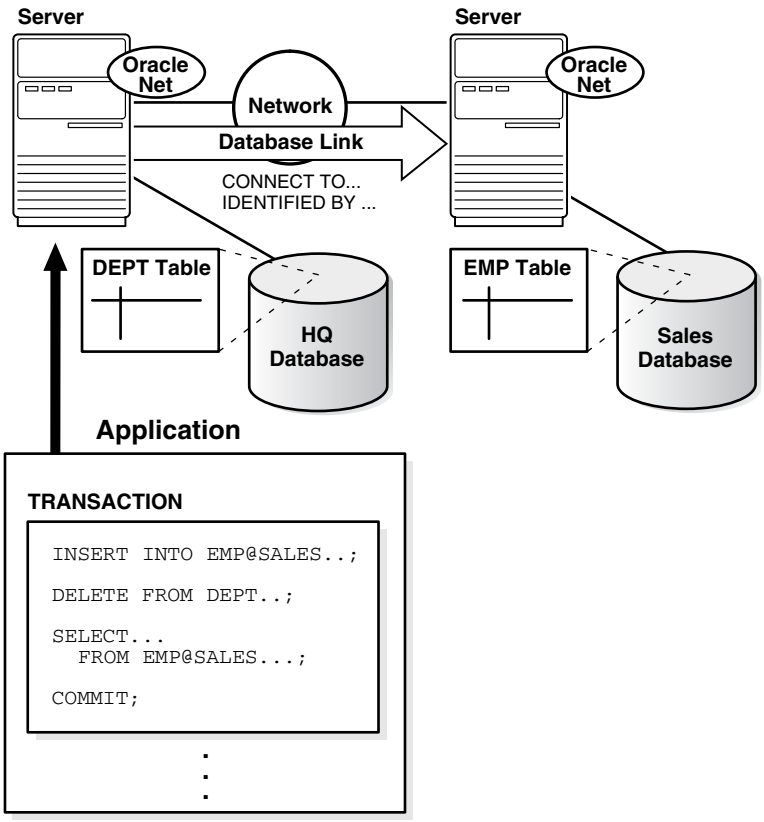
The advantage to generic connectivity is that it may not be required for you to purchase and configure a separate system-specific agent. You use an ODBC or OLE DB driver that can interface with the agent. However, some data access features are only available with transparent gateway agents.

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 31–2, 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 31–2 An Oracle Database 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 31–2, 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 31–2, 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?

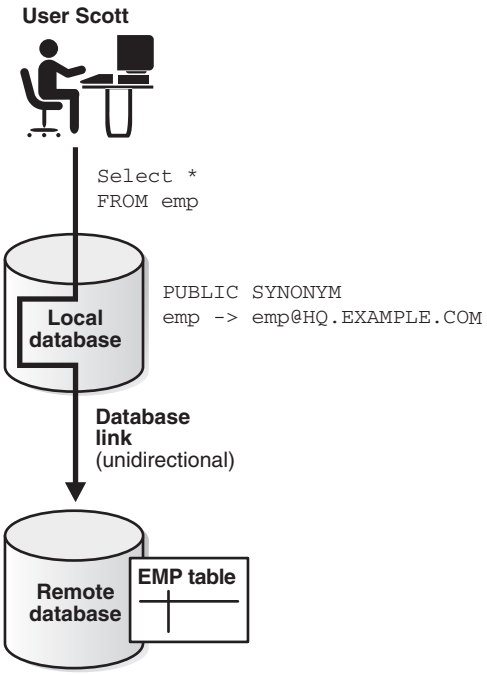
A database link is a pointer that defines a one-way communication path from an Oracle Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary 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 31–3 shows an example of user `scott` accessing the `emp` table on the remote database with the global name `hq.example.com`:

Figure 31–3 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 user name and password as their account on the local database.
Fixed user link	Users connect using the user name and password referenced in the link. For example, if Jane uses a fixed user link that connects to the `hq` database with the user name and password `scott/`password, 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:

- *Oracle Database SQL Language Reference* for syntax of the `CREATE DATABASE` statement
- *Oracle Database 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 must 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: *Oracle Database 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 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 must 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.

See Also:

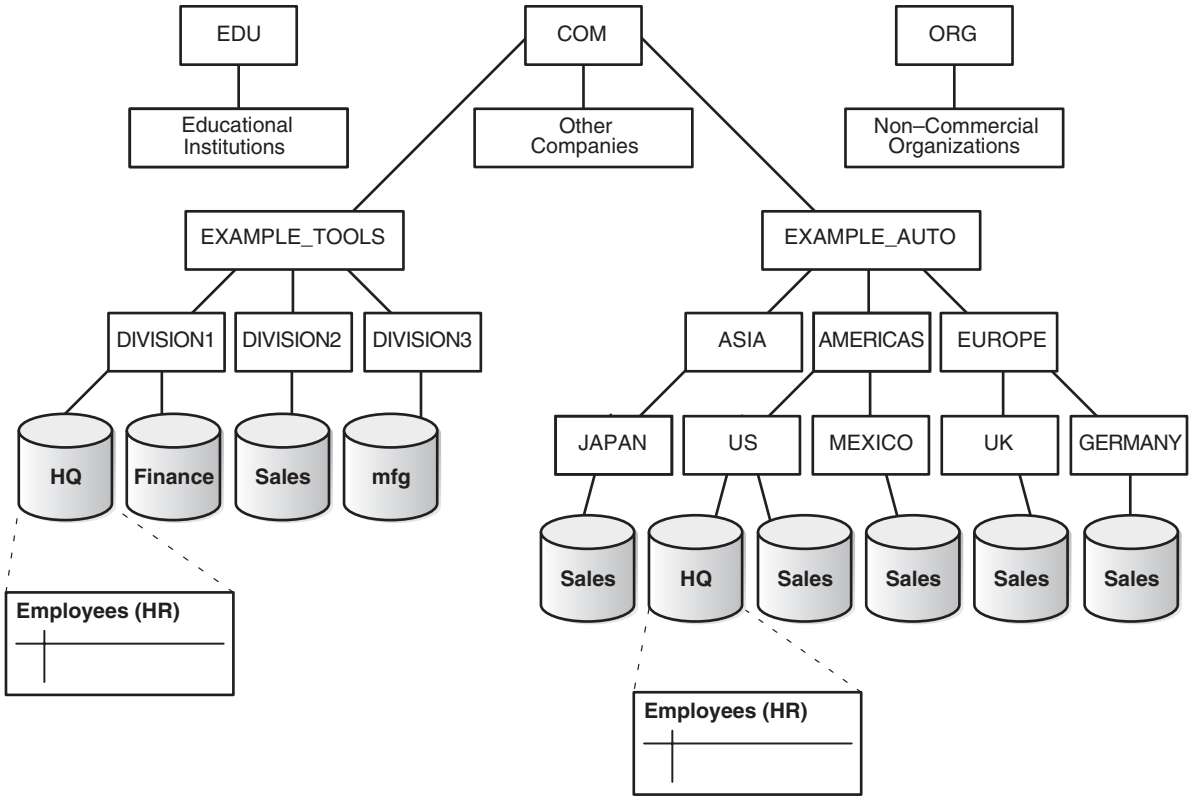
- "Users of Database Links" on page 31-11 for an explanation of database link users
- "Viewing Information About Database Links" for an explanation of how to hide passwords from non-administrative users

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. The database forms a global database name by prefixing the database 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 31–4 illustrates a representative hierarchical arrangement of databases throughout a network.

Figure 31–4 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 `example_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.example_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.example_auto.com` and `uk.europe.example_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 `europe` division as follows:

- `sales.us.americas.example_auto.com`
- `sales.uk.europe.example_auto.com`

> **See Also:** "Managing Global Names in a Distributed System" on page 32-1 to learn how to specify and change global database names

Global Name as a Loopback Database Link

You can use the global name of a database as a loopback database link without explicitly creating a database link. When the database link in a SQL statement matches the global name of the current database, the database link is effectively ignored.

For example, assume the global name of a database is `db1.example.com`. You can run the following SQL statement on this database:

```
SELECT * FROM hr.employees@db1.example.com;
```

In this case, the `@db1.example.com` portion of the SQL statement is effectively ignored.

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.example.com`, then the database link is also called `sales.us.example.com`.

When you set the initialization parameter `GLOBAL_NAMES` to `TRUE`, the database 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.example.com`, and `GLOBAL_NAMES` is `TRUE`, then the link name must be called `hq.example.com`. Note that the database 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 32-3).

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.example.com` as `foo`.

> **Note:** Oracle 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.example.com USING 'sales1';
```

> **See Also:** *Oracle Database Reference* for more information about specifying the initialization parameter `GLOBAL_NAMES`

Types of Database Links

Oracle Database 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: ■ `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 for private database links.	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.

Type	Owner	Description
Global	User called PUBLIC. View ownership data through views shown for private database links.	Creates a network-wide link. When an Oracle network uses a directory server, the directory server automatically create and manages global database links (as net service names) 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. **Note**: In earlier releases of Oracle Database, a global database link referred to a database link that was registered with an Oracle Names server. The use of an Oracle Names server has been deprecated. In this document, global database links refer to the use of net service names from the directory server.

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 a directory server, 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 32-7 to learn how to create different types of database links
- "Viewing Information About Database Links" on page 32-16 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	Description	Sample Link Creation Syntax
Connected user	A local user accessing a database link in which no fixed username and password have been specified. If `SYSTEM` accesses a public link in a query, then the connected user is `SYSTEM`, and the database connects to the `SYSTEM` schema in the remote database. **Note:** A connected user does not have to be the user who created the link, but is any user who is accessing the link.	`CREATE PUBLIC DATABASE LINK hq USING 'hq';`

User Type	Description	Sample Link Creation Syntax
Current user	A global user in a CURRENT_USER 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. See *Oracle Database Advanced Security Administrator's Guide* for information about global security	CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';
Fixed user	A user whose username/password is part of the link definition. If a link includes a fixed user, the fixed user's username and password are used to connect to the remote database.	CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY *password* USING 'hq';

Note: The following users cannot be target users of database links: SYS and PUBLIC.

See Also: "Specifying Link Users" on page 32-8 to learn how to specify users when 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, and credentials do not have to be stored in the link definition 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 the database 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:

REMOTE_OS_AUTHENT Value	Consequences
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.

Note: The REMOTE_OS_AUTHENT initialization parameter is deprecated. It is retained for backward compatibility only.

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 *password*. 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/*password*.

Fixed user links have a user name and password associated with the connect string. The user name and password are stored with other link information in data dictionary tables.

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's 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's 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 31-17 for more information about security issues relating to database links
- *Oracle Database Advanced Security Administrator's Guide*
- *Oracle Database PL/SQL Language Reference* for more information about invoker's rights functions, procedures, or packages.

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.example_auto.com` database:

SQL Statement	Connects To Database	Connects As	Link Type
`CREATE DATABASE LINK sales.us.americas.example_auto.com USING 'sales_us';`	`sales` using net service name `sales_us`	Connected user	Private connected user
`CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';`	`sales` using service name `am_sls`	Current global user	Private current user
`CREATE DATABASE LINK sales.us.americas.example_auto.com CONNECT TO scott IDENTIFIED BY` *`password`* `USING 'sales_us';`	`sales` using net service name `sales_us`	`scott` using password *`password`*	Private fixed user
`CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY` *`password`* `USING 'rev';`	`sales` using net service name `rev`	`scott` using password *`password`*	Public fixed user
`CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.example_auto.com CONNECT TO scott IDENTIFIED BY` *`password`* `AUTHENTICATED BY anupam IDENTIFIED BY` *`password1`* `USING 'sales';`	`sales` using net service name `sales`	`scott` using password *`password`*, authenticated as `anupam` using password *`password1`*	Shared public fixed user

See Also:

- "Creating Database Links" on page 32-6 to learn how to create link
- *Oracle Database SQL Language 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 Database 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 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.example.com, a user or application can reference remote data as follows:

```
SELECT * FROM scott.emp@sales.division3.example.com;  # emp table in scott's
schema
SELECT loc FROM scott.dept@sales.division3.example.com;
```

If GLOBAL_NAMES is set to FALSE, then you can use any name for the link to sales.division3.example.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 the database has no remote describe capability. The database must do a SELECT * on the remote object in order to determine its structure.

Synonyms for Schema Objects

Oracle Database 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.example.com;
```

You can create the synonym emp for emp@hq.example.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 32-20 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**), the database forms object names hierarchically. For example, the database 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 name is always unique within the database. Furthermore, the database resolves application references to the local name of the object.

In a distributed database, a schema object such as a table is accessible to all applications in the system. The database 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@sales1
```

You then issue the following statements using database link `hq.example.com` to access objects in the `scott` and `jane` schemas on remote database `hq`:

```
SELECT * FROM scott.emp@hq.example.com;
INSERT INTO jane.accounts@hq.example.com (acc_no, acc_name, balance)
  VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.example.com
  SET balance = balance + 500;
DELETE FROM jane.accounts@hq.example.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 Database distributed database system:

- Site Autonomy
- Distributed Database Security
- Auditing Database Links
- Administration Tools

See Also:

- Chapter 32, "Managing a Distributed Database" to learn how to administer homogenous systems
- *Oracle Database Heterogeneous Connectivity User'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 Database 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 Database 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

The database supports all of the security features that are available with a non-distributed 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.
- 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 Database 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 Database 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 password USING 'sales';
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO nick IDENTIFIED BY password1
     AUTHENTICATED BY david IDENTIFIED BY password2 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, the database 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.
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.
Public	No	Works the same as a private nonauthenticated 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.

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 must 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 32-6 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

The database 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 Database distributed database system. Windows NT native authentication is an example of a non-Oracle authentication solution.

> **See Also:**
>
> - *Oracle Database Advanced Security Administrator's Guide*
> - *Oracle Database Enterprise User Security Administrator's Guide*

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, the database also supports schema-independent users, which are global users that an access a single, generic schema in every database.

Schema-Independent Global Users The database 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

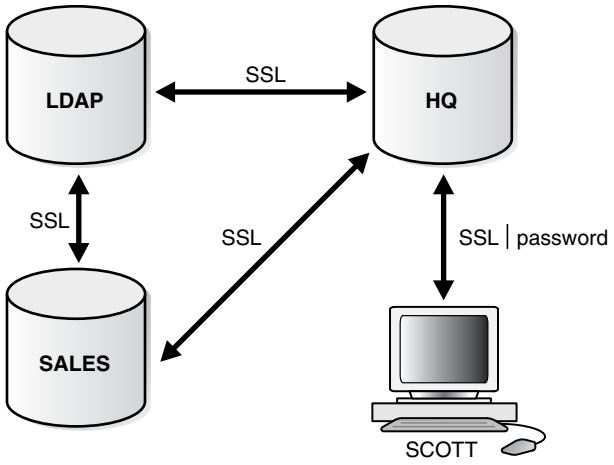
The administrator of each database is not required to create a global user account for each enterprise user on each database to which the enterprise user must connect. Instead, multiple enterprise users can connect to the same database schema, called a **shared schema**.

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 `hq` 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 `hq` application.

Figure 31–5 illustrates an example of global user security using the enterprise directory service:

Figure 31–5 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
`tb`	`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`	`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';`

`sales`	`CREATE USER guest IDENTIFIED GLOBALLY AS '';` `CREATE ROLE clerk2 GRANT select ON dept;`

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 Database Enterprise User 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 Database 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.example.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 Database distributed database system:

- Enterprise Manager
- Third-Party Administration Tools
- SNMP Support

Enterprise Manager

Enterprise Manager is the Oracle 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 Database platform in any location worldwide. In addition, these Oracle Database 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 must 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 OpenView
- Digital POLYCENTER Manager on NetView
- IBM NetView/6000
- Novell NetWare Management System
- SunSoft SunNet Manager

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 multiple nodes.

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.example_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.example_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.example_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.example_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
    WHERE deptno = 10;
END;
COMMIT;
```

The database 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:** *Oracle 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.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.example_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.example_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 non-distributed, 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 non-distributed database are discussed in the *Oracle Database Concepts*. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

The database **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 34, "Distributed Transactions Concepts" for more information about the Oracle Database 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
`SELECT * FROM joan.dept@sales.example.com`	`dept`	`sales`	`example.com`
`SELECT * FROM emp@mktg.us.example.com`	`emp`	`mktg`	`us.example.com`

Whenever a SQL statement includes a reference to a global object name, the database 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.example.com;
```

The database searches for a database link called `orders.us.example.com`. The database performs this operation to determine the path to the specified remote database.

The database 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 a directory 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.example.com;
```

In this case, both the database name (`prod1`) and domain components (`us.example.com`) are specified, so the database searches for private, public, and global database links. The database 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, the database 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), the database 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@locdb
SELECT * FROM scott.emp@orders;
```

If the network domain for `locdb` is `us.example.com`, then the database appends this domain to `orders` to construct the complete global database name of `orders.us.example.com`. The database searches for database links that match only the constructed global name. If a matching link is not found, the database 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 the database 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@locdb
SELECT * from scott.emp;
```

Because the second statement does not specify a global database name using a database link connect string, the database does not search for database links.

Terminating the Search for Name Resolution

The database does not necessarily stop searching for matching database links when it finds the first match. The database 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:

User Operation	Database Response	Example
Do *not* 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` *`password`* `USING 'prod'`

User Operation	Database Response	Example
Specify the `CONNECT TO CURRENT_USER` clause	Uses a current user database link	`CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'`
Do *not* 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, the database returns an error.	`CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER`

After the database 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, the database 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:

Type of Link Specified	Location of Object Resolution
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 the database 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 the database 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.example.com`.
- The local database is named `hq.division3.example.com`.
- A directory 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@hq

CREATE PUBLIC DATABASE LINK sales.division3.example.com
CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Later, JWARD connects and issues the following statements:

```
CONNECT jward@hq

CREATE DATABASE LINK sales.division3.example.com
  CONNECT TO tsmith IDENTIFIED BY radio;

UPDATE tsmith.emp@sales.division3.example.com
  SET deptno = 40
  WHERE deptno = 10;
```

The database processes the final statement as follows:

1. The database 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. The database finds a matching private database link in the schema jward. Nevertheless, the private database link jward.sales.division3.example.com does not indicate a complete path to the remote sales database, only a remote account. Therefore, the database now searches for a matching public database link.
3. The database finds the public database link in scott's schema. From this public database link, the database takes the service name dbstring.
4. Combined with the remote account taken from the matching private fixed user database link, the database 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. The database 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 the database 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.example.com.
- The local database is named hq.division3.example.com.
- A directory 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 31-28 is already created on local database hq:

  ```
  CREATE PUBLIC DATABASE LINK sales.division3.example.com
    CONNECT TO guest IDENTIFIED BY network
    USING 'dbstring';
  ```

Consider the following statements issued at local database hq:

```
CONNECT scott@hq

CREATE DATABASE LINK sales.division3.example.com;

DELETE FROM emp@sales
  WHERE empno = 4299;
```

The database processes the final DELETE statement as follows:

1. The database 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.example.com
     WHERE empno = 4299;
   ```

2. The database searches the local database for a database link with a matching name.
3. The database finds a matching *private* connected user link in the schema scott, but the private database link indicates no path at all. The database 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.example.com
     CONNECT TO guest IDENTIFIED BY network
     USING 'dbstring';
   ```

4. The database takes the database net service name dbstring from the public database link. At this point, the database has determined a complete path.
5. The database connects to the remote database as scott/*password* 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 the database stores the definition of the object without expanding the global object name. If the name is partial, however, the database expands the name using the domain of the local database name.

The following table explains when the database completes the expansion of a partial global object name for views, synonyms, and program units:

User Operation	Database Response
Create a view	Does *not* expand partial global names. The data dictionary stores the exact text of the defining query. Instead, the database 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. However, synonyms do not expand database link names at run time, so they do not change.

Scenarios for Global Name Changes

For example, consider two databases named `sales.uk.example.com` and `hq.uk.example.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;
```

The database expands the `employee` synonym definition and stores it as:

```
scott.emp@hr.uk.example.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:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` becomes `hq.us.example.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.example.com`	`SELECT * FROM scott.emp@hr.us.example.com`
`SELECT * FROM employee`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.uk.example.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:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` is not changed

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.example.com`	`SELECT * FROM scott.emp@hr.us.example.com`
`SELECT * FROM employee`	`SELECT * FROM scott.emp@hr.uk.example.com`	`SELECT * FROM scott.emp@hr.uk.example.com`

In this case, the defining query of the `employee_names` view expands to a nonexistent global database name. However, the `employee` synonym continues to reference the correct database, `hq.uk.example.com`.

Distributed Database Application Development

Application development in a distributed system raises issues that are not applicable in a non-distributed 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 33, "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 Database 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 Database 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.example_auto.com;
CREATE PUBLIC SYNONYM dept
  FOR scott.dept@sales.us.americas.example_auto.com;
```

Now, rather than access the remote tables with a query such as:

```
SELECT ename, dname
  FROM scott.emp@sales.us.americas.example_auto.com e,
       scott.dept@sales.us.americas.example_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

The Oracle Database 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 non-distributed 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.
- The database 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 the database, 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

The database also provide many features to transparently replicate data among the nodes of the system. For more information about Oracle Database replication features, see *Oracle Database Advanced 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.example_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 Database 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 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 Database processes the data and how it accesses the data.

> **See Also:** "Using Cost-Based Optimization" on page 33-3 for more information about cost-based optimization

Character Set Support for Distributed Environments

Oracle Database supports environments in which clients, Oracle Database servers, and non-Oracle Database servers use different character sets. `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
`NLS_LANG` (environment variable)	Client/Server	Client

`NLS_LANGUAGE`
`NLS_CHARACTERSET`
`NLS_TERRITORY`
	Client/Server
	Not Heterogeneous Distributed
	Heterogeneous Distributed
		Oracle Database server

Parameters	Environment	Defined For
HS_LANGUAGE	Heterogeneous Distributed	Non-Oracle Database server Transparent gateway
NLS_NCHAR (environment variable) HS_NLS_NCHAR	Heterogeneous Distributed	Oracle Database server Transparent gateway

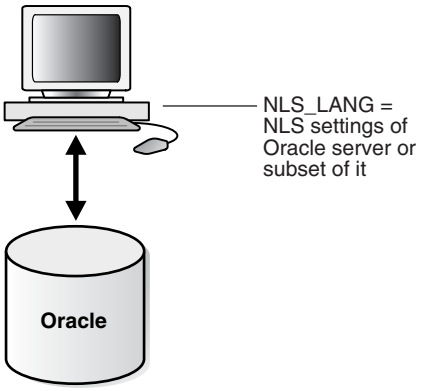
See Also:

- *Oracle Database Globalization Support Guide* for information about NLS parameters
- *Oracle Database Heterogeneous Connectivity User'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 31–6.

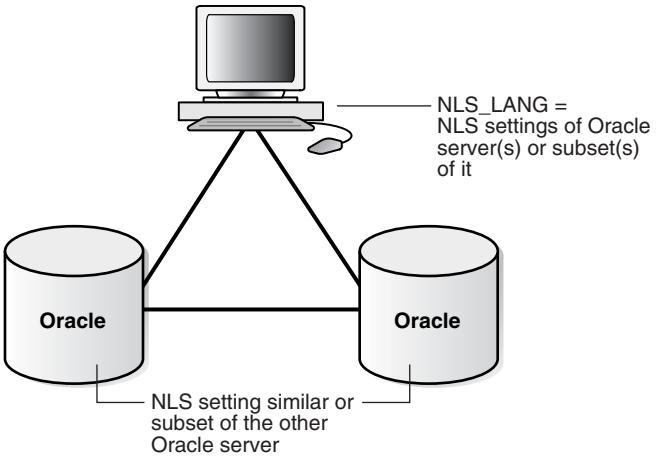
Figure 31–6 NLS Parameter Settings in a Client/Server Environment



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 31–7:

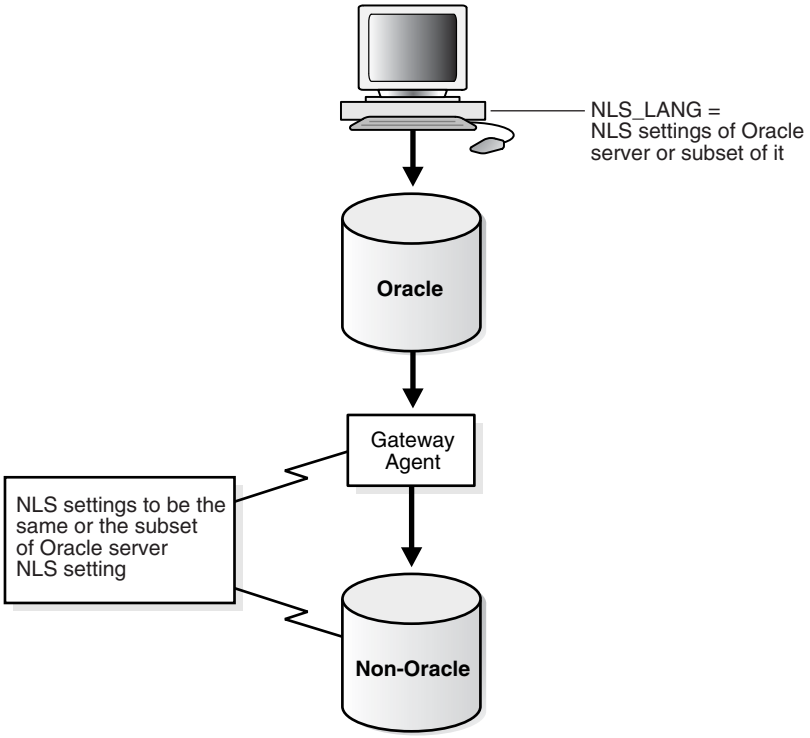
Figure 31–7 NLS Parameter Settings in a Homogeneous Environment



Heterogeneous Distributed Environment

In a heterogeneous environment, the globalization support parameter settings of the client, the transparent gateway, and the non-Oracle Database data source should be either the same or a subset of the database server character set as illustrated in Figure 31–8. Transparent gateways have full globalization support.

Figure 31–8 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 Database 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: *Oracle Database Heterogeneous Connectivity User's Guide* for more detailed information about Heterogeneous Services

32

Managing a Distributed Database

In this chapter:

- Managing Global Names in a Distributed System
- Creating Database Links
- Using Shared Database Links
- Managing Database Links
- Viewing Information About Database Links
- Creating Location Transparency
- Managing Statement Transparency
- Managing a Distributed Database: Examples

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`

Component	Parameter	Requirements	Example
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.example.com

These are examples of valid global database names:

DB_NAME	DB_DOMAIN	Global Database Name
sales	example.com	sales.example.com
sales	us.example.com	sales.us.example.com
mktg	us.example.com	mktg.us.example.com
payroll	example.org	payroll.example.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 global name. At this point, the database 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 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 the mfg 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.example.com, and hq enforces global naming, then you can create the following link names to hq:

- HQ.EXAMPLE.COM@SN1
- HQ.EXAMPLE.COM@SN2

> **See Also:** "Using Connection Qualifiers to Specify Service Names Within Link Names" on page 32-10 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 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 global name. For example, issue the following:

```
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.EXAMPLE.COM
```

Changing the Domain in a Global Database Name

Use the ALTER DATABASE statement to change the domain in a database 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.EXAMPLE.COM
   ```

2. Rename the global database name using an ALTER DATABASE statement. For example, enter:

   ```
   ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.example.com;
   ```

3. Query the GLOBAL_NAME table to check the new name. For example, enter:

   ```
   SELECT * FROM GLOBAL_NAME;

   GLOBAL_NAME
   --------------------------------------------------------------------------------
   SALES.US.EXAMPLE.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 Database resolves the names. You discover that the database 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.EXAMPLE.COM and query the GLOBAL_NAME data dictionary view to determine the current database global name:

```
CONNECT SYSTEM@sales.us.example.com
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
------------------------------------------------------------------------
SALES.US.EXAMPLE.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.EXAMPLE.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';
```

   The database 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 the database uses to resolve the partially specified global database name:

```
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.EXAMPLE.COM
```

   This result indicates that the domain part of the global database name of the local database is us.example.com. The database 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.example.com:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.example.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
------------------------------------------------------------------------
SALES.JP.EXAMPLE.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.EXAMPLE.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 the database resolves the domain part of the global database name of supply:

```
CREATE DATABASE LINK supply USING 'supply';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.EXAMPLE.COM
SUPPLY.JP.EXAMPLE.COM
```

This result indicates that the database resolves the partially specified link name by using the domain jp.example.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. The database 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.EXAMPLE.COM domain, not the JP.EXAMPLE.COM domain. Consequently, you rename the global database name as follows:

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.example.com;
SELECT * FROM GLOBAL_NAME;

GLOBAL_NAME
--------------------------------------------------------------------------------
SALES.ASIA.JP.EXAMPLE.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.EXAMPLE.COM
```

The result indicates that the domain setting in the parameter file is 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 the database resolves the partially-specified global name:

```
CREATE DATABASE LINK warehouse USING 'warehouse';
SELECT DB_LINK FROM USER_DB_LINKS;

DB_LINK
------------------
HQ.US.EXAMPLE.COM
SUPPLY.JP.EXAMPLE.COM
WAREHOUSE.ASIA.JP.EXAMPLE.COM
```

Again, you see that the database 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: *Oracle 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 lets you 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:

SQL Statement	Result
`CREATE DATABASE LINK supply.us.example.com;`	A private link using the global database name to the remote `supply` database. The link uses the userid/password of the connected user. So if `scott` (identified by *password*) uses the link in a query, the link establishes a connection to the remote database as `scott/`*password*.
`CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY` *password* `USING 'us_supply';`	A private fixed user link called link_2 to the database with service name us_supply. The link connects to the remote database with the userid/password of `jane/`*password* regardless of the connected user.
`CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';`	A private link called `link_1` to the database with service name `us_supply`. The link uses the userid/password of the current user to log onto the remote database. **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 31-11). Current user links are part of the Oracle Advanced Security option.

See Also: *Oracle Database SQL Language 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:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.example.com;`	A public link to the remote `supply` database. The link uses the userid/password of the connected user. So if `scott` (identified by *password*) uses the link in a query, the link establishes a connection to the remote database as `scott/`*password*.
`CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_USER USING 'supply';`	A public link called `pu_link` to the database with service name `supply`. The link uses the userid/password of the current user to log onto the remote database. **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 31-11).
`CREATE PUBLIC DATABASE LINK sales.us.example.com CONNECT TO jane IDENTIFIED BY` *password*;	A public fixed user link to the remote `sales` database. The link connects to the remote database with the userid/password of `jane/`*password*.

See Also: *Oracle Database SQL Language Reference* for `CREATE PUBLIC DATABASE LINK` syntax

Creating Global Database Links

In earlier releases, you defined **global database links** in the Oracle Names server. The Oracle Names server has been deprecated. Now, you can use a directory server in which databases are identified by net service names. In this document these are what are referred to as global database links.

See the *Oracle Database Net Services Administrator's Guide* to learn how to create directory entries that act as 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 Database 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:

SQL Statement	Result
`CREATE PUBLIC DATABASE LINK supply.us.example.com CONNECT TO scott AS` *`password`*`;`	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/`*`password`*.
`CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY` *`password`* `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/`*`password`*.

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 31-11.

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.example.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: *Oracle Database SQL Language 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 Database server using TCP/IP while others use DECNET.

To facilitate such functionality, the database lets you 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.example.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.example.com@hq_1
  USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.example.com@hq_2
  USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.example.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.example.com@hq_1
```

Note that in this example, there are two @ symbols.

Using 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 31-7 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
Nonshared	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.
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 nonshared 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 nonshared 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).

Note: In a multitiered environment, there is a restriction that if you use a shared database link to connect to a remote database, then that remote database cannot link to another database with a database link that cannot be migrated. That link must use a shared server, or it must be another shared 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'];
```

Whenever you use the keyword `SHARED`, the clause `AUTHENTICATED BY` is required. The schema specified in the `AUTHENTICATED BY` clause must exist in the remote database and must be granted at least the `CREATE SESSION` privilege. The credentials of this schema can be considered the authentication method between the local database and the remote database. These credentials are required to protect the remote shared server processes from clients that masquerade as a database link user and attempt to gain unauthorized access to information.

After a connection is made with a shared database link, operations on the remote database proceed with the privileges of the `CONNECT TO` user or `CURRENT_USER`, not the `AUTHENTICATED BY` schema.

The following example creates a fixed user, shared link to database `sales`, connecting as `scott` and authenticated as `linkuser`:

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY password
AUTHENTICATED BY linkuser IDENTIFIED BY ostrich
USING 'sales';
```

See Also: *Oracle Database SQL Language Reference* for information about the `CREATE DATABASE LINK` statement

Configuring Shared Database Links

You can configure shared database links in the following ways:

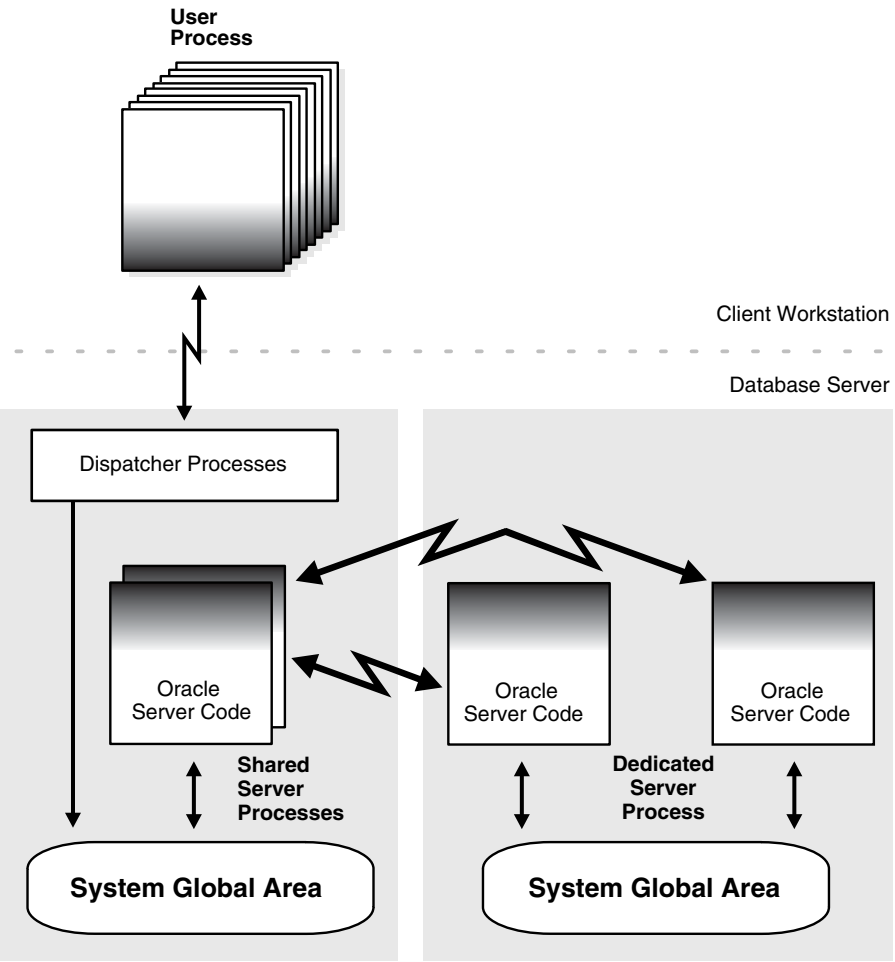
- Creating Shared Links to Dedicated Servers
- Creating Shared Links to Shared Servers

Creating Shared Links to Dedicated Servers

In the configuration illustrated in Figure 32–1, 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.

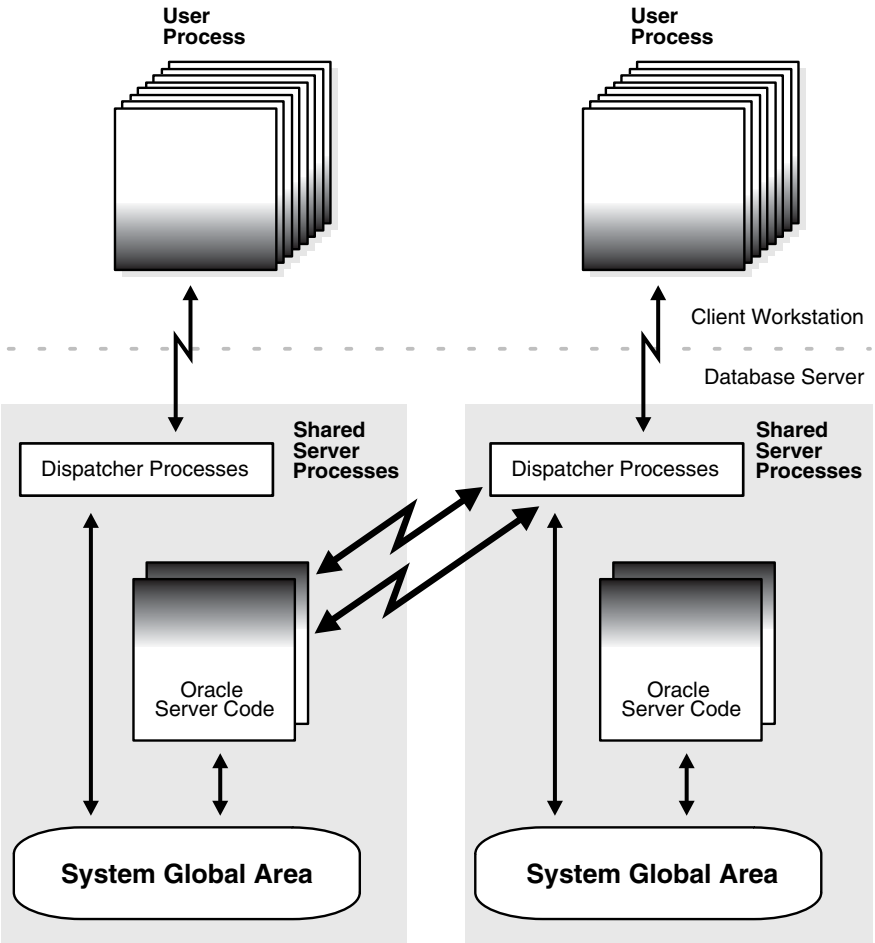
Figure 32–1 A Shared Database Link to Dedicated Server Processes



Creating Shared Links to Shared Servers

The configuration illustrated in Figure 32–2 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 32–2 Shared Database Link to Shared Server



See Also: "Shared Server Processes" on page 5-2 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.

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@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.EXAMPLE.COM
   MKTG.US.EXAMPLE.COM
   2 rows selected.
   ```

3. Drop the desired link using the `DROP DATABASE LINK` statement. For example, enter:

   ```
   DROP DATABASE LINK sales.us.example.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@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.EXAMPLE.COM
SALES.US.EXAMPLE.COM
INST2.US.EXAMPLE.COM
RMAN2.US.EXAMPLE.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.example.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 databases, 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:** *Oracle Database Reference* for more information about the `OPEN_LINKS` initialization parameter

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
`DBA_DB_LINKS`	Lists all database links in the database.
`ALL_DB_LINKS`	Lists all database links accessible to the connected user.
`USER_DB_LINKS`	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
OWNER	All except USER_*	The user who created the database link. If the link is public, then the user is listed as PUBLIC.
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_*	Not used. Maintained for backward compatibility only.
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.EXAMPLE.COM             SYS      inst1   23-JUN-99
PUBLIC     DBL1.UK.EXAMPLE.COM               BLAKE    ora51   23-JUN-99
PUBLIC     RMAN2.US.EXAMPLE.COM                       inst2   23-JUN-99
PUBLIC     DEPT.US.EXAMPLE.COM                        inst2   23-JUN-99
JANE       DBL.UK.EXAMPLE.COM                BLAKE    ora51   23-JUN-99
SCOTT      EMP.US.EXAMPLE.COM                SCOTT    inst2   23-JUN-99
6 rows selected.
```

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.EXAMPLE.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

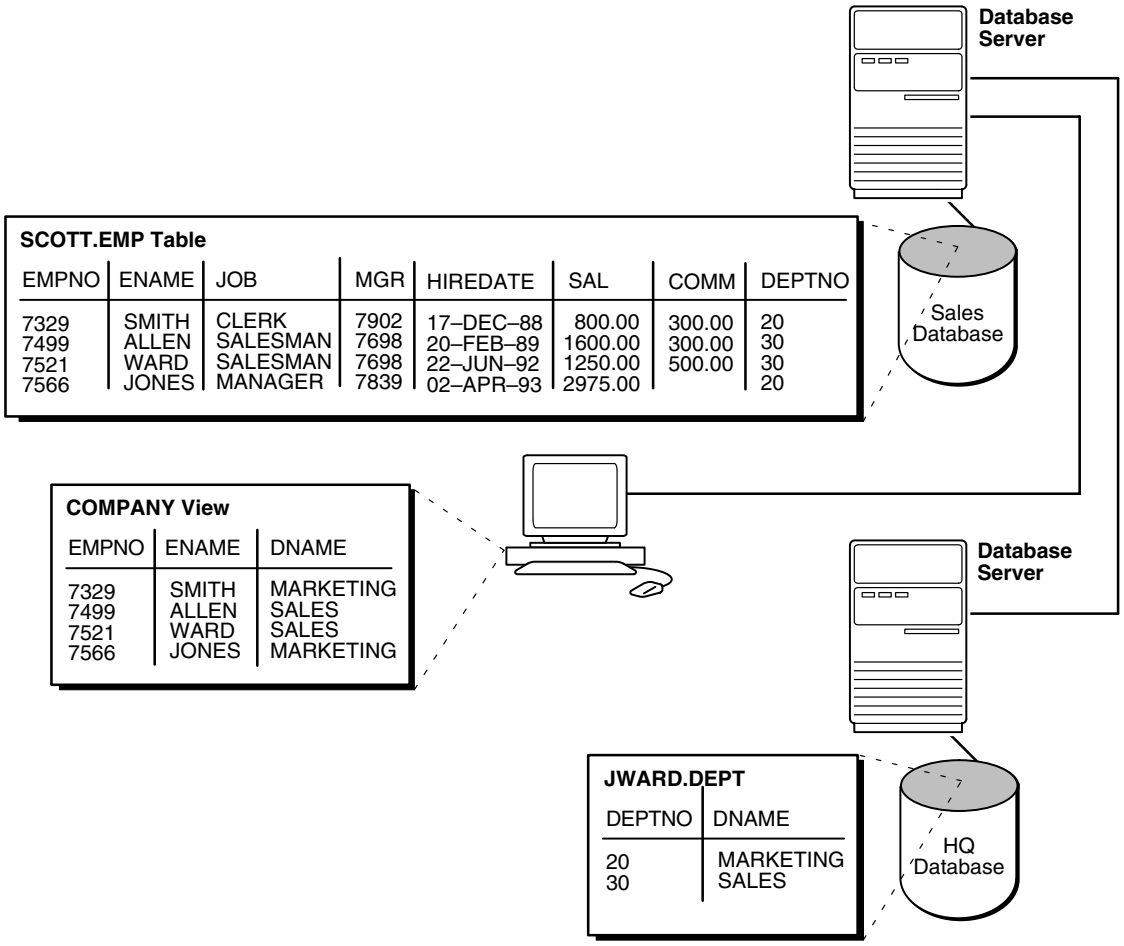
Using Views to Create Location Transparency

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.example.com b
   WHERE a.deptno = b.deptno;
```

Figure 32–3 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 non-distributed 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 schema of the creator 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 the database

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.example.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.example.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.example.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 schema of the underlying object 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.example.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.example.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@local_db

CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
  EXECUTE term_emp@hq.example.com;
END;
```

Now, assume that `scott` connects to the remote database and creates the remote procedure:

```
CONNECT scott@hq.example.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.example.com`.

Using Local Synonyms to Reference Remote Procedures

For example, `scott` connects to the local `sales.example.com` database and creates the following procedure:

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.example.com
```

```
WHERE empno = enum;
END;
```

User peggy then connects to the supply.example.com database and creates the following synonym for the procedure that scott created on the remote sales database:

```
SQL> CONNECT peggy@supply
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.example.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

The database 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.example.com e, jward.dept@hq.example.com d
  WHERE e.deptno = d.deptno;
```

In a homogeneous environment, 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.example.com
  SELECT * FROM jward.emp;
```

Restrictions for Statement Transparency:

Several restrictions apply to statement transparency.

- Data manipulation language statements that update objects on a remote non-Oracle Database system cannot reference any objects on the local Oracle

Database. For example, a statement such as the following will cause an error to be raised:

```
INSERT INTO remote_table@link as SELECT * FROM local_table;
```

- 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.
- The database 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 lets you execute DDL.
- The LIST CHAINED ROWS clause of an ANALYZE statement cannot reference remote tables.
- In a distributed database system, the database 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 Database supports the USERENV function for queries only.

- Several 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.
- There is a restriction in the Oracle Database 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 35-19 to learn how to manage this problem.

  See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_SQL package

Managing a Distributed Database: Examples

This section presents examples illustrating management of database links:

- Example 1: Creating a Public Fixed User Database Link
- Example 2: Creating a Public Fixed User Shared Database Link
- Example 3: Creating a Public Connected User Database Link
- Example 4: Creating a Public Connected User Shared Database Link
- Example 5: Creating a Public Current User Database Link

Example 1: Creating a Public Fixed User Database Link

The following statements connect to the local database as `jane` and create a public fixed user database link to database `sales` for `scott`. The database is accessed through its net service name `sldb`:

```
CONNECT jane@local

CREATE PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO scott IDENTIFIED BY password
  USING 'sldb';
```

After executing these statements, any user connected to the local database can use the `sales.division3.example.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.example.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.

Example 2: 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@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO scott IDENTIFIED BY password
  AUTHENTICATED BY scott IDENTIFIED BY password
  USING 'sldb';
```

Now, 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 preceding example, each local shared server can establish one connection to the remote server. Whenever a local shared server process must access the remote server through the `sales.division3.example.com` database link, the local shared server process reuses established network connections.

Example 3: 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@local

CREATE PUBLIC DATABASE LINK redwood
  USING 'sldb';
```

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.example.com does not have emp as a table, view, or (public) synonym, an error will be returned.

```
CONNECT fox@local

SELECT * FROM emp@redwood;
```

Example 4: 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@local

CREATE SHARED PUBLIC DATABASE LINK sales.division3.example.com
  AUTHENTICATED BY crazy IDENTIFIED BY horse
  USING 'sldb';
```

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 must access the remote server through the sales.division3.example.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.

Example 5: 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@local

CREATE PUBLIC DATABASE LINK sales.division3.example.com
  CONNECT TO CURRENT_USER
  USING 'sldb';
```

Note: To use this link, the current user must be a global user.

The consequences of this database link are as follows:

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@local_db
```

```
CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
   DELETE FROM emp@sales.division3.example.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@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 rights and privileges, see the *Oracle Database PL/SQL Language Reference*.

You can accomplish the same result by using a fixed user database link to `scott`'s remote schema.

33

Developing Applications for a Distributed Database System

In this chapter:

- Managing the Distribution of Application Data
- Controlling Connections Established by Database Links
- Maintaining Referential Integrity in a Distributed System
- Tuning Distributed Queries
- Handling Errors in Remote Procedures

> **See Also:** *Oracle Database Advanced Application Developer's Guide* for more information about application development in an Oracle Database environment

Managing the Distribution of Application 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 undo segments. To rerelease 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: *Oracle Database SQL Language 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, the database 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.

The database 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: *Oracle Database PL/SQL Language Reference* 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:

- **Collocated**

  Two or more tables located in the same database.

- **Inline view**

  A `SELECT` statement that is substituted for a table in a parent `SELECT` statement. The embedded `SELECT` statement, shown within the parentheses is an example of an inline view:

  ```
  SELECT e.empno,e.ename,d.deptno,d.dname
     FROM (SELECT empno, ename from
           emp@orc1.world) e, dept d;
  ```

- **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 recommends that you form your distributed query using collocated inline views to increase the performance of your distributed query. Oracle Database 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, remote1 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 remote1 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 preceding `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 main task of optimization 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 cannot 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 these elements, see "Using Hints" on page 33-5 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 must complete the following tasks to set up your system to take advantage of cost-based optimization:

- 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:** *Oracle Database Performance Tuning Guide* 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.

> **Note:** You must connect locally with respect to the tables when executing the DBMS_STATS procedure.
>
> You must first connect to the remote site and then execute a 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:**
>
> - *Oracle Database Performance Tuning Guide* for information about generating statistics
> - *Oracle Database 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 use 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 33-4.

The DRIVING_SITE hint lets you 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:** *Oracle Database Performance Tuning Guide* for more information about using hints

Using the NO_MERGE Hint

The NO_MERGE hint prevents the database from merging an inline view into a potentially non-collocated SQL statement (see "Using Hints" on page 33-5). 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 lets you 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:** *Oracle Database Performance Tuning Guide* 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 preceding 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:

```
@UTLXPLS.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 the database 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:** *Oracle Database PL/SQL Language Reference* for more information about PL/SQL procedures

34

Distributed Transactions Concepts

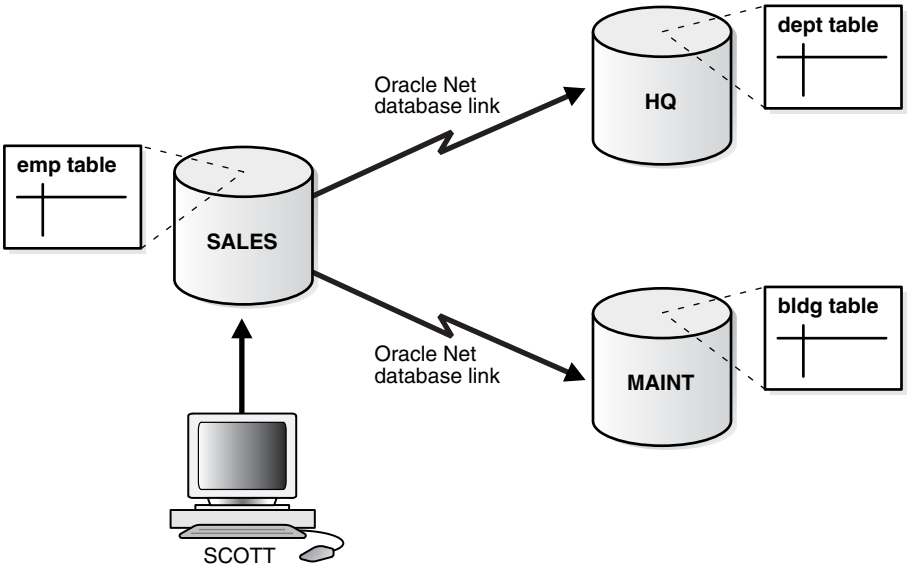
In this chapter:

- 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 34–1:

Figure 34–1 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.example.com
  SET loc = 'REDWOOD SHORES'
  WHERE deptno = 10;
```

```
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
UPDATE scott.bldg@maint.us.example.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 are the 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 are the supported transaction control statements:

- `COMMIT`
- `ROLLBACK`

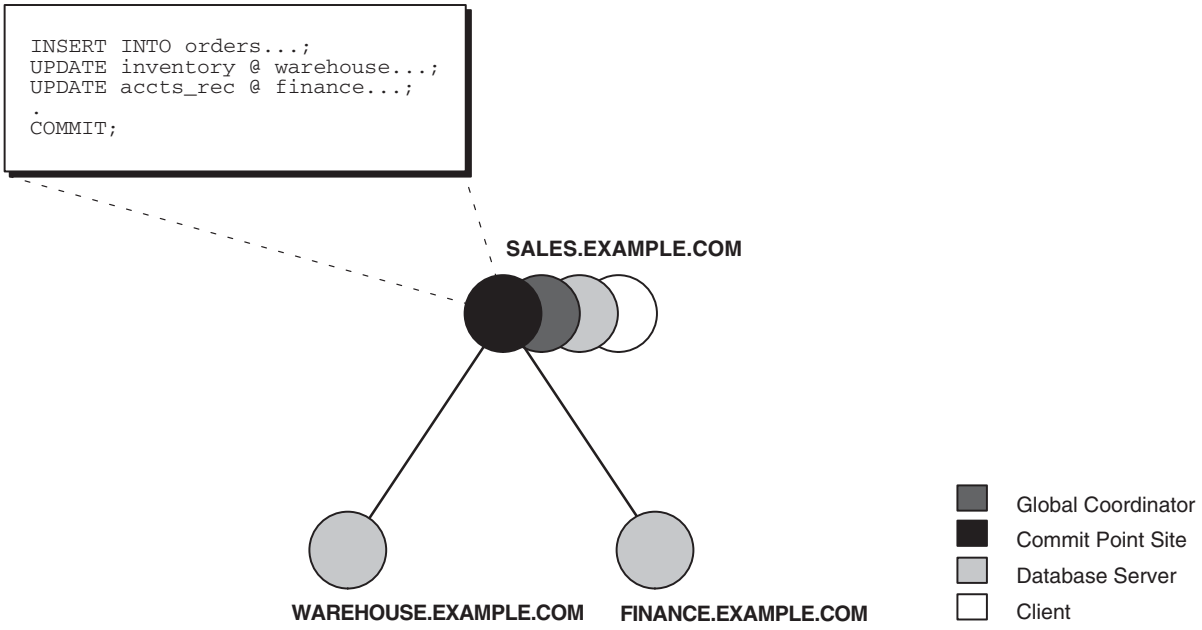
- SAVEPOINT

See Also: *Oracle Database SQL Language Reference* for more information about these SQL statements

Session Trees for Distributed Transactions

As the statements in a distributed transaction are issued, the database 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 34–2 illustrates a session tree:

Figure 34–2 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 34-5)

- 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 a database on another node. The referenced node is a database server. In Figure 34–2, 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 34–2, an application at the `sales` node initiates a distributed transaction that accesses data from the `warehouse` and `finance` nodes. Therefore, `sales.example.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 34–2, `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 34–2, the transaction issued at the node `sales` references information from the database servers `warehouse` and `finance`. Therefore, `sales.example.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 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

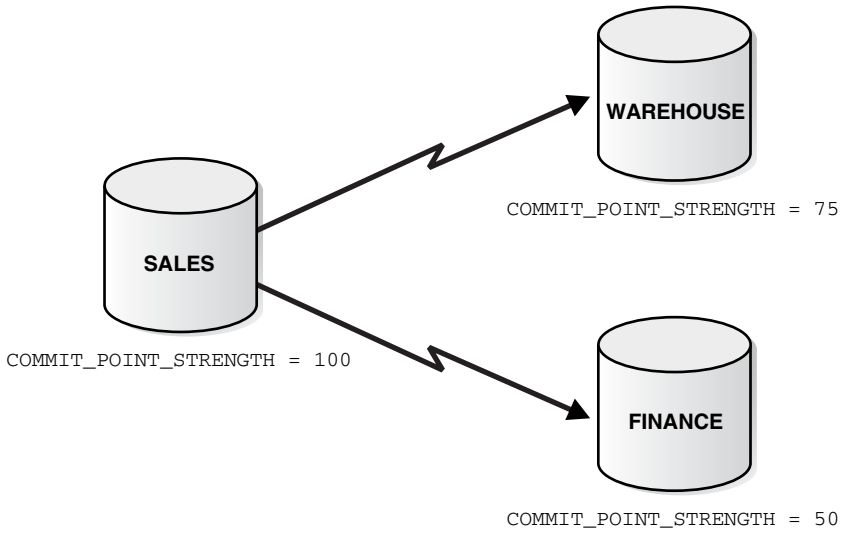
- Instructs all nodes to initiate a global rollback of the transaction if there is an abort response

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 34–3 illustrates an example of distributed system, with `sales` serving as the commit point site:

Figure 34–3 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 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

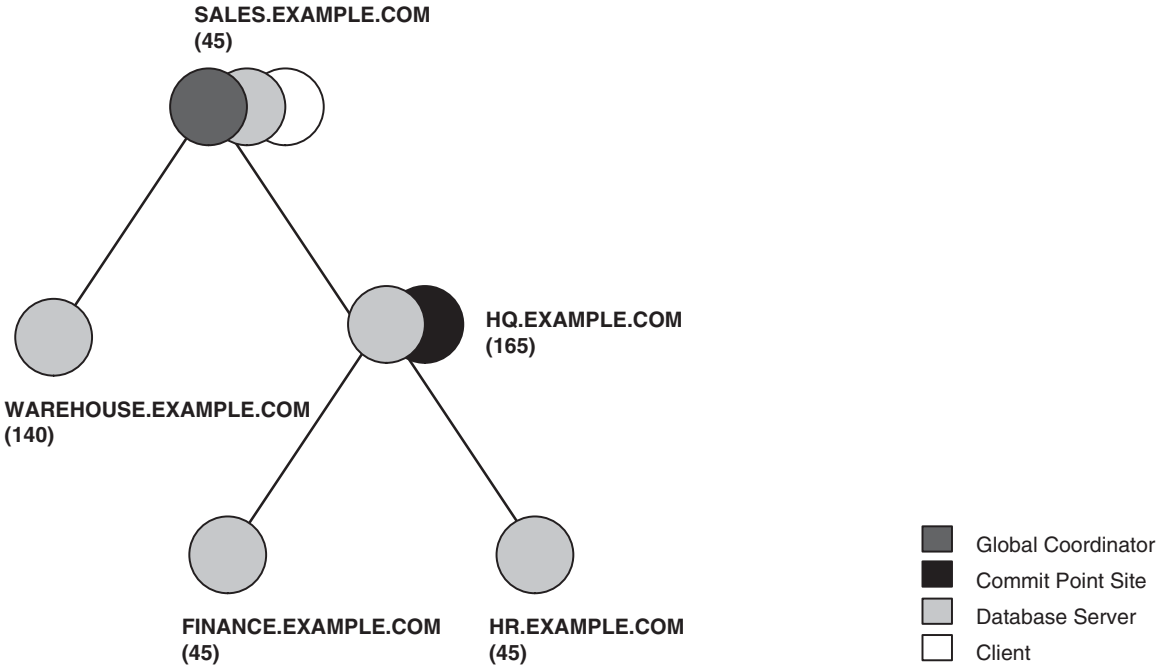
Every database server must be assigned a commit point strength. If a database server is referenced in a distributed transaction, the value of its commit point strength determines which role it plays in the two-phase commit. Specifically, the commit point strength determines whether a given node is the 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 the database 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, the database 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 34–4 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 34–4 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 the database 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, the database 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 34–4 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 35-1 to learn how to set the commit point strength of a node
- *Oracle 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 the database 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.

The database 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. The database 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 the database 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, the database 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 34-5) are told to prepare to commit. By preparing, a node:

- Records information in the 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 35-5).

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 any data on the node, 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: ■ 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 the database eliminates read-only nodes from subsequent processing.
Completely read-only with prepare phase	All of following occur: ■ No data changes. ■ Transaction is *not* 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: ■ 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 undo segments.

Note that if a distributed transaction is set to read-only, then it does not use undo segments. If many users connect to the database and their transactions are *not* set to `READ ONLY`, then they allocate undo 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 34-9).
3. The node allocates the resources it must commit the transaction if data is changed.
4. The node saves redo records corresponding to changes made by the transaction to its 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 34-9) 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 34-9).

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 34-11).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, the database commits the local portion of the distributed transaction and releases locks.
6. At each node, the database 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 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, the database 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 35-19 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 system running Oracle Database 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 system, network, or software problem is resolved. Until RECO can resolve the transaction, the data is locked for both reads and writes. The database 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

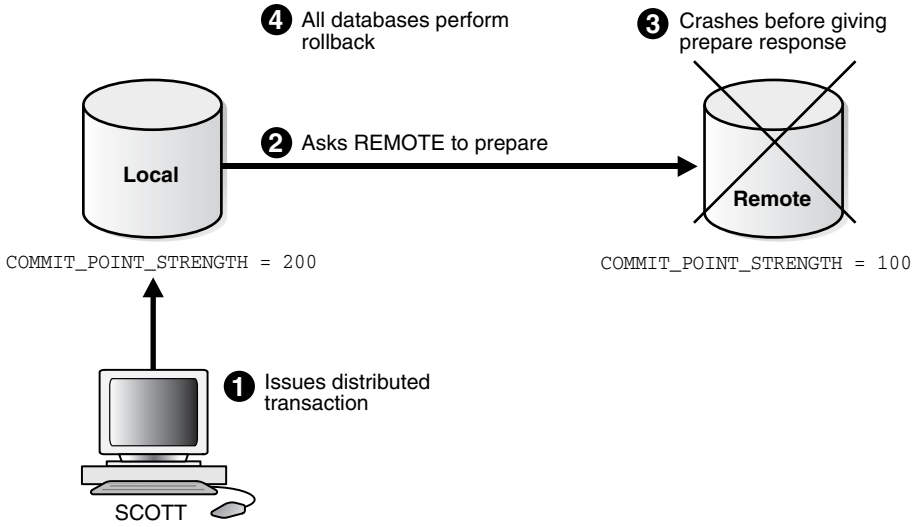
Automatic Resolution of In-Doubt Transactions

In the majority of cases, the database 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 34–5 illustrates the sequence of events when there is a failure during the prepare phase of a distributed transaction:

Figure 34–5 Failure During Prepare Phase



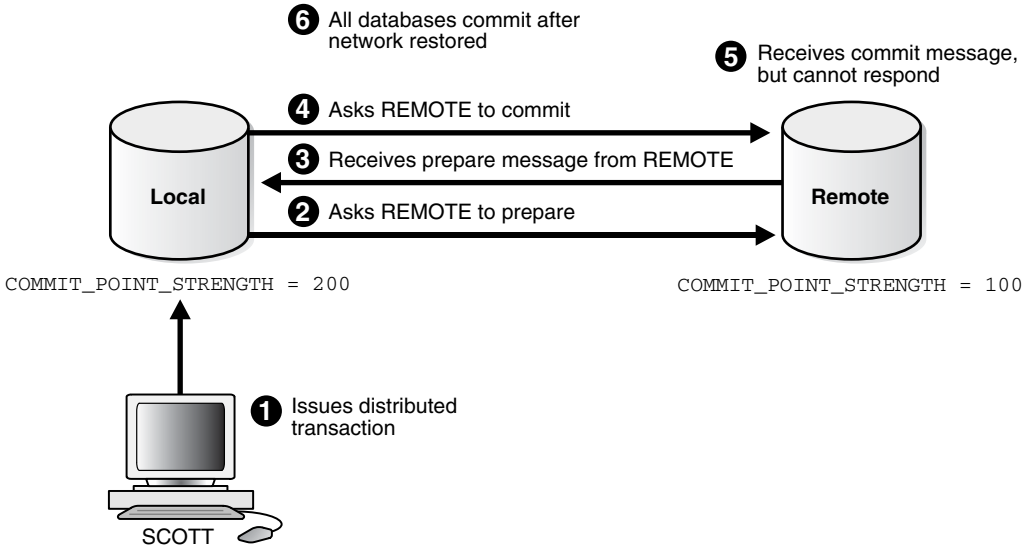
The following steps occur:

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 34–6 illustrates the sequence of events when there is a failure during the commit phase of a distributed transaction:

Figure 34–6 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 35-5 for a description of failure situations and how the database 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 undo segments.
- The cause of the system, 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 35-5
- "Manually Overriding In-Doubt Transactions" on page 35-8

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, the database records an SCN for this commit in the redo log.

The database uses SCNs to coordinate distributed transactions among different databases. For example, the database 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.example.com` and `warehouse.example.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 several SQL statements to enter the order into the `sales` database and update the inventory in the `warehouse` database:

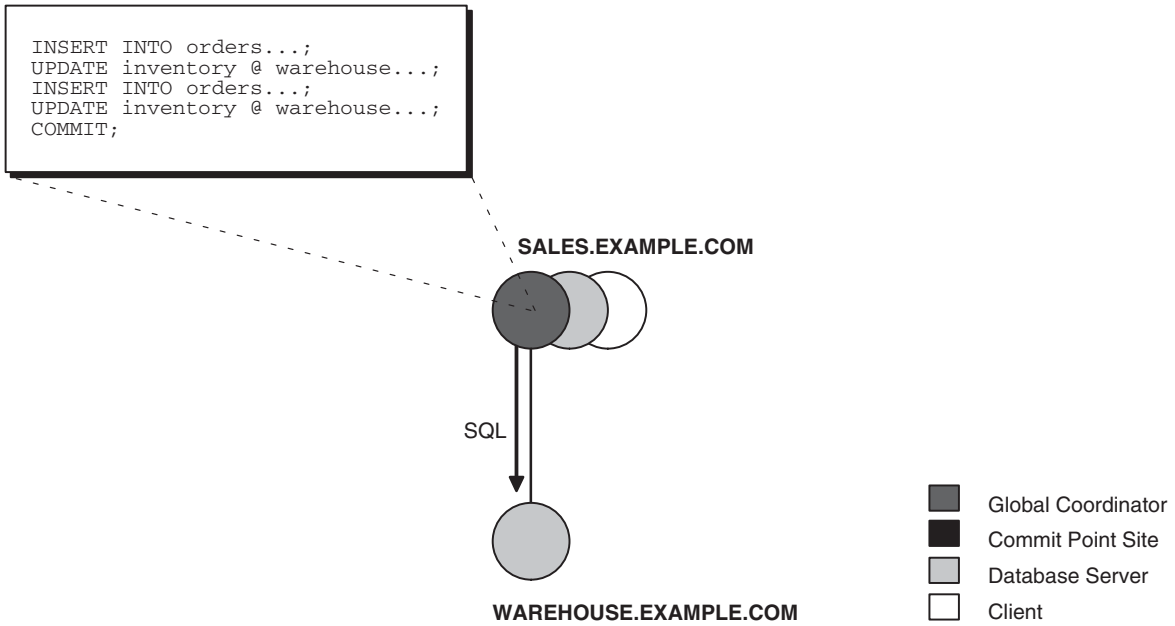
```
CONNECT scott@sales.example.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.example.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.example.com ...;
COMMIT;
```

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 34–7.

Figure 34–7 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.example.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.example.com` and `warehouse.example.com` are both database servers.
- Because `sales.example.com` updates the inventory, it is a client of `warehouse.example.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 Database Determines Commit Point Site

The database determines the commit point site immediately following the `COMMIT` statement. `sales.example.com`, the global coordinator, is determined to be the commit point site, as shown in Figure 34–8.

> **See Also:** "Commit Point Strength" on page 34-6 for more information about how the commit point site is determined

Figure 34–8 Determining the Commit Point Site

Stage 3: Global Coordinator Sends Prepare Response

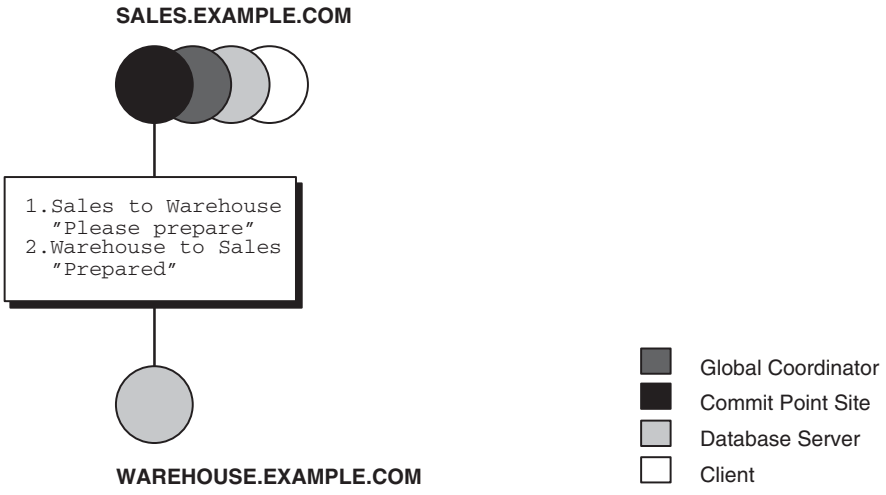
The prepare stage involves the following steps:

1. After the database 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.example.com` is the only node asked to prepare.
2. Node `warehouse.example.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.example.com` receives a prepare message because `sales.example.com` is the commit point site.
3. Node `warehouse.example.com` responds to `sales.example.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 34–9 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.example.com`, receiving acknowledgment that `warehouse.example.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.example.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 the ability of a given node 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.

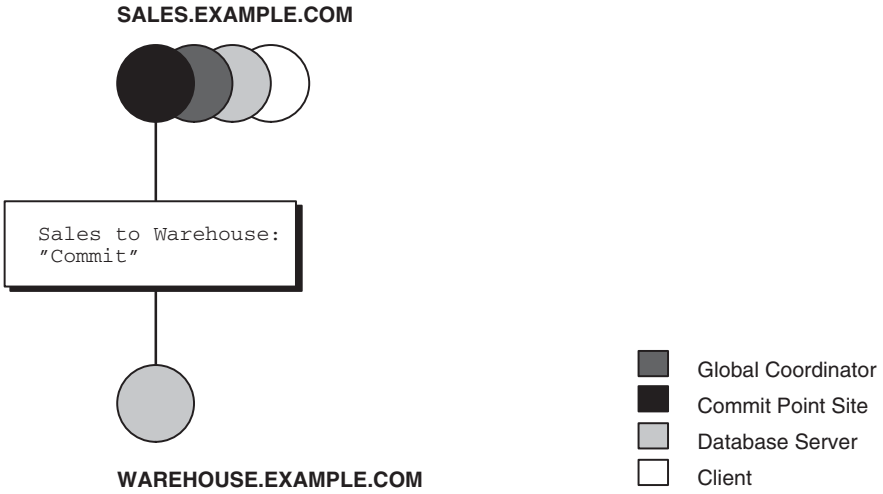
Stage 6: Global and Local Coordinators Tell All Nodes to Commit

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 34–10, `sales.example.com`, which is both the commit point site and the global coordinator, has already committed the transaction locally. `sales` now instructs `warehouse.example.com` to commit the transaction.

Figure 34–10 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 are accomplished automatically and in a fraction of a second.

35

Managing Distributed Transactions

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 non-blocking if a failure occurs during a prepare or commit phase. The `COMMIT_POINT_STRENGTH` initialization parameter determines the commit point strength of a node.

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 the database 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 transaction 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 34-5 for a conceptual overview of commit points

Naming Transactions

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 in the `TRAN_COMMENT` field of the DBA_2PC_PENDING 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
`DBA_2PC_PENDING`	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 *Oracle Database Reference*):

Table 35–1 DBA_2PC_PENDING

Column	Description
LOCAL_TRAN_ID	Local transaction identifier in the format *integer.integer.integer*. **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.
GLOBAL_TRAN_ID	Global database identifier in the format *global_db_name.db_hex_id.local_tran_id*, where *db_hex_id* 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. **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.
STATE	STATE can have the following values: ■ 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.
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 system.
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.EXAMPLE.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`	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. The view is empty until populated by an in-doubt transaction. After the transaction is resolved, the view is purged.

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 *Oracle Database Reference*):

Table 35–2 DBA_2PC_NEIGHBORS

Column	Description
`LOCAL_TRAN_ID`	Local transaction identifier with the format *integer.integer.integer*. **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.
`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.

Table 35–2 (Cont.) DBA_2PC_NEIGHBORS

Column	Description
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.
INTERFACE	C is a commit message; N is either a message indicating a prepared state or a request for a read-only commit. 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. 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.

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@hq.example.com AS SYSDBA
SQL> @neighbors_script

LOCAL_TRAN_ID IN_OUT DATABASE                  DBUSER_OWNER    INT
------------- ------ ------------------------- --------------- ---
1.15.870      out    SALES.EXAMPLE.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 system running Oracle Database 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 34-11 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-02053: 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 preceding 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 the database 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 conditions 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 undo segment from being used by other transactions. The first portion of the local transaction ID of an in-doubt distributed transaction corresponds to the ID of the undo segment, as listed by the data dictionary view `DBA_2PC_PENDING`.
- 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 decide 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 none of these conditions apply, *always* allow the automatic recovery features of the database to complete the transaction. If any of these conditions 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, the comment of an in-doubt distributed transaction 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 35-2

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 the row for this transaction.

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:

User Committing the Transaction	Privilege Required
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 lets you 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.EXAMPLE.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.EXAMPLE.COM.55d1c563.1.93.29', 829381993;
```

> **See Also:** *Oracle Database SQL Language 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:

User Committing the Transaction	Privilege Required
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:** *Oracle Database SQL Language 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.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TRANSACTION` package

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

Determining When to Use DBMS_TRANSACTION

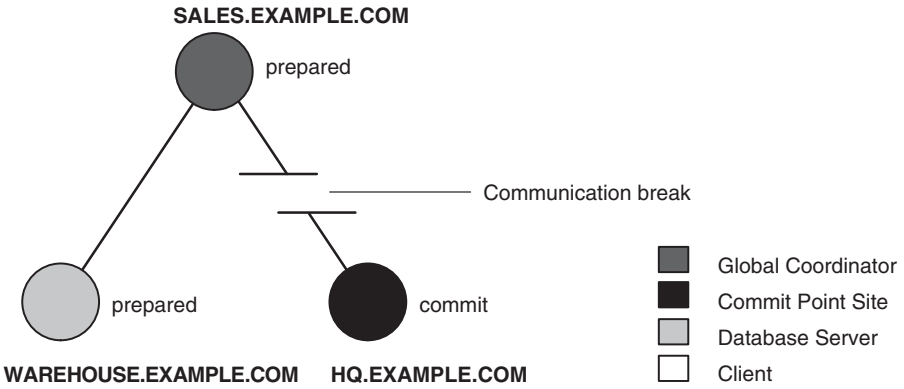
The following tables indicates what the various states indicate about the distributed transaction what the administrator's action should be:

STATE Column	State of Global Transaction	State of Local Transaction	Normal Action	Alternative Action
Collecting	Rolled back	Rolled back	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Committed	Committed	Committed	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Prepared	Unknown	Prepared	None	Force commit or rollback
Forced commit	Unknown	Committed	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Forced rollback	Unknown	Rolled back	None	`PURGE_LOST_DB_ENTRY` (only if autorecovery cannot resolve transaction)
Forced commit	Mixed	Committed	Manually remove inconsistencies then use `PURGE_MIXED`	-
Forced rollback	Mixed	Rolled back	Manually remove inconsistencies then use `PURGE_MIXED`	-

Manually Committing an In-Doubt Transaction: Example

Figure 35–1, 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 confirmation of the commit point site 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 35–1 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@warehouse.example.com AS SYSDBA
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

The database returns the following information:

```
Column Name            Value
---------------------- -------------------------------------
LOCAL_TRAN_ID          1.21.17
GLOBAL_TRAN_ID         SALES.EXAMPLE.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.hhhhhhhh.local_transaction_id

where:

- *global_database_name* is the database name of the global coordinator.
- *hhhhhhhh* 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 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 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@warehouse.example.com AS SYSDBA
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.EXAMPLE.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.EXAMPLE.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@sales.example.com AS SYSDBA
SQL> SELECT * FROM DBA_2PC_PENDING
   > WHERE GLOBAL_TRAN_ID = 'SALES.EXAMPLE.COM.55d1c563.1.93.29';

Column Name            Value
---------------------- --------------------------------------
LOCAL_TRAN_ID          1.93.29
GLOBAL_TRAN_ID         SALES.EXAMPLE.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.EXAMPLE.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.EXAMPLE.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.EXAMPLE.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.example.com
   WHERE GLOBAL_TRAN_ID = 'SALES.EXAMPLE.COM.55d1c563.1.93.29';

Column Name           Value
--------------------- --------------------------------------
LOCAL_TRAN_ID         1.45.13
GLOBAL_TRAN_ID        SALES.EXAMPLE.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@sales.example.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.EXAMPLE.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@warehouse.example.com AS SYSDBA
SQL> COMMIT FORCE 'SALES.EXAMPLE.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 the pending transaction table of a database through the `DBA_2PC_PENDING` data dictionary view (see Table 35–1).

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 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, the database 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. The database rolls back the statement.
3. The database 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, the database issues the following error message:

```
ORA-01591: lock held by in-doubt distributed transaction identifier
```

In this case, the database 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 these 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

This section describes the features available and the steps necessary to perform such operations.

Forcing a Distributed Transaction to Fail

You can include comments in the COMMENT parameter of the COMMIT statement. To intentionally induce a failure during the two-phase commit phases of a distributed transaction, include the following comment in the COMMENT parameter:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-n';
```

where *n* is one of the following integers:

n	Effect
1	Crash commit point after collect
2	Crash non-commit-point site after collect
3	Crash before prepare (non-commit-point site)
4	Crash after prepare (non-commit-point site)
5	Crash commit point site before commit
6	Crash commit point site after commit
7	Crash non-commit-point site before commit
8	Crash non-commit-point site after commit
9	Crash commit point site before forget
10	Crash non-commit-point site before forget

For example, the following statement returns the following messages if the local commit point strength is greater than the remote commit point strength and both nodes are updated:

```
COMMIT COMMENT 'ORA-2PC-CRASH-TEST-7';

ORA-02054: transaction 1.93.29 in-doubt
ORA-02059: ORA_CRASH_TEST_7 in commit comment
```

At this point, the in-doubt distributed transaction appears in the DBA_2PC_PENDING view. If enabled, RECO automatically resolves the transaction.

Disabling and Enabling RECO

The RECO background process of an Oracle Database 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 the pending transaction table of each database.

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 preceding statement.

Managing Read Consistency

An important restriction exists in the Oracle Database implementation of distributed read consistency. The problem arises because each system has its own SCN, which you can view as the database 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 systems 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.

Part VI

Appendixes

Part VI contains Appendices for the Oracle Database Administrator's Guide. It contains the following sections:

- Appendix A, "Support for DBMS_JOB in Release 11gR2"

A

Support for DBMS_JOB in Release 11gR2

In this appendix:

- About DBMS_JOB
- Moving from DBMS_JOB to Oracle Scheduler

About DBMS_JOB

DBMS_JOB is a PL/SQL package that you use to schedule jobs. It is replaced by Oracle Scheduler, which is more powerful and flexible. Although Oracle recommends that you switch from DBMS_JOB to Oracle Scheduler, DBMS_JOB is still supported for backward compatibility.

Configuring DBMS_JOB

The JOB_QUEUE_PROCESSES initialization parameter specifies the maximum number of processes that can be created for the execution of jobs. Beginning with Oracle Database release 11*g*, JOB_QUEUE_PROCESSES defaults to 1000. The job coordinator process starts only as many job queue processes as are required, based on the number of jobs to run and available resources. You can set JOB_QUEUE_PROCESSES to a lower number to limit the number of job queue processes.

Setting JOB_QUEUE_PROCESSES to 0 disables DBMS_JOB jobs and DBMS_SCHEDULER jobs.

Using Both DBMS_JOB and Oracle Scheduler

DBMS_JOB and Oracle Scheduler (the Scheduler) use the same job coordinator to start job slaves. You can use the JOB_QUEUE_PROCESSES initialization parameter to limit the number job slaves for both DBMS_JOB and the Scheduler.

If JOB_QUEUE_PROCESSES is 0, both DBMS_JOB and Oracle Scheduler jobs are disabled.

See Also:

- Chapter 29, "Scheduling Jobs with Oracle Scheduler"
- "Setting Scheduler Preferences" on page 30-2
- *Oracle Database Reference* for more information about the JOB_QUEUE_PROCESSES initialization parameter

Moving from DBMS_JOB to Oracle Scheduler

This section illustrates some examples of how you can take jobs created with the DBMS_JOB package and rewrite them using Oracle Scheduler, which you configure and control with the DBMS_SCHEDULER package.

Creating a Job

An example of creating a job using DBMS_JOB is the following:

```
VARIABLE jobno NUMBER;
BEGIN
 DBMS_JOB.SUBMIT(:jobno, 'INSERT INTO employees VALUES (7935, ''SALLY'',
   ''DOGAN'', ''sally.dogan@examplecorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
    NULL, NULL, NULL);', SYSDATE, 'SYSDATE+1');
 COMMIT;
END;
/
```

An equivalent statement using DBMS_SCHEDULER is the following:

```
BEGIN
 DBMS_SCHEDULER.CREATE_JOB(
   job_name          =>  'job1',
   job_type          =>  'PLSQL_BLOCK',
   job_action        =>  'INSERT INTO employees VALUES (7935, ''SALLY'',
     ''DOGAN'', ''sally.dogan@examplecorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
      NULL, NULL, NULL);');
   start_date        =>  SYSDATE,
   repeat_interval   =>  'FREQ = DAILY; INTERVAL = 1');
END;
/
```

Altering a Job

An example of altering a job using DBMS_JOB is the following:

```
BEGIN
 DBMS_JOB.WHAT(31, 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
   ''tom.dogan@examplecorp.com'', NULL, SYSDATE,''AD_PRES'', NULL,
   NULL, NULL, NULL);');
 COMMIT;
END;
/
```

This changes the action for JOB1 to insert a different value. An equivalent statement using DBMS_SCHEDULER is the following:

```
BEGIN
 DBMS_SCHEDULER.SET_ATTRIBUTE(
   name          => 'JOB1',
   attribute     => 'job_action',
   value         => 'INSERT INTO employees VALUES (7935, ''TOM'', ''DOGAN'',
      ''tom.dogan@examplecorp.com'', NULL, SYSDATE, ''AD_PRES'', NULL,
      NULL, NULL, NULL);');
END;
/
```

Removing a Job from the Job Queue

The following example removes a job using DBMS_JOB, where 14144 is the number of the job being run:

```
BEGIN
   DBMS_JOB.REMOVE(14144);
COMMIT;
END;
/
```

Using DBMS_SCHEDULER, you would issue the following statement instead:

```
BEGIN
   DBMS_SCHEDULER.DROP_JOB('myjob1');
END;
/
```

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_SCHEDULER package
- Chapter 29, "Scheduling Jobs with Oracle Scheduler"

Index

Symbols

?, 2-14
@, 2-14

A

abort response, 34-9
  two-phase commit, 34-9
accounts
  DBA operating system account, 1-14
  users SYS and SYSTEM, 1-14
ADD LOGFILE clause
  ALTER DATABASE statement, 12-10
ADD LOGFILE MEMBER clause
  ALTER DATABASE statement, 12-11
adding
  columns, 20-29
  columns in compressed tables, 20-30
ADMIN_TABLES procedure
  creating admin table, 25-3
  DBMS_REPAIR package, 25-2
  example, 25-6, 25-7
ADMINISTER_RESOURCE_MANAGER system privilege, 27-6
administering
  the Scheduler, 30-1
administration
  distributed databases, 32-1
administrator passwords, synchronizing password file and data dictionary, 1-25
ADR
  See automatic diagnostic repository
ADR base, 9-7
ADR home, 9-7
ADRCI utility, 9-7
Advisor
  Data Repair, 9-3
  Undo, 16-6
AFTER SUSPEND system event, 19-8
AFTER SUSPEND trigger, 19-8
  example of registering, 19-10
agent
  Heterogeneous Services, definition of, 31-3
aggregate functions
  statement transparency in distributed databases, 32-23
alert log, 9-5
  about, 8-1
  size of, 8-2
  using, 8-1
  viewing, 9-20
  when written, 8-3
alert thresholds
  setting for locally managed tablespaces, 19-2
alerts
  server-generated, 8-4
  threshold-based, 8-4
  viewing, 19-3
ALL_DB_LINKS view, 32-16
allocation
  extents, 20-29
ALTER CLUSTER statement
  ALLOCATE EXTENT clause, 22-6
  using for hash clusters, 23-9
  using for index clusters, 22-6
ALTER DATABASE ADD LOGFILE statement
  using Oracle Managed Files, 17-16
ALTER DATABASE statement
  ADD LOGFILE clause, 12-10
  ADD LOGFILE MEMBER clause, 12-11
  ARCHIVELOG clause, 13-4
  CLEAR LOGFILE clause, 12-15
  CLEAR UNARCHIVED LOGFILE clause, 12-5
  database partially available to users, 3-9
  DATAFILE...OFFLINE DROP clause, 15-7
  datafiles online or offline, 15-8
  DROP LOGFILE clause, 12-13
  DROP LOGFILE MEMBER clause, 12-14
  MOUNT clause, 3-9
  NOARCHIVELOG clause, 13-4
  OPEN clause, 3-10
  READ ONLY clause, 3-10
  RENAME FILE clause, 15-10
  tempfiles online or offline, 15-8
  UNRECOVERABLE DATAFILE clause, 12-15
ALTER INDEX statement
  COALESCE clause, 21-7
  MONITORING USAGE clause, 21-19
ALTER SEQUENCE statement, 24-13
ALTER SESSION
  Enabling resumable space allocation, 19-7

ALTER SESSION statement
ADVISE clause, 35-7
CLOSE DATABASE LINK clause, 33-2
SET SQL_TRACE initialization parameter, 8-3
setting time zone, 2-22
ALTER SYSTEM statement
ARCHIVE LOG ALL clause, 13-5
DISABLE DISTRIBUTED RECOVERY clause, 35-18
ENABLE DISTRIBUTED RECOVERY clause, 35-18
ENABLE RESTRICTED SESSION clause, 3-11
enabling Database Resource Manager, 27-35
QUIESCE RESTRICTED, 3-15
RESUME clause, 3-16
SCOPE clause for SET, 2-35
SET RESOURCE_MANAGER_PLAN, 27-36
SET SHARED_SERVERS initialization parameter, 5-8
setting initialization parameters, 2-35
SUSPEND clause, 3-16
SWITCH LOGFILE clause, 12-14
UNQUIESCE, 3-16
ALTER TABLE statement
ADD (column) clause, 20-29
ALLOCATE EXTENT clause, 20-29
DEALLOCATE UNUSED clause, 20-29
DISABLE ALL TRIGGERS clause, 18-9
DISABLE integrity constraint clause, 18-12
DROP COLUMN clause, 20-30
DROP integrity constraint clause, 18-13
DROP UNUSED COLUMNS clause, 20-31
ENABLE ALL TRIGGERS clause, 18-9
ENABLE integrity constraint clause, 18-13
external tables, 20-68
MODIFY (column) clause, 20-29
modifying index-organized table attributes, 20-61
MOVE clause, 20-28, 20-61
reasons for use, 20-27
RENAME COLUMN clause, 20-30
SET UNUSED clause, 20-31
ALTER TABLESPACE statement
adding an Oracle managed datafile, example, 17-14
adding an Oracle managed tempfile, example, 17-15
ONLINE clause, example, 14-17
READ ONLY clause, 14-18
READ WRITE clause, 14-19
RENAME DATAFILE clause, 15-9
RENAME TO clause, 14-23
taking datafiles/tempfiles online/offline, 15-8
ALTER TRIGGER statement
DISABLE clause, 18-9
ENABLE clause, 18-9
altering
(Scheduler) windows, 29-57
event schedule, 29-34
event-based job, 29-33
indexes, 21-15
job classes, 29-54
jobs, 29-15
programs, 29-23
schedules, 29-25
ANALYZE statement
CASCADE clause, 18-3
CASCADE clause, FAST option, 18-3
corruption reporting, 25-3
listing chained rows, 18-4
remote tables, 33-5
validating structure, 18-3, 25-3
analyzing schema objects, 18-2
analyzing tables
distributed processing, 33-5
APPEND hint, 20-6
application development
distributed databases, 31-32, 33-1, 33-9
application development for distributed databases, 33-1
analyzing execution plan, 33-6
database links, controlling connections, 33-1
handling errors, 33-2, 33-8
handling remote procedure errors, 33-8
managing distribution of data, 33-1
managing referential integrity constraints, 33-2
terminating remote connections, 33-2
tuning distributed queries, 33-2
tuning using collocated inline views, 33-3
using cost-based optimization, 33-3
using hints to tune queries, 33-5
ARCHIVE_LAG_TARGET initialization parameter, 12-9
archived redo logs
alternate destinations, 13-10
archiving modes, 13-4
data dictionary views, 13-14
destination availability state, controlling, 13-9
destination status, 13-9
destinations, specifying, 13-6
failed destinations and, 13-11
mandatory destinations, 13-11
minimum number of destinations, 13-11
multiplexing, 13-6
normal transmission of, 13-10
re-archiving to failed destination, 13-12
sample destination scenarios, 13-11
standby transmission of, 13-10
status information, 13-15
transmitting, 13-10
ARCHIVELOG mode, 13-3
advantages, 13-3
archiving, 13-2
automatic archiving in, 13-3
definition of, 13-3
distributed databases, 13-3
enabling, 13-4
manual archiving in, 13-3
running in, 13-3
switching to, 13-4
taking datafiles offline and online in, 15-7

archiver process
trace output (controlling), 13-13
archiver process (ARCn), 5-20
archiving
alternate destinations, 13-10
changing archiving mode, 13-4
controlling number of processes, 13-5
destination availability state, controlling, 13-9
destination failure, 13-11
destination status, 13-9
manual, 13-5
NOARCHIVELOG vs. ARCHIVELOG mode, 13-2
setting initial mode, 13-4
to failed destinations, 13-12
trace output, controlling, 13-13
viewing information on, 13-15
at-sign, 2-14
auditing
database links, 31-22
authentication
database links, 31-18
operating system, 1-20
selecting a method, 1-18
using password file, 1-21
AUTO_TASK_CONSUMER_GROUP
of Resource Manager, 26-5
AUTOEXTEND, 15-5
AUTOEXTEND clause
for bigfile tablespaces, 14-21
automatic diagnostic repository, 9-1, 9-5
in Oracle Client, 9-9
in Oracle Real Application Clusters, 9-9
structure, contents and location of, 9-7
automatic file extension, 15-5
automatic maintenance tasks
assigning to maintenance windows, 26-4
definition, 26-1
enabling and disabling, 26-3
predefined, 26-2
resource allocation, 26-6
Scheduler job names, 26-2
automatic memory management, 6-1
about, 6-4
enabling, 6-4
supported platforms, 6-23
automatic segment space management, 14-5
automatic undo management, 2-18, 16-2
migrating to, 16-11

B

background processes, 5-19
FMON, 15-17
BACKGROUND_DUMP_DEST initialization parameter, 9-5
backups
after creating new databases, 2-15
effects of archiving on, 13-2
batch jobs, authenticating users in, 2-44
bigfile tablespaces
creating, 14-7
creating temporary, 14-12
description, 14-6
setting database default, 2-21
BLANK_TRIMMING initialization parameter, 20-29
BLOB data type, 20-13
block size, redo log files, 12-7
BLOCKSIZE clause
of CREATE TABLESPACE, 14-14
buffer caches
extended buffer cache (32-bit), 6-19
multiple buffer pools, 6-16
buffer pools, 6-16
BUFFER_POOL_KEEP initialization parameter, 6-16
BUFFER_POOL_RECYCLE initialization parameter, 6-16
buffers
buffer cache in SGA, 6-15

C

CACHE option
CREATE SEQUENCE statement, 24-16
caches
buffer
multiple buffer pools, 6-16
sequence numbers, 24-16
calendaring expressions, 29-26
calendaring syntax, 29-26
calls
remote procedure, 31-34
capacity planning
space management
capacity planning, 19-32
CASCADE clause
when dropping unique or primary keys, 18-13
CATBLOCK.SQL script, 8-7
centralized user management
distributed systems, 31-19
chain condition syntax, 29-44
chain rules, 29-44
chain steps
defining, 29-43
chained rows
eliminating from table, procedure, 18-4
CHAINED_ROWS table
used by ANALYZE statement, 18-4
chains
creating, 29-42
creating and managing job, 29-41
creating jobs for, 29-48
disabling, 29-50
dropping, 29-48
dropping rules from, 29-49
enabling, 29-47
handling stalled, 29-53
monitoring running, 29-52
overview, 28-8
pausing, 29-51

running, 29-49
setting privileges, 30-2
steps
pausing, 29-51
skipping, 29-52
stopping, 29-50
stopping individual steps, 29-50
change vectors, 12-2
CHAR data type
increasing column length, 20-29
character set
choosing, 2-3
CHECK_OBJECT procedure
DBMS_REPAIR package, 25-2
example, 25-7
finding extent of corruption, 25-4
checkpoint process (CKPT), 5-20
checksums
for data blocks, 15-12
redo log blocks, 12-14
CLEAR LOGFILE clause
ALTER DATABASE statement, 12-15
clearing redo log files, 12-5, 12-15
client/server architectures
distributed databases, 31-4
globalization support, 31-35
cloning
a database, 1-6
an Oracle home, 1-6
CLOSE DATABASE LINK clause
ALTER SESSION statement, 33-2
closing database links, 32-14
closing windows, 29-58
clusters
about, 22-1
allocating extents, 22-6
altering, 22-6
analyzing, 18-2
cluster indexes, 22-7
cluster keys, 22-1, 22-3, 22-4
clustered tables, 22-1, 22-3, 22-5, 22-6, 22-8
columns for cluster key, 22-3
creating, 22-4
data dictionary views reference, 22-8
deallocating extents, 22-6
dropping, 22-7
estimating space, 22-3, 22-4
guidelines for managing, 22-2
hash clusters, 23-1
location, 22-4
privileges, 22-4, 22-6, 22-8
selecting tables, 22-3
single-table hash clusters, 23-6
truncating, 18-5
validating structure, 18-3
coalescing indexes
costs, 21-7
cold backup
performing with a detached Oracle Scheduler job, 29-11
collocated inline views
tuning distributed queries, 33-3
column encryption, 2-43
columns
adding, 20-29
adding to compressed table, 20-30
displaying information about, 20-72
dropping, 20-30, 20-31
dropping in compressed tables, 20-31
encrypted, 20-8, 20-28
increasing length, 20-29
modifying definition, 20-29
renaming, 20-30
virtual, 20-2
virtual, indexing, 21-3
commands
submitting, 1-6
COMMENT statement, 20-71
comments
adding to problem activity log, 9-16
COMMIT COMMENT statement
used with distributed transactions, 35-2, 35-7
commit phase, 34-8, 34-17
in two-phase commit, 34-10, 34-11
commit point site, 34-5
commit point strength, 34-6, 35-1
determining, 34-7
distributed transactions, 34-5, 34-6
how the database determines, 34-6
commit point strength
definition, 34-6
specifying, 35-1
COMMIT statement
FORCE clause, 35-8, 35-9
forcing, 35-6
two-phase commit and, 31-26
COMMIT_POINT_STRENGTH initialization parameter, 34-6, 35-1
committing transactions
commit point site for distributed transactions, 34-5
compatibility level, 2-31
COMPATIBLE Initialization Parameter, 2-31
components
srvctl component names and abbreviations, 4-31
compressed tables
adding a column, 20-30
dropping columns in, 20-31
compression, table, 20-5
compression, tablespace, 14-8
configuring
an Oracle database, 2-1
Oracle Scheduler, 30-1
CONNECT command
starting an instance, 3-5
CONNECT INTERNAL
desupported, 1-18
CONNECT statement, SQL*Plus, 1-9
connected user database links, 32-9
advantages and disadvantages, 31-12

definition, 31-11
example, 31-14
REMOTE_OS_AUTHENT initialization parameter, 31-12
connecting
with SQL*Plus, 1-7
connection qualifiers
database links and, 32-10
connections
terminating remote, 33-2
constraints
See also integrity constraints
disabling at table creation, 18-12
distributed system application development issues, 33-2
dropping integrity constraints, 18-13
enable novalidate state, 18-11
enabling example, 18-12
enabling when violations exist, 18-11
exceptions, 18-11, 18-14
exceptions to integrity constraints, 18-14
integrity constraint states, 18-10
keeping index when disabling, 18-12
keeping index when dropping, 18-12
ORA-02055 constraint violation, 33-2
renaming, 18-13
setting at table creation, 18-11
when to disable, 18-10
control files
adding, 11-4
changing size, 11-4
conflicts with data dictionary, 11-7
creating, 11-1, 11-3, 11-5
creating as Oracle Managed Files, 17-15
creating as Oracle Managed Files, examples, 17-21
data dictionary views reference, 11-9
default name, 2-28, 11-4
dropping, 11-9
errors during creation, 11-7
guidelines for, 11-2
importance of multiplexed, 11-2
initial creation, 11-3
location of, 11-3
log sequence numbers, 12-3
mirroring, 2-28, 11-2
moving, 11-4
multiplexed, 11-2
names, 11-2
number of, 11-2
overwriting existing, 2-28
relocating, 11-4
renaming, 11-4
requirement of one, 11-1
size of, 11-3
specifying names before database creation, 2-28
troubleshooting, 11-7
unavailable during startup, 3-6
CONTROL_FILES initialization parameter
overwriting existing control files, 2-28
specifying file names, 11-2
when creating a database, 2-28, 11-3
CONTROLFILE REUSE clause, 2-28
copying jobs, 29-20
coraenv and oraenv, 1-8
core files, 9-6
corruption
repairing data block, 25-1
cost-based optimization, 33-3
distributed databases, 31-34
hints, 33-5
using for distributed queries, 33-3
CREATE BIGFILE TABLESPACE statement, 14-7
CREATE BIGFILE TEMPORARY TABLESPACE statement, 14-12
CREATE CLUSTER statement
creating clusters, 22-5
example, 22-5
for hash clusters, 23-2
HASH IS clause, 23-3, 23-6
HASHKEYS clause, 23-3, 23-7
SIZE clause, 23-7
CREATE CONTROLFILE statement
about, 11-5
checking for inconsistencies, 11-7
creating as Oracle Managed Files, examples, 17-15, 17-21
NORESETLOGS clause, 11-6
Oracle Managed Files, using, 17-15
RESETLOGS clause, 11-6
CREATE DATABASE LINK statement, 32-7
CREATE DATABASE statement
CONTROLFILE REUSE clause, 11-4
DEFAULT TEMPORARY TABLESPACE clause, 2-13, 2-18
example of database creation, 2-11
EXTENT MANAGEMENT LOCAL clause, 2-17
MAXLOGFILES parameter, 12-8
MAXLOGMEMBERS parameter, 12-8
password for SYS, 2-16
password for SYSTEM, 2-16
setting time zone, 2-22
specifying FORCE LOGGING, 2-23
SYSAUX DATAFILE clause, 2-12
UNDO TABLESPACE clause, 2-13, 2-18
used to create an undo tablespace, 16-8
using Oracle Managed Files, 17-7
using Oracle Managed Files, examples, 17-10, 17-19, 17-22
CREATE INDEX statement
NOLOGGING, 21-5
ON CLUSTER clause, 22-6
using, 21-9
with a constraint, 21-10
CREATE PFILE FROM MEMORY command, 2-37
CREATE SCHEMA statement
multiple tables and views, 18-1
CREATE SEQUENCE statement, 24-13
CACHE option, 24-16
examples, 24-16

NOCACHE option, 24-16
CREATE SPFILE statement, 2-33
CREATE SYNONYM statement, 24-17
CREATE TABLE statement
AS SELECT clause, 20-4, 20-15
CLUSTER clause, 22-5
COMPRESS clause, 20-60
creating temporary table, 20-14
example of, 20-13
INCLUDING clause, 20-58
index-organized tables, 20-56
MONITORING clause, 20-26
NOLOGGING clause, 20-4
ORGANIZATION EXTERNAL clause, 20-66
parallelizing, 20-15
PCTTHRESHOLD clause, 20-58
TABLESPACE clause, specifying, 20-4
CREATE TABLESPACE statement
BLOCKSIZE CLAUSE, using, 14-14
FORCE LOGGING clause, using, 14-15
using Oracle Managed Files, 17-12
using Oracle Managed Files, examples, 17-13
CREATE TEMPORARY TABLESPACE statement, 14-12
using Oracle Managed Files, 17-14
using Oracle managed files, example, 17-14
CREATE UNDO TABLESPACE statement
using Oracle Managed Files, 17-12
using Oracle Managed Files, example, 17-13
using to create an undo tablespace, 16-8
CREATE UNIQUE INDEX statement
using, 21-9
CREATE VIEW statement
about, 24-2
OR REPLACE clause, 24-4
WITH CHECK OPTION, 24-2, 24-5
CREATE_SIMPLE_PLAN procedure
Database Resource Manager, 27-24
creating
an Oracle database, 2-1
chains, 29-42
control files, 11-3
database services, 2-41
databases, 2-2
event schedule, 29-33
event-based job, 29-32
indexes, 21-8
after inserting table data, 21-2
associated with integrity constraints, 21-9
NOLOGGING, 21-5
online, 21-11
USING INDEX clause, 21-10
job classes, 29-54
jobs, 29-2
programs, 29-21
Scheduler windows, 29-57
schedules, 29-25
sequences, 24-16
window groups, 29-61
creating database links, 32-6
connected user, 32-9
connected user scenarios, 32-25
current user, 32-9
current user scenario, 32-26
examples, 31-14
fixed user, 32-8
fixed user scenario, 32-25
obtaining necessary privileges, 32-6
private, 32-7
public, 32-7
service names within link names, 32-10
shared, 32-10
shared connected user scenario, 32-26
specifying types, 32-7
creating databases
backing up the new database, 2-15
default temporary tablespace, specifying, 2-18
example, 2-11
manually from a script, 2-2
overriding default tablespace type, 2-21
planning, 2-2
preparing to, 2-2
prerequisites for, 2-4
setting default tablespace type, 2-21
specifying bigfile tablespaces, 2-20, 2-21
UNDO TABLESPACE clause, 2-18
upgrading to a new release, 2-2
using Oracle Managed Files, 2-19, 17-7
with DBCA, 2-5
with locally managed tablespaces, 2-17
creating datafiles, 15-4
creating sequences, 24-13
creating synonyms, 24-17
creating views, 24-2
credentials, Oracle Scheduler
about, 28-7
creating, 29-6
granting privileges on, 28-7
critical errors
diagnosing, 9-1
CRSCTL utility
Oracle Restart, 4-3
current user database links
advantages and disadvantages, 31-13
cannot access in shared schema, 31-20
definition, 31-12
example, 31-14
schema independence, 31-20
CURRVAL pseudo-column, 24-14
restrictions, 24-15
cursors
and closing database links, 33-2
customize package page, accessing, 9-40
customizing an incident package, 9-39, 9-40

D

data
loading using external tables, 20-66
data block corruption

repairing, 25-1
data blocks
altering size of, 2-28
nonstandard block size, 2-29
shared in clusters, 22-1
specifying size of, 2-28
standard block size, 2-28
verifying, 15-12
data dictionary
conflicts with control files, 11-7
purging pending rows from, 35-9, 35-10
See also views, data dictionary
data encryption
distributed systems, 31-22
data manipulation language
statements allowed in distributed transactions, 31-24
Data Recovery Advisor, repairing data corruptions with, 9-31
Data Repair Advisor, 9-3
database
cloning, 1-6
creating, 2-2
creating and configuring, 2-1
creating with DBCA, 2-5
data dictionary views reference, 2-45
starting up, 3-1
database administrators
DBA role, 1-15
operating system account, 1-14
password files for, 1-19
responsibilities of, 1-1
security and privileges of, 1-14
security officer versus, 7-1
SYS and SYSTEM accounts, 1-14
task definitions, 1-3
utilities for, 1-28
database buffers
multiple buffer pools, 6-16
Database Configuration Assistant, 2-2
shared server configuration, 5-10
database destinations, Oracle Scheduler
about, 28-6
creating, 29-7
Database File System, xxxix
database jobs, Oracle Scheduler, 28-15
database links
advantages, 31-8
auditing, 31-22
authentication, 31-18
authentication without passwords, 31-19
closing, 32-14, 33-2
connected user, 31-11, 31-12, 32-9, 32-25
connections, determining open, 32-17
controlling connections, 33-1
creating, 32-6, 32-25, 32-26
creating shared, 32-12
creating, examples, 31-14
creating, scenarios, 32-24
current user, 31-12, 31-13, 32-9
data dictionary USER views, 32-16
definition, 31-6
distributed queries, 31-24
distributed transactions, 31-25
dropping, 32-15
enforcing global naming, 32-2
enterprise users and, 31-20
fixed user, 31-12, 31-13, 32-25
global, 31-11
global names, 31-8
loopback, 31-9
global object names, 31-26
handling errors, 33-2
limiting number of connections, 32-16
listing, 32-16, 35-2, 35-4
loopback, 31-9
managing, 32-14
minimizing network connections, 32-11
name resolution, 31-26
names for, 31-10
private, 31-10
public, 31-10
referential integrity in, 33-2
remote transactions, 31-24, 31-25
resolution, 31-26
restrictions, 31-16
roles on remote database, 31-16
schema objects and, 31-14
service names used within link names, 32-10
shared, 31-7, 32-11, 32-12, 32-13
shared SQL, 31-25
synonyms for schema objects, 31-15
tuning distributed queries, 33-2
tuning queries with hints, 33-5
tuning using collocated inline views, 33-3
types of links, 31-10
types of users, 31-11
users, specifying, 32-8
using cost-based optimization, 33-3
viewing, 32-16
database objects
obtaining growth trends for, 19-33
database program unit, definition, 28-1
database resident connection pooling, 5-4
advantages, 5-4
configuration parameters, 5-17
configuring the connection pool, 5-17
data dictionary views reference, 5-19
disabling, 5-16
enabling, 5-16
restrictions, 5-6
triggers, 5-5
Database Resource Manager
active session pool with queuing, 27-23
administering system privilege, 27-6
and operating system control, 27-57
automatic consumer group switching, 27-23
CREATE_SIMPLE_PLAN procedure, 27-24
data dictionary views reference, 27-60
description, 27-1

enabling, 27-35
execution time limit, 27-24
resource allocation methods, 27-27, 27-28
resource consumer groups, 27-3, 27-7, 27-27
resource plan directives, 27-3, 27-29, 27-34
resource plans, 27-3, 27-5, 27-18, 27-24, 27-35, 27-37, 27-47
undo pool, 27-24
used for quiescing a database, 3-15
validating plan schema changes, 27-33
database services
about, 2-39
controlling automatic startup of, 3-5
data dictionary views, 2-42
managing application workloads with, 2-39
database services, creating, 2-41
Database Smart Flash Cache
configuring and tuning, 6-21
overview, 6-3
database writer process
calculating checksums for data blocks, 15-12
database writer process (DBWn), 5-20
DATABASE_PROPERTIES view
rename of default temporary tablespace, 14-24
databases
administering, 1-1
administration of distributed, 32-1
altering availability, 3-9
backing up, 2-15
control files of, 11-2
default temporary tablespace, specifying, 2-18
dropping, 2-44
global database names in distributed systems, 2-27
mounting a database, 3-7
mounting to an instance, 3-9
names, about, 2-27
names, conflicts in, 2-27
opening a closed database, 3-10
planning, 1-4
planning creation, 2-2
quiescing, 3-14
read-only, opening, 3-10
recovery, 3-9
renaming, 11-5, 11-6
restricting access, 3-11
resuming, 3-16
shutting down, 3-11
specifying control files, 2-28
suspending, 3-16
undo management, 2-18
upgrading, 2-2
with locally managed tablespaces, 2-17
datafile headers
when renaming tablespaces, 14-23
datafiles
adding to a tablespace, 15-4
bringing online and offline, 15-6
checking associated tablespaces, 14-48
copying using database, 15-12
creating, 15-4
creating Oracle Managed Files, 17-5, 17-17
data dictionary views reference, 15-24
database administrators access, 1-14
default directory, 15-5
definition, 15-1
deleting, 14-24
dropping, 15-7, 15-11
dropping Oracle managed, 17-18
file numbers, 15-1
fully specifying filenames, 15-5
guidelines for managing, 15-1
headers when renaming tablespaces, 14-23
identifying OS filenames, 15-10
location, 15-4
mapping files to physical devices, 15-15
minimum number of, 15-2
MISSING, 11-7
online, 15-7
Oracle managed, 17-1
relocating, 15-8
renaming, 15-8
reusing, 15-5
size of, 15-3
statements to create, 15-4
storing separately from redo log files, 15-4
unavailable when database is opened, 3-6
verifying data blocks, 15-12
DB_BLOCK_CHECKING initialization parameter, 25-3, 25-4
DB_BLOCK_CHECKSUM initialization parameter, 15-12
enabling redo block checking with, 12-14
DB_BLOCK_SIZE initialization parameter, 6-15
and nonstandard block sizes, 14-14
setting, 2-28
DB_CACHE_SIZE initialization parameter, 6-16
specifying multiple block sizes, 14-14
DB_CREATE_FILE_DEST initialization parameter, 17-4
setting, 17-4
DB_CREATE_ONLINE_LOG_DEST_n initialization parameter, 17-4
setting, 17-5
DB_DOMAIN initialization parameter
setting for database creation, 2-26, 2-27
DB_FILES initialization parameter
determining value for, 15-2
DB_KEEP_CACHE_SIZE initialization parameter, 6-16
DB_NAME initialization parameter
setting before database creation, 2-26
DB_nK_CACHE_SIZE initialization parameter, 6-16
specifying multiple block sizes, 14-14
using with transportable tablespaces, 14-43
DB_RECOVERY_FILE_DEST initialization parameter, 17-4
setting, 17-5
DB_RECYCLY_CACHE_SIZE initialization parameter, 6-16

DB_UNRECOVERABLE_SCN_TRACKING
initialization parameter, ii-xxxviii, 20-21
DBA
See database administrators
DBA role, 1-15
DBA_2PC_NEIGHBORS view, 35-4
using to trace session tree, 35-4
DBA_2PC_PENDING view, 35-2, 35-9, 35-16
using to list in-doubt transactions, 35-2
DBA_DB_LINKS view, 32-16
DBA_RESUMABLE view, 19-9
DBA_UNDO_EXTENTS view
undo tablespace extents, 16-11
DBCA
See Database Configuration Assistant
DBFS, xxxix
links, xxxix
DBMS_FILE_TRANSFER package
copying datafiles, 15-12
DBMS_JOB
about, A-1
moving jobs to Oracle Scheduler, A-2
DBMS_METADATA package
GET_DDL function, 18-25
using for object definition, 18-25
DBMS_REDEFINITION package
performing online redefinition with, 20-34
required privileges, 20-48
DBMS_REPAIR
logical corruptions, 25-4
DBMS_REPAIR package
examples, 25-5
limitations, 25-2
procedures, 25-2
using, 25-2, 25-10
DBMS_RESOURCE_MANAGER package, 27-3, 27-7, 27-12
procedures (table of), 27-6
DBMS_RESOURCE_MANAGER_PRIVS package, 27-7
procedures (table of), 27-6
DBMS_RESUMABLE package, 19-9
DBMS_SCHEDULER.GET_FILE, retrieving external job stdout with, 29-14
DBMS_SERVER_ALERT package
setting alert thresholds, 19-2
DBMS_SPACE package, 19-28
example for unused space, 19-29
FREE_BLOCK procedure, 19-29
SPACE_USAGE procedure, 19-29
UNUSED_SPACE procedure, 19-29
DBMS_SPACE_ADMIN
DROP_EMPTY_SEGMENTS procedure, 19-28
MATERIALIZE_DEFERRED_SEGMENTS procedure, 20-11
DBMS_STATS package, 18-3
MONITORING clause of CREATE TABLE, 20-26
DBMS_STORAGE_MAP package
invoking for file mapping, 15-20
views detailing mapping information, 15-21
DBMS_TRANSACTION package
PURGE_LOST_DB_ENTRY procedure, 35-10
DBVERIFY utility, 25-3
DDL lock timeout, 2-30
DDL_LOCK_TIMEOUT initialization parameter, 2-30
DEALLOCATE UNUSED clause, 19-28
deallocating unused space, 19-12
DBMS_SPACE package, 19-28
DEALLOCATE UNUSED clause, 19-28
declarative referential integrity constraints, 33-2
dedicated server processes, 5-1
trace files for, 8-1
default temporary tablespace
renaming, 14-24
default temporary tablespaces
specifying at database creation, 2-13, 2-18
specifying bigfile tempfile, 2-21
DEFAULT_CONSUMER_GROUP for Database Resource Manager, 27-16, 27-50
deferred segment creation
in tables, 20-9
indexes, 21-4
deferred segments
materializing, 20-11
defining
chain steps, 29-43
dependencies
between schema objects, 18-17
displaying, 18-26
destinations, Oracle Scheduler
about, 28-6
creating, 29-7
detached jobs, 28-19
DIAGNOSTIC_DEST initialization parameter, 8-2, 9-7
dictionary-managed tablespaces
migrating SYSTEM to locally managed, 14-29
Digital POLYCENTER Manager on NetView, 31-23
direct-path INSERT
benefits, 20-18
how it works, 20-18
index maintenance, 20-21
locking considerations, 20-22
logging mode, 20-20
parallel INSERT, 20-19
parallel load compared with parallel INSERT, 20-18
space considerations, 20-21
disabling
chains, 29-50
jobs, 29-19
programs, 29-24
SQL patch, 9-30
window groups, 29-63
windows, 29-59
disabling recoverer process, 35-18
dispatcher process (Dnnn), 5-20
dispatcher processes, 5-12, 5-15
DISPATCHERS initialization parameter

setting attributes of, 5-10
setting initially, 5-12
distributed applications
distributing data, 33-1
distributed databases
administration overview, 31-17
application development, 31-32, 33-1, 33-9
client/server architectures, 31-4
commit point strength, 34-6
cost-based optimization, 31-34
direct and indirect connections, 31-5
distributed processing, 31-2
distributed queries, 31-24
distributed updates, 31-24
forming global database names, 32-1
global object names, 31-16, 32-1
globalization support, 31-34
location transparency, 31-32, 32-18
management tools, 31-23
managing read consistency, 35-19
nodes of, 31-4
overview, 31-1
remote object security, 32-20
remote queries and updates, 31-24
replicated databases and, 31-3
resumable space allocation, 19-6
running in ARCHIVELOG mode, 13-3
running in NOARCHIVELOG mode, 13-3
scenarios, 32-24
schema object name resolution, 31-28
schema-dependent global users, 31-19
schema-independent global users, 31-20
security, 31-17
site autonomy of, 31-17
SQL transparency, 31-33
starting a remote instance, 3-9
transaction processing, 31-24
transparency, 31-32
distributed processing
distributed databases, 31-2
distributed queries, 31-24
analyzing tables, 33-5
application development issues, 33-2
cost-based optimization, 33-3
optimizing, 31-34
distributed systems
data encryption, 31-22
distributed transactions, 31-25
case study, 34-14
commit point site, 34-5
commit point strength, 34-6, 35-1
committing, 34-6
database server role, 34-4
defined, 34-1
DML and DDL, 34-2
failure during, 35-17
global coordinator, 34-4
local coordinator, 34-4
lock timeout interval, 35-17
locked resources, 35-17
locks for in-doubt, 35-17
manually overriding in-doubt, 35-6
naming, 35-2, 35-7
session trees, 34-3, 34-4, 34-5, 35-4
setting advice, 35-7
transaction control statements, 34-2
transaction timeouts, 35-17
two-phase commit, 34-14, 35-6
viewing database links, 35-2
distributed updates, 31-24
DML
See data manipulation language
DML error logging, inserting data with, 20-22
DRIVING_SITE hint, 33-6
DROP ALL STORAGE clause, 18-7
DROP CLUSTER statement
CASCADE CONSTRAINTS clause, 22-7
dropping cluster, 22-7
dropping cluster index, 22-7
dropping hash cluster, 23-9
INCLUDING TABLES clause, 22-7
DROP DATABASE statement, 2-44
DROP LOGFILE clause
ALTER DATABASE statement, 12-13
DROP LOGFILE MEMBER clause
ALTER DATABASE statement, 12-14
DROP SYNONYM statement, 24-18
DROP TABLE statement
about, 20-50
CASCADE CONSTRAINTS clause, 20-50
for clustered tables, 22-8
DROP TABLESPACE statement, 14-24
dropping
chain steps, 29-50
chains, 29-48
columns
marking unused, 20-31
remove unused columns, 20-31
columns in compressed tables, 20-31
database links, 32-15
datafiles, 15-11
datafiles, Oracle managed, 17-18
job classes, 29-55
jobs, 29-17
programs, 29-23
rules from chains, 29-49
schedules, 29-26
SQL patch, 9-30
tables
CASCADE clause, 20-50
consequences of, 20-50
tempfiles, 15-11
Oracle managed, 17-18
window groups, 29-62
windows, 29-59
DUMP_ORPHAN_KEYS procedure, 25-4
checking sync, 25-4
DBMS_REPAIR package, 25-2
example, 25-9
recovering data, 25-5

dumps, 9-6

E

ECID, 9-4
editions
  in CONNECT statements, 1-9
  managing, 18-21
  Scheduler jobs and, 28-27
e-mail notifications, Scheduler, 29-70
EMCA
  command-line arguments, 10-6
  sample EMCA input file, 10-10
EMPHASIS resource allocation method, 27-28
empty tables
  dropping segments, 19-28
enabling
  chains, 29-47
  jobs, 29-20
  programs, 29-24
  window groups, 29-62
  windows, 29-60
enabling recoverer process
  distributed transactions, 35-18
encryption
  column, 20-8
  tablespace, 14-8
encryption, transparent data, 2-43
enterprise users
  definition, 31-20
environment variables
  ORACLE_SID, 2-7
error logging, DML
  inserting data with, 20-22
errors
  alert log and, 8-1
  assigning names with PRAGMA_EXCEPTION_INIT, 33-8
  critical, 9-1
  exception handler, 33-8
  integrity constrain violation, 33-2
  ORA-00028, 5-24
  ORA-01090, 3-12
  ORA-01173, 11-7
  ORA-01176, 11-7
  ORA-01177, 11-7
  ORA-01215, 11-7
  ORA-01216, 11-7
  ORA-01578, 15-12
  ORA-01591, 35-17
  ORA-02049, 35-17
  ORA-02050, 35-6
  ORA-02051, 35-6
  ORA-02054, 35-6
  RAISE_APPLICATION_ERROR() procedure, 33-8
  remote procedure, 33-8
  rollback required, 33-2
  trace files and, 8-1
  when creating control file, 11-7
  while starting a database, 3-8
  while starting an instance, 3-8
event message
  passing to event-based job, 29-34
event schedule
  altering, 29-34
  creating, 29-33
event-based job
  altering, 29-33
  creating, 29-32
  passing event messages to, 29-34
events
  using to start Scheduler jobs, 29-30
events (Scheduler)
  overview, 29-30
example
  setting maximum utilization limit for plans and subplans, 27-42
examples
  managing parallel statement execution using Resource Manager, 27-45
exception handler, 33-8
EXCEPTION keyword, 33-8
exceptions
  assigning names with PRAGMA_EXCEPTION_INIT, 33-8
  integrity constraints, 18-14
  user-defined, 33-9
executing
  remote external jobs, 30-4
execution context identifier, 9-4
execution plans
  analyzing for distributed queries, 33-6
export operations
  restricted mode and, 3-8
expressions, calendaring, 29-26
EXTENT MANAGEMENT LOCAL clause
  CREATE DATABASE, 2-17
extents
  allocating cluster extents, 22-6
  allocating for tables, 20-29
  data dictionary views for, 19-30
  deallocating cluster extents, 22-6
  displaying free extents, 19-32
external destinations, Oracle Scheduler
  about, 28-6
  creating, 29-7
external jobs
  retrieving stdout and stderr, 28-17, 28-18, 29-14, 29-20
external jobs, Oracle Scheduler, 28-16
external procedures
  managing processes for, 5-22
external tables
  altering, 20-68
  creating, 20-66
  defined, 20-65
  dropping, 20-70
  privileges required, 20-70
  uploading data example, 20-66

F

Fast Recovery Area
  former name, xl
fast recovery area
  as archive log destination, 13-7
  initialization parameters to specify, 2-27
  with Oracle managed files, 17-4
fault diagnosability infrastructure, 9-1
file mapping
  examples, 15-22
  how it works, 15-16
  how to use, 15-19
  overview, 15-16
  structures, 15-18
  views, 15-21
file system
  used for Oracle managed files, 17-2
file watchers
  about, 29-35
  changing detection interval, 29-40
  creating, 29-36
  managing, 29-40
FILE_MAPPING initialization parameter, 15-20
filenames
  Oracle Managed Files, 17-6
files
  creating Oracle Managed Files, 17-5, 17-17
finalizing
  an incident package, definition, 9-34
FIX_CORRUPT_BLOCKS procedure
  DBMS_REPAIR, 25-2
  example, 25-8
  marking blocks corrupt, 25-5
fixed user database links
  advantages and disadvantages, 31-13
  creating, 32-8
  definition, 31-12
  example, 31-14
Flash Cache
  See Database Smart Flash Cache
Flashback Drop
  about, 20-51
  purging recycle bin, 20-53
  querying recycle bin, 20-52
  recycle bin, 20-51
  restoring objects, 20-54
Flashback Table
  overview, 20-49
flood-controlled incidents
  defined, 9-4
  viewing, 9-19
FMON background process, 15-17
FMPUTL external process
  used for file mapping, 15-17
FORCE clause
  COMMIT statement, 35-8
  ROLLBACK statement, 35-8
FORCE LOGGING clause
  CREATE CONTROLFILE, 2-23
  CREATE DATABASE, 2-23
  CREATE TABLESPACE, 14-15
  performance considerations, 2-23
FORCE LOGGING mode, 20-20
forcing
  COMMIT or ROLLBACK, 35-3, 35-6
forcing a log switch, 12-14
  using ARCHIVE_LAG_TARGET, 12-9
  with the ALTER SYSTEM statement, 12-14
forget phase
  in two-phase commit, 34-11
free space
  listing free extents, 19-32
  tablespaces and, 14-49
function-based indexes, 21-11

G

generic connectivity
  definition, 31-4
global coordinators, 34-4
  distributed transactions, 34-4
global database consistency
  distributed databases and, 34-11
global database links, 31-11
  creating, 32-8
global database names
  changing the domain, 32-3
  database links, 31-8
    loopback, 31-9
  enforcing for database links, 31-10
  enforcing global naming, 32-2
  forming distributed database names, 32-1
  impact of changing, 31-31
  querying, 32-3
global object names
  database links, 31-26
  distributed databases, 32-1
global users, 32-26
  schema-dependent in distributed systems, 31-19
  schema-independent in distributed
    systems, 31-20
GLOBAL_NAME view
  using to determine global database name, 32-3
GLOBAL_NAMES initialization parameter
  database links, 31-10
globalization support
  client/server architectures, 31-35
  distributed databases, 31-34
GRANT statement
  SYSOPER/SYSDBA privileges, 1-26
granting privileges and roles
  SYSOPER/SYSDBA privileges, 1-26
groups, Oracle Scheduler, 28-14
growth trends
  of database objects, 19-33
GV$DBLINK view, 32-17

H

hash clusters

advantages and disadvantages, 23-1
altering, 23-9
choosing key, 23-6
contrasted with index clusters, 23-1
controlling space use of, 23-6
creating, 23-2
data dictionary views reference, 23-9
dropping, 23-9
estimating storage, 23-8
examples, 23-8
hash function, 23-1, 23-2, 23-3, 23-6, 23-7
HASH IS clause, 23-3, 23-6
HASHKEYS clause, 23-3, 23-7
single-table, 23-6
SIZE clause, 23-7
sorted, 23-3
hash functions
for hash cluster, 23-1
health checks, 9-2
Health Monitor, 9-21
checks, 9-22
generating reports, 9-24
running, 9-23
viewing reports, 9-24
viewing reports using ADRCI, 9-26
heterogeneous distributed systems
definition, 31-3
Heterogeneous Services
overview, 31-3
HI_SHARED_MEMORY_ADDRESS parameter, 6-19
hints, 33-5
DRIVING_SITE, 33-6
NO_MERGE, 33-6
using to tune distributed queries, 33-5
HP OpenView, 31-23

I

IBM NetView/6000, 31-23
import operations
restricted mode and, 3-8
incident package
correlated, 9-35
correlated, creating, editing, and uploading, 9-45
correlated, deleting, 9-45
creating, editing and uploading custom, 9-32
customizing, 9-39, 9-40
defined, 9-2
viewing, 9-40
incident packaging service, 9-2
incidents
about, 9-3
flood-controlled, 9-4
viewing, 9-19
index clusters
See clusters
indexes
altering, 21-15
analyzing, 18-2
choosing columns to index, 21-3
cluster indexes, 22-5, 22-6, 22-7
coalescing, 21-7, 21-17
column order for performance, 21-3
creating, 21-8
data dictionary views reference, 21-21
deferred segment creation, 21-4
determining unusable status of, 21-17, 21-18
disabling and dropping constraints cost, 21-8
dropping, 21-4, 21-20
estimating size, 21-4
estimating space use, 19-33
explicitly creating a unique index, 21-9
function-based, 21-11
guidelines for managing, 21-1
invisible, 21-6
keeping when disabling constraint, 18-12
keeping when dropping constraint, 18-12
key compression, 21-12
limiting for a table, 21-4
monitoring space use of, 21-20
monitoring usage, 21-19
parallelizing index creation, 21-5
rebuilding, 21-7, 21-16
rebuilt after direct-path INSERT, 20-21
renaming, 21-19
setting storage parameters for, 21-4
shrinking, 19-26
space used by, 21-20
statement for creating, 21-9
tablespace for, 21-5
temporary segments and, 21-2
unusable, 21-6, 21-13, 21-17
validating structure, 18-3
when to create, 21-3
index-organized tables
analyzing, 20-63
AS subquery, 20-59
converting to heap, 20-64
creating, 20-56
described, 20-55
INCLUDING clause, 20-58
key compression, 20-59
maintaining, 20-60
ORDER BY clause, using, 20-64
parallel creation, 20-59
rebuilding with MOVE clause, 20-61
storing nested tables, 20-58
storing object types, 20-57
threshold value, 20-58
in-doubt transactions, 34-11
after a system failure, 35-6
automatic resolution, 34-12
deciding how to handle, 35-5
deciding whether to perform manual override, 35-6
defined, 34-10
manual resolution, 34-13
manually committing, 35-8
manually committing, example, 35-11
manually overriding, 35-6, 35-8

manually overriding, scenario, 35-11
manually rolling back, 35-9
overview, 34-11
pending transactions table, 35-16
purging rows from data dictionary, 35-9, 35-10
recoverer process and, 35-18
rolling back, 35-8, 35-9
SCNs and, 34-14
simulating, 35-17
tracing session tree, 35-4
viewing database links, 35-2
INITIAL parameter
cannot alter, 20-28
initialization parameter file
about, 2-24
creating, 2-8
creating by copying and pasting from alert log, 2-38
creating for database creation, 2-8
editing before database creation, 2-24
individual parameter names, 2-26
sample, 2-26
server parameter file, 2-32
initialization parameter files
default locations, 3-3
search order, 3-3
initialization parameters
about, 2-24
and database startup, 3-3
ARCHIVE_LAG_TARGET, 12-9
BUFFER_POOL_KEEP, 6-16
BUFFER_POOL_RECYCLE, 6-16
changing, 2-35
clearing, 2-37
COMMIT_POINT_STRENGTH, 34-6, 35-1
CONTROL_FILES, 2-28, 11-2, 11-3
DB_BLOCK_CHECKING, 25-4
DB_BLOCK_CHECKSUM, 12-14, 15-12
DB_BLOCK_SIZE, 2-28, 14-14
DB_CACHE_SIZE, 14-14
DB_DOMA, 2-26
DB_DOMAIN, 2-27
DB_FILES, 15-2
DB_NAME, 2-26
DB_nK_CACHE_SIZE, 14-14, 14-43
DISPATCHERS, 5-12
FILE_MAPPING, 15-20
for buffer cache, 6-15
GLOBAL_NAMES, 31-10
HI_SHARED_MEMORY_ADDRESS, 6-19
LOCK_SGA, 6-19
LOG_ARCHIVE_DEST, 13-7
LOG_ARCHIVE_DEST_*n*, 13-7, 13-12
LOG_ARCHIVE_DEST_STATE_*n*, 13-9
LOG_ARCHIVE_MAX_PROCESSES, 13-5
LOG_ARCHIVE_MIN_SUCCEED_DEST, 13-11
LOG_ARCHIVE_TRACE, 13-13
MAX_DUMP_FILE_SIZE, 8-2
OPEN_LINKS, 32-16
PROCESSES, 2-29
REMOTE_LOGIN_PASSWORDFILE, 1-24
REMOTE_OS_AUTHENT, 31-12
resetting, 2-37
RESOURCE_MANAGER_PLAN, 27-35
server parameter file and, 2-32, 2-38
SET SQL_TRACE, 8-3
setting, 2-35
shared server and, 5-6
SHARED_MEMORY_ADDRESS, 6-19
SHARED_SERVERS, 5-8
SORT_AREA_SIZE, 21-2
SPFILE, 2-35, 3-4
SQL_TRACE, 8-1
STATISTICS_LEVEL, 20-26
UNDO_MANAGEMENT, 2-18
UNDO_TABLESPACE, 2-30, 16-2
USE_INDIRECT_DATA_BUFFERS, 6-19
INITRANS parameter
altering, 20-28
INSERT statement
with DML error logging, 20-22
installing
patches, 1-6
instance caging, 27-48
with maximum utilization limit, 27-49
instances
aborting, 3-13
shutting down immediately, 3-12
shutting down normally, 3-12
transactional shutdown, 3-13
instances, managing CPU for multiple, 27-48
integrity constraints
See also constraints
cost of disabling, 21-8
cost of dropping, 21-8
creating indexes associated with, 21-9
dropping tablespaces and, 14-24
ORA-02055 constraint violation, 33-2
INTERNAL username
connecting for shutdown, 3-12
invisible indexes, 21-6
IOT
See index-organized tables
IPS, 9-2

J

Java Message Service (JMS), 10-11
job classes
altering, 29-54
creating, 29-54
dropping, 29-55
managing Scheduler job attributes, resources, and priorities with, 29-53
overview, 28-10
job coordinator, 28-23
job destination ID, defined, 29-16, 29-67
job log, Scheduler
viewing, 29-65
job recovery (Scheduler), 30-16

job scheduling
dependency, 28-2
event-based, 28-2
time-based, 28-1
JOB_QUEUE_PROCESSES initialization parameter, 28-24, A-1
jobs
altering, 29-15
and editions, 28-27
copying, 29-20
creating, 29-2
creating and managing Scheduler, 29-2
creating for chains, 29-48
credentials, 28-7
database, 28-15
detached, 28-19
disabling, 29-19
dropping, 29-17
e-mail notifications, 29-70
enabling, 29-20
event-based, 29-32
external, 28-16
lightweight, 28-20
lightweight, example of creating, 29-4
monitoring, 29-64
monitoring with events raised by the Scheduler, 29-68
multiple-destination, 28-21
status of child jobs, 30-11
overview, 28-5
priorities, 29-55
remote database, 28-16
remote external
about, 28-18
running, 29-15
starting when a file arrives on a system, 29-35
starting with events raised by your application, 29-31
status, 29-64, 30-24
stopping, 29-16
troubleshooting remote, 30-16
viewing information on running, 30-10
viewing stdout and stderr for, 29-20
join views
definition, 24-3
DELETE statements, 24-8
key-preserved tables in, 24-7
modifying, 24-6
rules for modifying, 24-8
updating, 24-6
joins
statement transparency in distributed databases, 32-23

K

key compression, 20-59
indexes, 21-12
key-preserved tables
in join views, 24-7
in outer joins, 24-10
keys
cluster, 22-1, 22-4

L

large objects, 20-13
lightweight jobs, 28-20
example, 29-4
example of creating, 29-4
links
See database links
LIST CHAINED ROWS clause
of ANALYZE statement, 18-4
listeners
removing with srvctl, 4-54
listing database links, 32-16, 35-2, 35-4
loading data
using external tables, 20-66
LOBs, 20-13
local coordinators, 34-4
distributed transactions, 34-4
locally managed tablespaces, 14-3
automatic segment space management in, 14-5
DBMS_SPACE_ADMIN package, 14-26
detecting and repairing defects, 14-26
migrating SYSTEM from dictionary-managed, 14-29
shrinking, temporary, 14-23
tempfiles, 14-12
temporary, creating, 14-12
location transparency in distributed databases
creating using synonyms, 32-20
creating using views, 32-19
restrictions, 32-23
using procedures, 32-22
lock timeout interval
distributed transactions, 35-17
LOCK_SGA parameter, 6-19
locks
in-doubt distributed transactions, 35-17
monitoring, 8-7
log
window (Scheduler), 29-56
log sequence number
control files, 12-3
log switches
description, 12-3
forcing, 12-14
log sequence numbers, 12-3
multiplexed redo log files and, 12-5
privileges, 12-14
using ARCHIVE_LAG_TARGET, 12-9
waiting for archiving to complete, 12-5
log writer process (LGWR), 5-20
multiplexed redo log files and, 12-5
online redo logs available for use, 12-2
trace files and, 12-5
writing to online redo log files, 12-2
LOG_ARCHIVE_DEST initialization parameter

specifying destinations using, 13-7
LOG_ARCHIVE_DEST_*n* initialization parameter, 13-7
REOPEN attribute, 13-12
LOG_ARCHIVE_DEST_STATE_*n* initialization parameter, 13-9
LOG_ARCHIVE_DUPLEX_DEST initialization parameter
specifying destinations using, 13-7
LOG_ARCHIVE_MAX_PROCESSES initialization parameter, 13-5
LOG_ARCHIVE_MIN_SUCCEED_DEST initialization parameter, 13-11
LOG_ARCHIVE_TRACE initialization parameter, 13-13
LOGGING clause
CREATE TABLESPACE, 14-15
logging mode
direct-path INSERT, 20-20
NOARCHIVELOG mode and, 20-20
logical corruptions from DBMS_REPAIR, 25-4
logical volume managers
mapping files to physical devices, 15-15, 15-24
used for Oracle Managed Files, 17-2
LOGON trigger
setting resumable mode, 19-8
logs
job, 30-11
window (Scheduler), 29-56, 30-11
LONG columns, 32-24
LONG RAW columns, 32-24

M

maintenance tasks, automatic
See automatic maintenance tasks
maintenance window
creating, 26-4
definition, 26-1
MAINTENANCE_WINDOW_GROUP, 26-2
modifying, 26-4
predefined, 26-7
removing, 26-5
Scheduler, 26-2
managing
memory, 6-1
space threshold alerts for the undo tablespace, 16-11
managing datafiles, 15-1
managing sequences, 24-12
managing synonyms, 24-17
managing tables, 20-1
managing views, 24-1
manual archiving
in ARCHIVELOG mode, 13-5
manual overrides
in-doubt transactions, 35-8
materializing deferred segments, 20-11
MAX_DUMP_FILE_SIZE initialization parameter, 8-2
MAXDATAFILES parameter
changing, 11-5
MAXINSTANCES, 11-5
MAXLOGFILES parameter
changing, 11-5
CREATE DATABASE statement, 12-8
MAXLOGHISTORY parameter
changing, 11-5
MAXLOGMEMBERS parameter
changing, 11-5
CREATE DATABASE statement, 12-8
MAXTRANS parameter
altering, 20-28
media recovery
effects of archiving on, 13-2
memory
extended buffer cache (32-bit), 6-19
managing, 6-1
overview of architecture of, 6-2
system global area (SGA)
initialization parameters, 6-19
locking into physical memory, 6-19
starting address, 6-19
memory management
about, 6-1
automatic, 6-1, 6-4
data dictionary views reference, 6-23
MEMORY_MAX_TARGET parameter, 6-4
MEMORY_TARGET parameter, 6-4
migrated rows
eliminating from table, procedure, 18-4
MINEXTENTS parameter
cannot alter, 20-28
mirrored files
control files, 2-28, 11-2
online redo log, 12-5
online redo log location, 12-6
online redo log size, 12-7
MISSING datafiles, 11-7
monitoring
performance, 8-6
running chains, 29-52
MONITORING clause
CREATE TABLE, 20-26
MONITORING USAGE clause
of ALTER INDEX statement, 21-19
mounting a database, 3-7
moving control files, 11-4
multiple instances, managing CPU for, 27-48
multiple temporary tablespaces, 14-13, 14-14
multiple-destination jobs, Oracle Scheduler, 28-21
status of child jobs, 30-11
multiplexed control files
importance of, 11-2
multiplexing
archived redo logs, 13-6
control files, 11-2
redo log file groups, 12-4
redo log files, 12-4

N

name resolution in distributed databases
  database links, 31-26
  impact of global name changes, 31-31
  procedures, 31-30
  schema objects, 31-16, 31-28
  synonyms, 31-30
  views, 31-30
  when global database name is complete, 31-27
  when global database name is partial, 31-27
  when no global database name is specified, 31-27
named user limits
  setting initially, 2-31
networks
  connections, minimizing, 32-11
  distributed databases use of, 31-1
NEXT parameter
  altering, 20-28
NEXTVAL pseudo-column, 24-14
  restrictions, 24-15
NO_DATA_FOUND keyword, 33-8
NO_MERGE hint, 33-6
NOARCHIVELOG mode
  archiving, 13-2
  definition, 13-2
  dropping datafiles, 15-7
  LOGGING mode and, 20-20
  media failure, 13-2
  no hot backups, 13-2
  running in, 13-2
  switching to, 13-4
  taking datafiles offline in, 15-7
NOCACHE option
  CREATE SEQUENCE statement, 24-16
NOLOGGING clause
  CREATE TABLESPACE, 14-15
NOLOGGING mode
  direct-path INSERT, 20-20
normal transmission mode
  definition, 13-10
Novell NetWare Management System, 31-23

O

object privileges
  for external tables, 20-70
objects
  See schema objects
offline tablespaces
  priorities, 14-15
  taking offline, 14-15
OLTP table compression, 20-5
online redefinition of tables, 20-33
  abort and cleanup, 20-39
  examples, 20-42
  features of, 20-34
  intermediate synchronization, 20-39
  redefining a single partition, 20-41
    rules for, 20-41
  restrictions, 20-39
  with DBMS_REDEFINITION, 20-34
online redo log files
  See online redo logs
online redo logs
  See also redo log files
  creating groups, 12-10
  creating members, 12-11
  data dictionary views reference, 12-16
  dropping groups, 12-12
  dropping members, 12-12
  forcing a log switch, 12-14
  guidelines for configuring, 12-4
  INVALID members, 12-14
  location of, 12-6
  managing, 12-1
  moving files, 12-11
  number of files in the, 12-8
  optimum configuration for the, 12-8
  renaming files, 12-11
  renaming members, 12-11
  specifying ARCHIVE_LAG_TARGET, 12-9
  STALE members, 12-14
online segment shrink, 19-26
OPEN_LINKS initialization parameter, 32-16
opening windows, 29-58
operating system authentication, 1-20
operating systems
  database administrators requirements for, 1-14
  renaming and relocating files, 15-8
ORA-01013 error message, 3-14
ORA-02055 error
  integrity constraint violation, 33-2
ORA-02067 error
  rollback required, 33-2
ORA-12838 error, direct path insert, 20-20
Oracle Call Interface
  See OCI
Oracle Data Guard
  support by the Scheduler, 28-27, 30-22
Oracle Database
  release numbers, 1-13
Oracle Database File System, xxxix
Oracle Database users
  types of, 1-1
Oracle Enterprise Manager, 3-2
Oracle home
  cloning, 1-6
Oracle Managed Files
  adding to an existing database, 17-23
  behavior, 17-17
  benefits, 17-3
  CREATE DATABASE statement, 17-7
  creating, 17-5
  creating control files, 17-15
  creating datafiles, 17-12
  creating online redo log files, 17-16
  creating tempfiles, 17-14
  described, 17-1
  dropping datafile, 17-18
  dropping online redo log files, 17-18

dropping tempfile, 17-18
initialization parameters, 17-3
introduction, 2-19
renaming, 17-18
Oracle managed files
naming, 17-6
scenarios for using, 17-19
Oracle Managed Files feature
See Oracle managed files
Oracle Restart
about, 4-1
configuration
adding components to, 4-11
modifying, 4-16
removing components from, 4-14
viewing for a component, 4-16
configuring, 4-9
CRSCTL utility, 4-3
disabling and enabling management for a component, 4-14
environment variables in, 4-17
patches
installing, 4-25
registering a component with, 4-12
starting, 4-3
starting and stopping components managed by, 4-25
Oracle home, 4-25
status of components, 4-15
stopping, 4-3
Oracle Scheduler
See Scheduler
Oracle Scheduler Agent, 30-4
Oracle Scheduler agent
on Windows, 30-9
OracleSchedulerExecutionAgent, 30-9
tasks, 30-8
Windows Service, 30-9
Oracle Scheduler agents
registering with databases, 30-9
Oracle Universal Installer, 2-2
Oracle wallet, 14-9, 20-8
ORACLE_SID environment variable, 2-7
OracleSchedulerExecutionAgent, 30-9
oraenv and coraenv, 1-8
ORAPWD utility, 1-23
ORGANIZATION EXTERNAL clause
of CREATE TABLE, 20-66
orphan key table
example of building, 25-7
OSDBA group, 1-20
OSOPER group, 1-20
OTHER_GROUPS
for Database Resource Manager, 27-4
OTHER_GROUPS for Database Resource Manager, 27-32, 27-34, 27-47
outer joins, 24-9
key-preserved tables in, 24-10
overlapping windows, 28-12

P

package
See incident package
packages
DBMS_FILE_TRANSFER, 15-12
DBMS_METADATA, 18-25
DBMS_REDEFINITION, 20-34, 20-48
DBMS_REPAIR, 25-1
DBMS_RESOURCE_MANAGER, 27-3, 27-6, 27-7, 27-12
DBMS_RESOURCE_MANAGER_PRIVS, 27-6, 27-7
DBMS_RESUMABLE, 19-9
DBMS_SPACE, 19-28, 19-29
DBMS_STATS, 18-3, 20-26
DBMS_STORAGE_MAP, 15-21
packaging and uploading problems, 9-35
parallel execution
managing, 5-20
parallel hints, 5-21
parallelizing index creation, 21-5
resumable space allocation, 19-6
parallel hints, 5-21
parallel statement execution
directive attributes for managing, 27-46
managing using Resource Manager, 27-22
PARALLEL_DEGREE_LIMIT_ABSOLUTE resource allocation method, 27-28
parallelizing table creation, 20-4, 20-15
parameter files
See initialization parameter file
partitioned tables
redefining partitions online, 20-41
rules for, 20-41
password
setting for SYSTEM account in CREATE DATABASE statement, 2-16
setting SYS in CREATE DATABASE statement, 2-16
password file
adding users, 1-26
creating, 1-23
ORAPWD utility, 1-23
removing, 1-27
setting REMOTE_LOGIN_PASSWORD, 1-24
synchronizing administrator passwords with the data dictionary, 1-25
viewing members, 1-27
password file authentication, 1-21
passwords
case sensitivity of, 1-18, 1-21, 1-22
default for SYS and SYSTEM, 1-14
password file, 1-26
setting REMOTE_LOGIN_PASSWORD parameter, 1-24
patches
installing, 1-6
Oracle Restart, 4-25
pausing chains and chain steps, 29-51
PCTINCREASE parameter, 20-28

pending area for Database Resource Manager plans, 27-35
validating plan schema changes, 27-33
pending transaction tables, 35-16
performance
index column order, 21-3
location of datafiles and, 15-4
monitoring, 8-6
PGA_AGGREGATE_TARGET initialization parameter, 6-19
plan schemas for Database Resource Manager, 27-18, 27-35, 27-50
validating plan changes, 27-33
plans for Database Resource Manager
examples, 27-37
PL/SQL
replaced views and program units, 24-4
PRAGMA_EXCEPTION_INIT procedure
assigning exception names, 33-8
predefined user accounts, 2-43
prepare phase
abort response, 34-9
in two-phase commit, 34-8
prepared response, 34-9
read-only response, 34-9
recognizing read-only nodes, 34-9
steps, 34-10
prepare/commit phases
effects of failure, 35-17
failures during, 35-6
locked resources, 35-17
pending transaction table, 35-16
prepared response
two-phase commit, 34-9
prerequisites
for creating a database, 2-4
PRIMARY KEY constraints
associated indexes, 21-10
dropping associated indexes, 21-20
enabling on creation, 21-9
foreign key references when dropped, 18-13
indexes associated with, 21-9
priorities
job, 29-55
private database links, 31-10
private synonyms, 24-17
privileges
adding redo log groups, 12-10
altering index, 21-15
altering tables, 20-26
closing a database link, 33-2
creating database links, 32-6
creating tables, 20-12
creating tablespaces, 14-2
database administrator, 1-14
drop table, 20-50
dropping indexes, 21-20
dropping online redo log members, 12-13
dropping redo log groups, 12-13
enabling and disabling triggers, 18-8
for external tables, 20-70
forcing a log switch, 12-14
managing with procedures, 32-23
managing with synonyms, 32-21
managing with views, 32-20
manually archiving, 13-5
renaming objects, 18-16
renaming redo log members, 12-11
RESTRICTED SESSION system privilege, 3-8
Scheduler, 30-23
sequences, 24-13, 24-16
setting chain (Scheduler), 30-2
synonyms, 24-17, 24-18
taking tablespaces offline, 14-15
truncating, 18-6
using a view, 24-4
using sequences, 24-14
views, 24-2, 24-4, 24-12
problem activity log
adding comments to, 9-16
problems
about, 9-3
adding comments to activity log, 9-16
problems (critical errors)
packaging and uploading, 9-35
procedures
external, 5-22
location transparency in distributed databases, 32-21
name resolution in distributed databases, 31-30
remote calls, 31-34
process monitor (PMON), 5-20
processes
See server processes
PROCESSES initialization parameter
setting before database creation, 2-29
PRODUCT_COMPONENT_VERSION view, 1-13
program global area (PGA), 6-2
program global areas, 6-2
programs
altering, 29-23
creating, 29-21
creating and managing, to define Scheduler jobs, 29-21
disabling, 29-24
dropping, 29-23
enabling, 29-24
overview, 28-4
public database links, 31-10
connected user, 32-25
fixed user, 32-25
public fixed user database links, 32-25
public synonyms, 24-17
PURGE_LOST_DB_ENTRY procedure
DBMS_TRANSACTION package, 35-10

Q

queries
distributed, 31-24

distributed application development issues, 33-2
location transparency and, 31-33
remote, 31-24
question mark, 2-14
quiescing a database, 3-14
quotas
tablespace, 14-2

R

RAISE_APPLICATION_ERROR() procedure, 33-8
read consistency
managing in distributed databases, 35-19
read-only database
opening, 3-10
read-only databases
limitations, 3-10
read-only response
two-phase commit, 34-9
read-only tables, 20-32
read-only tablespaces
datafile headers when rename, 14-23
delaying opening of datafiles, 14-20
making read-only, 14-18
making writable, 14-19
WORM devices, 14-20
Real Application Clusters
allocating extents for cluster, 22-6
sequence numbers and, 24-13
threads of online redo log, 12-1
rebuilding indexes, 21-16
costs, 21-7
online, 21-16
reclaiming unused space, 19-12
RECOVER clause
STARTUP command, 3-9
recoverer process
disabling, 35-18
distributed transaction recovery, 35-18
enabling, 35-18
pending transaction table, 35-18
recoverer process (RECO), 5-20
recovering
Scheduler jobs, 30-16
recovery
creating new control files, 11-5
Recovery Manager
starting a database, 3-2
starting an instance, 3-2
recycle bin
about, 20-51
purging, 20-53
renamed objects, 20-51
restoring objects from, 20-54
viewing, 20-52
redefining tables online
See online redefinition of tables
redo log files
See also online redo logs
active (current), 12-3
archiving, 13-2
available for use, 12-2
block size, setting, 12-7
circular use of, 12-2
clearing, 12-5, 12-15
contents of, 12-2
creating as Oracle Managed Files, 17-16
creating as Oracle Managed Files, example, 17-21
creating groups, 12-10
creating members, 12-10, 12-11
distributed transaction information in, 12-2
dropping groups, 12-12
dropping members, 12-12
group members, 12-4
groups, defined, 12-4
how many in redo log, 12-8
inactive, 12-3
instance recovery use of, 12-1
legal and illegal configurations, 12-5
LGWR and the, 12-2
log switches, 12-3
maximum number of members, 12-8
members, 12-4
mirrored, log switches and, 12-5
multiplexed, 12-4, 12-5
online, defined, 12-1
planning the, 12-4, 12-9
redo entries, 12-2
requirements, 12-5
specifying at database creation, 17-8
storing separately from datafiles, 15-4
threads, 12-1
unavailable when database is opened, 3-6
verifying blocks, 12-14
redo logs
See online redo log
See redo log files
redo records, 12-2
LOGGING and NOLOGGING, 14-15
referential integrity
distributed database application development, 33-2
release number format, 1-13
releases, 1-13
checking the Oracle Database release number, 1-13
relocating control files, 11-4
remote connections
connecting as SYSOPER/SYSDBA, 1-16
password files, 1-24
remote data
querying, 32-23
updating, 32-23
remote database jobs, 28-16
Scheduler agent setup, 30-6
remote external jobs
about, 28-18
executing, 30-4
Scheduler agent setup, 30-6
Remote Method Invocation (RMI), 10-11

remote procedure calls, 31-34
distributed databases and, 31-33
remote queries
distributed databases and, 31-24
remote transactions, 31-25
defined, 31-25
REMOTE_LOGIN_PASSWORDFILE initialization parameter, 1-24
REMOTE_OS_AUTHENT initialization parameter
connected user database links, 31-12
RENAME statement, 18-16
renaming control files, 11-4
renaming files
Oracle Managed Files, 17-18
renaming indexes, 21-19
REOPEN attribute
LOG_ARCHIVE_DEST_*n* initialization parameter, 13-12
repair table
example of building, 25-6
repairing data block corruption
DBMS_REPAIR, 25-1
repeat interval, schedule, 29-26
RESIZE clause
for single-file tablespace, 14-21
resource allocation methods
active session pool, 27-28
ACTIVE_SESS_POOL_MTH, 27-28
CPU, 27-18
CPU resource, 27-28
EMPHASIS, 27-28
limit on degree of parallelism, 27-28
MAX_UTILIZATION_METHOD, 27-19
PARALLEL_DEGREE_LIMIT_ABSOLUTE, 27-28
PARALLEL_DEGREE_LIMIT_MTH, 27-28
PARALLEL_DEGREE_LIMIT_P1, 27-20
PARALLEL_QUEUE_TIMEOUT, 27-22
PARALLEL_TARGET_PERCENTAGE, 27-21
QUEUEING_MTH, 27-28
queuing resource allocation method, 27-28
ROUND-ROBIN, 27-27
resource consumer groups, 27-3
changing, 27-12
creating, 27-27
DEFAULT_CONSUMER_GROUP, 27-16, 27-50
deleting, 27-50
granting the switch privilege, 27-16
managing, 27-7, 27-13
OTHER_GROUPS, 27-4, 27-32, 27-34, 27-47
parameters, 27-27
revoking the switch privilege, 27-17
setting initial, 27-8
switching a session, 27-12
switching sessions for a user, 27-13
SYS_GROUP, 27-47
updating, 27-50
Resource Manager
AUTO_TASK_CONSUMER_GROUP consumer group, 26-5
managing parallel statement execution, 27-22
resource plan directives, 27-3, 27-34
deleting, 27-51
for managing parallel statement execution, 27-46
specifying, 27-29
updating, 27-51
resource plans, 27-3, 27-5
creating, 27-24
DEFAULT_MAINTENANCE_PLAN, 26-6
DELETE_PLAN_CASCADE, 27-50
deleting, 27-50
examples, 27-37
parameters, 27-28
plan schemas, 27-18, 27-35, 27-50
SYSTEM_PLAN, 27-47
top plan, 27-34, 27-35
updating, 27-50
validating, 27-33
RESOURCE_MANAGER_PLAN initialization parameter, 27-35
RESTRICTED SESSION system privilege
restricted mode and, 3-8
result cache
and the shared pool size, 6-17
setting size of, 6-18
RESULT_CACHE_SIZE initialization parameter, 6-18
resumable space allocation
correctable errors, 19-6
detecting suspended statements, 19-8
disabling, 19-7
distributed databases, 19-6
enabling, 19-7
example, 19-10
how resumable statements work, 19-4
naming statements, 19-8
parallel execution and, 19-6
resumable operations, 19-5
setting as default for session, 19-8
timeout interval, 19-8, 19-9
RESUMABLE_TIMEOUT initialization parameter, 19-4
setting, 19-7
retention guarantee (for undo), 16-4
reversing table changes, 20-48
RMAN
See Recovery Manager
roles
DBA role, 1-15
obtained through database links, 31-16
ROLLBACK statement
FORCE clause, 35-8, 35-9
forcing, 35-6
rollbacks
ORA-02, 33-2
ROUND-ROBIN resource allocation method, 27-27
rows
listing chained or migrated, 18-4
rules
adding to a chain, 29-44
dropping from chains, 29-49

running
chains, 29-49
jobs, 29-15
SQL Repair Advisor, 9-29

S

Sample Schemas
description, 2-44
savepoints
in-doubt transactions, 35-8, 35-9
schagent utility, 30-8
Scheduler
administering, 30-1
architecture, 28-23
configuring, 30-1
credentials for jobs, 28-7
data dictionary views reference, 30-24
e-mail notifications, 29-70
examples of using, 30-17
import and export, 30-14
maintenance window, 26-2
monitoring and managing, 30-10
monitoring jobs, 29-64
objects, 28-3
overview, 28-1
scheduling tasks with, 29-1
security, 30-14
support for Oracle Data Guard, 28-27, 30-22
troubleshooting, 30-14
job does not run, 30-15
using in RAC, 28-25
Scheduler agent, 30-4
configuration, 30-6
installation, 30-6
setup, 30-6
Scheduler calendaring syntax, 29-26
Scheduler chain condition syntax, 29-44
Scheduler objects, naming, 29-1
Scheduler privileges reference, 30-23
SCHEDULER_BATCH_ERRORS view, 29-19
schedules
altering, 29-25
creating, 29-25
creating and managing, to define Scheduler jobs, 29-24
dropping, 29-26
overview, 28-4
scheduling database tasks, 29-1
schema objects
analyzing, 18-2
creating multiple objects, 18-1
data dictionary views reference, 18-25
defining using DBMS_METADATA package, 18-25
dependencies between, 18-17
distributed database naming conventions for, 31-16
global names, 31-16
listing by type, 18-26
name resolution in distributed databases, 31-16, 31-28
name resolution in SQL statements, 18-19
privileges to rename, 18-16
referencing with synonyms, 32-20
renaming, 18-16, 18-17
validating structure, 18-3
viewing information, 18-24, 19-29
schema objects space usage
data dictionary views reference, 19-30
SCN
See system change number
SCOPE clause
ALTER SYSTEM SET, 2-35
scripts, authenticating users in, 2-44
SEC_CASE_SENSITIVE_LOGON initialization parameter, 1-18, 1-21
security
accessing a database, 7-1
administrator of, 7-1
centralized user management in distributed databases, 31-19
database security, 7-1
distributed databases, 31-17
establishing policies, 7-1
privileges, 7-1
remote objects, 32-20
Scheduler, 30-14
using synonyms, 32-21
Segment Advisor, 19-12
configuring Scheduler job, 19-24
invoking with Enterprise Manager, 19-14
invoking with PL/SQL, 19-16
running manually, 19-14
viewing results, 19-18
views, 19-25
SEGMENT_FIX_STATUS procedure
DBMS_REPAIR, 25-2
segments
available space, 19-29
data dictionary views for, 19-30
deallocating unused space, 19-12
displaying information on, 19-30
dropping for empty tables, 19-28
shrinking, 19-26
SELECT statement
FOR UPDATE clause and location transparency, 32-23
SEQUENCE_CACHE_ENTRIES parameter, 24-16
sequences
accessing, 24-14
altering, 24-13
caching sequence numbers, 24-16
creating, 24-13, 24-16
CURRVAL, 24-15
data dictionary views reference, 24-19
dropping, 24-16
managing, 24-12
NEXTVAL, 24-14
Oracle Real Applications Clusters and, 24-13

SERVER parameter
net service name, 32-13
server parameter file
creating, 2-33
defined, 2-32
exporting, 2-37
migrating to, 2-33
recovering, 2-38
RMAN backup, 2-37
setting initialization parameter values, 2-35
SPFILE initialization parameter, 2-35
STARTUP command behavior, 2-32
viewing parameter settings, 2-38
server processes
archiver (ARCn), 5-20
background, 5-19
checkpoint (CKPT), 5-20
database writer (DBWn), 5-20
dedicated, 5-1
dispatcher (Dnnn), 5-20
dispatchers, 5-12
log writer (LGWR), 5-20
monitoring locks, 8-7
process monitor (PMON), 5-20
recoverer (RECO), 5-20
shared server, 5-2
system monitor (SMON), 5-20
trace files for, 8-1
server-generated alerts, 8-4
servers
role in two-phase commit, 34-4
service names
database links and, 32-10
services
controlling automatic startup of, 3-5
creating with SRVCTL and Oracle Restart, 4-19
role-based, 3-5
session trees for distributed transactions
clients, 34-4
commit point site, 34-5, 34-6
database servers, 34-4
definition, 34-3
global coordinators, 34-4
local coordinators, 34-4
tracing transactions, 35-4
sessions
active, 5-24
inactive, 5-25
setting advice for transactions, 35-7
terminating, 5-23
SET TIME_ZONE clause
ALTER SESSION, 2-22
CREATE DATABASE, 2-22
SET TRANSACTION statement
naming transactions, 35-2
SGA
See system global area
SGA_MAX_SIZE initialization parameter, 6-9
SGA_TARGET initialization parameter, 6-9
shared database links
configuring, 32-12
creating, 32-12
dedicated servers, creating links to, 32-12
determining whether to use, 32-11
example, 31-14
shared servers, creating links to, 32-13
SHARED keyword
CREATE DATABASE LINK statement, 32-12
shared server, 5-2
configuring dispatchers, 5-10
data dictionary views reference, 5-15
disabling, 5-9, 5-14
initialization parameters, 5-6
interpreting trace output, 8-3
setting minimum number of servers, 5-8
trace files for processes, 8-1
shared SQL
for remote and distributed statements, 31-25
SHARED_MEMORY_ADDRESS parameter, 6-19
shrinking segments online, 19-26
shutdown
default mode, 3-12
SHUTDOWN command
IMMEDIATE clause, 3-13
interrupting, 3-14
NORMAL clause, 3-12
silent mode
creating a database with DBCA in, 2-5
Simple Network Management Protocol (SNMP) support
database management, 31-23
single-file tablespaces
description, 14-6
single-instance
defined, 2-6
single-table hash clusters, 23-6
site autonomy
distributed databases, 31-17
SKIP_CORRUPT_BLOCKS procedure, 25-5
DBMS_REPAIR, 25-2
example, 25-9
skipping chain steps, 29-52
snapshot too old error, 16-4
SORT_AREA_SIZE initialization parameter
index creation and, 21-2
space
deallocating unused, 19-28
reclaiming unused, 19-12
space allocation
resumable, 19-4
space management
data types, space requirements, 19-29
deallocating unused space, 19-12
Segment Advisor, 19-12
shrink segment, 19-12
SPACE_ERROR_INFO procedure, 19-9
SPFILE initialization parameter, 2-35
specifying from client system, 3-4
SQL
submitting, 1-6

SQL failure
repairing with SQL Repair Advisor, 9-28
SQL patch
disabling, 9-30
removing, 9-30
viewing, 9-30
SQL Repair Advisor
about, 9-29
repairing SQL failure with, 9-28
running, 9-29
SQL statements
distributed databases and, 31-24
SQL test case builder, 9-3
SQL*Loader
about, 1-28
SQL*Plus, 1-6
about, 1-7
connecting with, 1-7
starting, 3-5
starting a database, 3-2
starting an instance, 3-2
SQL_TRACE initialization parameter
trace files and, 8-1
SRVCTL
add asm command, 4-32
add command, usage description, 4-32
add database command, 4-33
add listener command, 4-35
add ons command, 4-35
adding a disk group with, 4-32
case sensitivity, 4-30, 4-71
case sensitivity of commands, 4-30, 4-71
command reference, 4-30
commands, case sensitivity, 4-30, 4-71
component names, 4-31
config asm command, 4-38
config command, usage description, 4-38
config database command, 4-38
config listener command, 4-39
config ons command, 4-40
config service command, 4-40
creating and deleting databases services with, 4-19
disable asm command, 4-41
disable command, usage description, 4-41
disable database command, 4-41
disable diskgroup command, 4-42
disable listener command, 4-42
disable ons command, 4-42
disable service command, 4-43
enable asm command, 4-44
enable command, usage description, 4-44
enable database command, 4-44
enable diskgroup command, 4-45
enable listener command, 4-45
enable ons command, 4-45
enable service command, 4-46
getenv asm command, 4-47
getenv command, usage description, 4-47
getenv database command, 4-47
getenv listener command, 4-48
help for, 4-11
modify asm command, 4-49
modify command, usage description, 4-49
modify database command, 4-50
modify listener command, 4-50
modify ons command, 4-51
modify service command, 4-51
preparing to run, 4-10
reference, 4-30
remove asm command, 4-53
remove command, usage description, 4-53
remove database command, 4-54
remove diskgroup command, 4-54
remove listener command, 4-54
remove ons command, 4-55
remove service command, 4-55
setenv asm command, 4-56
setenv command, usage description, 4-56
setenv database command, 4-57
setenv listener command, 4-57
start asm command, 4-58
start command, usage description, 4-58
start database command, 4-59
start diskgroup command, 4-59
start home command, 4-59
start listener command, 4-60
start ons command, 4-60
start service command, 4-60
status asm command, 4-62
status command, usage description, 4-62
status database command, 4-62
status diskgroup command, 4-63
status home command, 4-63
status listener command, 4-64
status ons command, 4-64
status service command, 4-64
stop asm command, 4-65
stop command, usage description, 4-65
stop database command, 4-66
stop diskgroup command, 4-66
stop home command, 4-66
stop listener command, 4-67
stop ons command, 4-67
stop service command, 4-68
unsetenv asm command, 4-69
unsetenv command, usage description, 4-69
unsetenv database command, 4-70
unsetenv listener command, 4-70
SRVCTL stop option
default, 3-12
STALE status
of redo log members, 12-14
stalled chain (Scheduler), 29-53
standby transmission mode
definition of, 13-10
starting a database, 3-1
forcing, 3-8
Oracle Enterprise Manager, 3-2
recovery and, 3-9

Recovery Manager, 3-2
restricted mode, 3-7
SQL*Plus, 3-2
when control files unavailable, 3-6
when redo logs unavailable, 3-6
starting an instance
automatically at system startup, 3-9
database closed and mounted, 3-7
database name conflicts and, 2-27
forcing, 3-8
mounting and opening the database, 3-7
normally, 3-7
Oracle Enterprise Manager, 3-2
recovery and, 3-9
Recovery Manager, 3-2
remote instance startup, 3-9
restricted mode, 3-7
SQL*Plus, 3-2
when control files unavailable, 3-6
when redo logs unavailable, 3-6
without mounting a database, 3-7
startup
allocation of the SGA
starting a, 6-19
of database services, controlling, 3-5
STARTUP command
default behavior, 2-32
NOMOUNT clause, 2-10
RECOVER clause, 3-9
starting a database, 3-2, 3-6
statement transparency in distributed database
managing, 32-23
statistics
automatically collecting for tables, 20-26
STATISTICS_LEVEL initialization parameter
automatic statistics collection, 20-26
stderr
for local external jobs, 28-17, 28-18, 29-20
retrieving, 28-17, 28-18, 29-20
stdout
for local external jobs, 28-17, 28-18, 29-20
retrieving, 28-17, 28-18, 29-14, 29-20
steps, chain
dropping, 29-50
stopping
chain steps, 29-50
chains, 29-50
jobs, 29-16
storage parameters
INITIAL, 20-28
INITRANS, altering, 20-28
MAXTRANS, altering, 20-28
MINEXTENTS, 20-28
NEXT, 20-28
PCTINCREASE, 20-28
storage subsystems
mapping files to physical devices, 15-15, 15-24
stored procedures
managing privileges, 32-23
remote object security, 32-23
submitting SQL and commands to the database, 1-6
subqueries
in remote updates, 31-24
statement transparency in distributed
databases, 32-23
SunSoft SunNet Manager, 31-23
Support Workbench, 9-6
for Oracle ASM instance, 9-18
viewing problems with, 9-18
SWITCH LOGFILE clause
ALTER SYSTEM statement, 12-14
synonyms, 24-18
creating, 24-17, 32-20
data dictionary views reference, 24-19
definition and creation, 32-20
displaying dependencies of, 18-26
dropping, 24-18
examples, 32-21
location transparency in distributed
databases, 32-20
managing, 24-17, 24-18
managing privileges in remote database, 32-21
name resolution in distributed databases, 31-30
private, 24-17
public, 24-17
remote object security, 32-21
SYS account
default password, 1-14
objects owned, 1-15
privileges, 1-15
specifying password for CREATE DATABASE
statement, 2-16
SYS_GROUP for Database Resource Manager, 27-47
SYSAUX tablespace, 14-2
about, 2-17
cannot rename, 14-23
creating at database creation, 2-12, 2-17
DATAFILE clause, 2-17
monitoring occupants, 14-25
moving occupants, 14-25
SYSDBA system privilege
adding users to the password file, 1-26
connecting to database, 1-16
determining who has privileges, 1-27
granting and revoking, 1-26
SYSOPER system privilege
adding users to the password file, 1-26
connecting to database, 1-16
determining who has privileges, 1-27
granting and revoking, 1-26
SYSTEM account
default password, 1-14
objects owned, 1-15
specifying password for CREATE
DATABASE, 2-16
system change numbers
coordination in a distributed database
system, 34-11
in-doubt transactions, 35-8
using V$DATAFILE to view information

about, 15-25
when assigned, 12-2
system global area, 6-2
holds sequence number cache
specifying buffer cache sizes, 6-15
system monitor process (SMON), 5-20
system privileges
ADMINISTER_RESOURCE_MANAGER, 27-6
for external tables, 20-70
SYSTEM tablespace
cannot rename, 14-23
creating at database creation, 2-12
creating locally managed, 2-12, 2-17
restrictions on taking offline, 15-6
when created, 14-2
SYSTEM_PLAN for Database Resource Manager, 27-47

T

table size
estimating, 19-32
tables
about, 20-1
adding columns, 20-29
allocating extents, 20-29
altering, 20-27
altering physical attributes, 20-28
analyzing, 18-2
clustered (hash)
See hash clusters
compressed, 20-5
creating, 20-12
data dictionary views reference, 20-71
deferred segment creation, 20-9
designing before creating, 20-2
dropping, 20-50
dropping columns, 20-30
estimating size, 20-12
estimating space use, 19-32
external, 20-64
Flashback Drop, 20-50
Flashback Table, 20-49
guidelines for managing, 20-2
hash clustered
See hash clusters
increasing column length, 20-29
index-organized, 20-55
key-preserved, 24-7
limiting indexes on, 21-4
managing, 20-1
modifying column definition, 20-29
moving, 20-28
parallelizing creation, 20-4, 20-15
read-only, 20-32
redefining online, 20-33
renaming columns, 20-30
researching and reversing erroneous changes to, 20-48
restrictions when creating, 20-12
setting storage parameters, 20-12
shrinking, 19-26
specifying location, 20-4
statistics collection, automatic, 20-26
temporary, 20-14
truncating, 18-5
unrecoverable (NOLOGGING), 20-4
validating structure, 18-3
tablespace set, 14-36
tablespaces
adding datafiles, 15-4
assigning user quotas, 14-2
autoextending, 14-21
automatic segment space management, 14-5
bigfile, 2-20, 14-6
checking default storage parameters, 14-48
compressed, 14-8
containing XMLTypes, 14-32
creating undo tablespace at database creation, 2-18, 2-21
data dictionary views reference, 14-47
DBMS_SPACE_ADMIN package, 14-26
default temporary tablespace, creating, 2-18, 2-21
detecting and repairing defects, 14-26
diagnosing and repairing problems in locally managed, 14-26
dictionary managed, 14-8
dropping, 14-24
encrypted, 14-8
guidelines for managing, 14-1
increasing size, 14-21
listing files of, 14-48
listing free space in, 14-49
locally managed, 14-3
locally managed SYSTEM, 2-17
locally managed temporary, 14-12
location, 15-4
migrating SYSTEM to locally managed, 14-29
multiple block sizes, 14-43
on a WORM device, 14-20
Oracle Managed Files, managing, 17-22, 17-23
overriding default type, 2-21
quotas, assigning, 14-2
read-only, 14-17
renaming, 14-21, 14-23
setting default type, 2-21
single-file, 2-20, 2-21, 14-6, 14-22
specifying nonstandard block sizes, 14-14
SYSAUX, 14-2, 14-23
SYSAUX creation, 2-17
SYSAUX, managing, 14-25
SYSTEM, 14-2, 14-4, 14-18, 14-29
taking offline normal, 14-15
taking offline temporarily, 14-16
tempfiles in locally managed, 14-12
temporary, 14-10, 14-14
temporary bigfile, 14-12
temporary for creating large indexes, 21-11
transportable
See transportable tablespaces

undo, 16-1
using multiple, 14-2
using Oracle Managed Files, 17-12
tempfiles, 14-12
creating as Oracle managed, 17-14
dropping, 15-11
dropping Oracle managed, 17-18
temporary segments
index creation and, 21-2
temporary tables
assigning to a tablespace, 20-15
creating, 20-14
temporary tablespace, default
specifying at database creation, 17-10
temporary tablespaces
altering, 14-22
bigfile, 14-12
creating, 14-12
groups, 14-13
renaming default, 14-24
shrinking, locally managed, 14-23
terminating user sessions
active sessions, 5-24
identifying sessions, 5-24
inactive session, example, 5-25
inactive sessions, 5-25
test case
builder, SQL, 9-3
threads
online redo log, 12-1
threshold based alerts
managing with Oracle Enterprise Manager, 8-4
threshold-based alerts
server-generated, 8-4
thresholds
setting alert, 19-2
time zone
files, 2-22
setting for database, 2-22
trace files, 9-6
location of, 8-2
log writer process and, 12-5
size of, 8-2
using, 8-1
when written, 8-3
trace files, finding, 9-21
traces, 9-6
tracing
archivelog process, 13-13
transaction control statements
distributed transactions and, 34-2
transaction failures
simulating, 35-17
transaction management
overview, 34-7
transaction processing
distributed systems, 31-24
transactions
closing database links, 33-2
distributed and two-phase commit, 31-26
in-doubt, 34-10, 34-11, 34-14, 35-5
naming distributed, 35-2, 35-7
remote, 31-25
transmitting archived redo logs, 13-10
transparent data encryption, 2-43, 14-9, 20-8
transportable set
See transportable tablespace set
transportable tablespace set
defined, 14-35
transportable tablespaces, 14-30
compatibility considerations, 14-34
from backup, 14-30
introduction, 14-30
limitations, 14-32
multiple block sizes, 14-43
procedure, 14-35
when to use, 14-44
wizard in Enterprise Manager, 14-31
XMLTypes in, 14-32
transporting tablespaces between databases
See transportable tablespaces
triggers
disabling, 18-9
enabling, 18-9
.trm files, 9-6
TRUNCATE statement
DROP ALL STORAGE, 18-6
DROP STORAGE, 18-6
DROP STORAGE clause, 18-7
REUSE STORAGE, 18-6
REUSE STORAGE clause, 18-7
vs. dropping table, 20-50
tuning
analyzing tables, 33-5
cost-based optimization, 33-3
two-phase commit
case study, 34-14
commit phase, 34-10, 34-17
described, 31-26
discovering problems with, 35-6
distributed transactions, 34-7
example, 34-14
forget phase, 34-11
in-doubt transactions, 34-11, 34-14
phases, 34-7
prepare phase, 34-8, 34-10
recognizing read-only nodes, 34-9
specifying commit point strength, 35-1
steps in commit phase, 34-10
tracing session tree in distributed transactions, 35-4
viewing database links, 35-2

U

Undo Advisor, 16-6
undo management
automatic, 16-2
described, 16-1
initialization parameters for, 16-2

undo retention
automatic tuning of, 16-4
explained, 16-3
guaranteeing, 16-4
setting, 16-5
undo segments
in-doubt distributed transactions, 35-6
undo space
data dictionary views reference, 16-11
undo space management
automatic undo management mode, 16-2
Undo tablespace
specifying at database creation, 17-9
undo tablespace
managing, 16-7
managing space threshold alerts, 16-11
sizing a fixed-size, 16-6
undo tablespaces
altering, 16-9
creating, 16-8
data dictionary views reference, 16-11
dropping, 16-9
monitoring, 16-12
PENDING OFFLINE status, 16-10
renaming, 14-24
specifying at database creation, 2-13, 2-18, 2-21
statistics for, 16-11
switching, 16-10
user quotas, 16-10
UNDO_MANAGEMENT initialization parameter, 2-18
UNDO_TABLESPACE initialization parameter
for undo tablespaces, 2-30
starting an instance using, 16-2
UNIQUE key constraints
associated indexes, 21-10
dropping associated indexes, 21-20
enabling on creation, 21-9
foreign key references when dropped, 18-13
indexes associated with, 21-9
UNRECOVERABLE DATAFILE clause
ALTER DATABASE statement, 12-15
unusable indexes, 21-6
updates
location transparency and, 31-33
upgrading a database, 2-2
USE_INDIRECT_DATA_BUFFERS parameter, 6-19
user accounts
predefined, 2-43, 7-2
user names
SYS and SYSTEM, 1-14
USER_DB_LINKS view, 32-16
USER_DUMP_DEST initialization parameter, 9-5
USER_RESUMABLE view, 19-9
users
assigning tablespace quotas, 14-2
in a newly created database, 2-43
limiting number of, 2-31
predefined, 2-43
session, terminating, 5-25
utilities
for the database administrator, 1-28
SQL*Loader, 1-28
UTLCHAIN.SQL script
listing chained rows, 18-4
UTLCHN1.SQL script
listing chained rows, 18-4
UTLLOCKT.SQL script, 8-7

V

V$ARCHIVE view, 13-14
V$ARCHIVE_DEST view
obtaining destination status, 13-9
V$BLOCKING_QUIESCE view, 3-15
V$BUFFER_POOL view, 6-16
V$DATABASE view, 13-15
V$DBLINK view, 32-17
V$DIAG_CRITICAL_ERROR view, 9-10
V$DIAG_INFO view, 9-9
V$DISPATCHER view
monitoring shared server dispatchers, 5-12
V$DISPATCHER_RATE view
monitoring shared server dispatchers, 5-13
V$ENCRYPTED_TABLESPACES view, 14-10, 14-47
V$INSTANCE view
for database quiesce state, 3-16
V$LOG view, 12-16, 13-14
displaying archiving status, 13-14
V$LOG_HISTORY view, 12-16
V$LOGFILE view, 12-16
log file status, 12-14
V$OBJECT_USAGE view
for monitoring index usage, 21-19
V$PWFILE_USERS view, 1-27
V$QUEUE view
monitoring shared server dispatchers, 5-12
V$RESULT_CACHE_STATISTICS view, 6-18, 6-19
V$ROLLSTAT view
undo segments, 16-11
V$SESSION view, 5-25
V$SYSAUX_OCCUPANTS view
occupants of SYSAUX tablespace, 14-25
V$THREAD view, 12-16
V$TIMEZONE_NAMES view
time zone table information, 2-22
V$TRANSACTION view
undo tablespaces information, 16-11
V$UNDOSTAT view
statistics for undo tablespaces, 16-11
V$VERSION view, 1-13
VALIDATE STRUCTURE clause
of ANALYZE statement, 18-3
VALIDATE STRUCTURE ONLINE clause
of ANALYZE statement, 18-4
verifying blocks
redo log files, 12-14
viewing
alerts, 19-3
incident package details, 9-40

SQL patch, 9-30
views
creating, 24-2
creating with errors, 24-3
data dictionary
for archived redo logs, 13-14
for clusters, 22-8
for control files, 11-9
for database, 2-45
for database resident connection pooling, 5-19
for Database Resource Manager, 27-60
for datafiles, 15-24
for hash clusters, 23-9
for indexes, 21-21
for memory management, 6-23
for Oracle Scheduler, 30-24
for redo log, 12-16
for schema objects, 18-25
for sequences, 24-19
for shared server, 5-15
for space usage in schema objects, 19-30
for synonyms, 24-19
for tables, 20-71
for tablespaces, 14-47
for undo space, 16-11
for views, 24-19
data dictionary views for, 24-19
DBA_2PC_NEIGHBORS, 35-4
DBA_2PC_PENDING, 35-2
DBA_DB_LINKS, 32-16
DBA_RESUMABLE, 19-9
displaying dependencies of, 18-26
dropping, 24-12
file mapping views, 15-21
FOR UPDATE clause and, 24-2
invalid, 24-5
join
See join views
location transparency in distributed databases, 32-19
managing, 24-1, 24-4
managing privileges with, 32-20
name resolution in distributed databases, 31-30
ORDER BY clause and, 24-2
remote object security, 32-20
restrictions, 24-5
USER_RESUMABLE, 19-9
using, 24-4
V$ARCHIVE, 13-14
V$ARCHIVE_DEST, 13-9
V$DATABASE, 13-15
V$LOG, 13-14
V$LOGFILE, 12-14
V$OBJECT_USAGE, 21-19
wildcards in, 24-3
WITH CHECK OPTION, 24-2
virtual columns, 20-2
indexing, 21-3

W

wallet, Oracle, 14-9, 20-8
wildcards
in views, 24-3
window groups
creating, 29-61
disabling, 29-63
dropping, 29-62
dropping a member from, 29-62
enabling, 29-62
managing job scheduling and job priorities with, 29-60
overview, 28-14
window logs, 29-56
windows (Scheduler)
altering, 29-57
closing, 29-58
creating, 29-57
disabling, 29-59
dropping, 29-59
enabling, 29-60
opening, 29-58
overlapping, 28-12
overview, 28-11
windows, managing job scheduling and resource allocation with, 29-55
workloads
managing with database services, 2-39
WORM devices
and read-only tablespaces, 14-20
WRH$_UNDOSTAT view, 16-11

X

XMLTypes
in transportable tablespaces, 14-32