

Oracle8i

Backup and Recovery Guide

Release 8.1.5

February 1999

Part No. A67773-01

ORACLE[®]

Oracle8i Backup and Recovery Guide, Release 8.1.5

Part No. A67773-01

Copyright © 1999, Oracle Corporation. All rights reserved.

Primary Authors: Connie Dialeris, Joyce Fee, Lance Ashdown

Contributing Authors: Greg Pongracz, Francisco Sanchez, Steve Wertheimer

Contributors: Richard Anderson, Don Beusee, Bill Bridge, Sandra Cheevers, Mike Cusson, Rick Frank, John Frazzini, Tim Berry-Hart, Wei Hu, Mike Johnson, Joy Kundu, Gordon Larimer, Bill Lee, Juan Loaiza, Diana Lorentz, Eric Magrath, Alok Pareek, Lyn Pratt, Thomas Pystynen, Daniel Semler, Slartibartfast, Bob Thome, Ron Weiss.

Graphic Designer: Valarie Moore

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

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

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

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle is a registered trademark, and LogMiner, Net8, Oracle7, Oracle8, Oracle8i, PL/SQL, SQL*Net, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

Send Us Your Comments	xvii
Preface.....	xix
Part I Developing a Backup and Recovery Strategy	
1 What Is Backup and Recovery?	
What Is Backup and Recovery?	1-2
How Oracle Keeps Records of Database Transactions	1-4
Backup and Recovery Operations.....	1-4
Elementary Backup and Recovery Strategy	1-5
Which Data Structures Are Important for Backup and Recovery?	1-6
Datafiles	1-6
Control Files	1-7
Rollback Segments.....	1-8
Online Redo Log Files.....	1-8
Archived Redo Log Files	1-10
Understanding Basic Backup Strategy	1-11
Why Are Backups Important?	1-11
What Types of Failures Can Occur?	1-12
What Should You Back Up?.....	1-13
Which Backup Method Should You Use?.....	1-13
Should You Make Consistent or Inconsistent Backups?.....	1-16
How Often Should You Make Backups?.....	1-17

Understanding Basic Recovery Strategy	1-17
How Does Recovery Work?	1-18
What Are the Types of Recovery?	1-18
What Is Media Recovery?	1-19
Which Recovery Method Should You Use?	1-24

2 Managing Data Structures

Overview of Backup and Recovery Data Structures	2-2
Managing the Control File	2-3
Displaying Control File Information	2-4
Backing Up the Control File After Structural Changes	2-5
Maintaining Multiple Control Files	2-7
Recovering from the Loss of Control Files	2-9
Managing the Online Redo Log	2-11
Displaying Online Redo Log Information	2-12
Multiplexing Online Redo Log Files	2-12
Managing the Archived Redo Logs	2-14
Choosing Between NOARCHIVELOG and ARCHIVELOG Mode	2-15
Displaying Archived Redo Log Information	2-18
Archiving Redo Logs to Multiple Locations	2-19

3 Developing a Backup and Recovery Strategy

Choosing Backup Types	3-2
Whole Database Backups	3-2
Tablespace Backups	3-6
Datafile Backups	3-8
Control File Backups	3-8
Choosing Backup Methods	3-8
Recovery Manager	3-9
Operating System (O/S)	3-10
Export	3-10
Enterprise Backup Utility	3-11
Feature Comparison of Backup Methods	3-11
Choosing Backup Formats	3-12
Backup Sets	3-12

Image Copies	3-13
Operating System Backups	3-14
Logical Backups	3-14
Developing a Backup Strategy	3-14
Decide Whether to Run in ARCHIVELOG or NOARCHIVELOG Mode	3-15
Multiplex Control Files, Online Redo Logs, and Archived Redo Logs	3-17
Perform Backups Frequently and Regularly	3-18
Perform Backups When You Make Structural Changes	3-18
Perform Frequent Backups of Often-Used Tablespaces	3-19
Perform Backups After Unrecoverable/Unlogged Operations	3-19
Perform Whole Database Backups After Using the RESETLOGS Option	3-19
Store Older Backups	3-21
Know the Constraints for Distributed Database Backups	3-22
Export Database Data for Added Protection and Flexibility	3-22
Do Not Back Up Online Redo Logs	3-23
Developing a Recovery Strategy	3-24
Test Backup and Recovery Strategies	3-24
Handling Non-Media Failures	3-24
Recovering from Media Failure	3-27

Part II Using Recovery Manager

4 Recovery Manager Concepts

Overview of Recovery Manager	4-2
Recovery Manager Features	4-4
What Recovery Manager Is Not	4-4
Recovery Manager Commands	4-5
Recovery Manager PL/SQL Packages	4-5
How Recovery Manager Compiles and Executes Commands	4-6
Types of Recovery Manager Commands	4-7
User Execution of Recovery Manager Commands	4-10
Recovery Manager Command Errors	4-12
Recovery Manager Metadata	4-13
Storage of Metadata in the Recovery Catalog	4-13
Storage of Metadata Exclusively in the Control File	4-15

Media Management	4-16
Backup and Restore Operations Using a Media Manager.....	4-17
Media Manager Crosschecks	4-18
Proxy Copy	4-19
Media Manager Testing	4-19
Backup Solutions Program	4-20
Lists and Reports	4-20
Lists of Backups and Copies.....	4-21
Reports on Backups, Copies, and Database Schema	4-22
Channel Allocation	4-24
Channel Control Options.....	4-25
Channel Parallelization	4-26
Backup Sets	4-28
Storage of Backup Sets	4-29
Backup Set Compression	4-30
Filenames for Backup Pieces	4-31
Size of Backup Pieces	4-31
Number of Backup Sets.....	4-31
Multiplexed Backup Sets	4-32
Duplexed Backup Sets.....	4-33
Parallelization of Backups	4-34
Backup Errors.....	4-36
Backup Types	4-36
Full Backups	4-37
Incremental Backups	4-38
Backup Constraints.....	4-44
Image Copies	4-45
RMAN Image Copies	4-45
O/S Image Copies	4-45
Tags for Backups and Image Copies.....	4-46
Restoring Files	4-47
File Selection in Restore Operations.....	4-48
Restore Constraints.....	4-48
Media Recovery	4-49
Application of Incremental Backups and Redo Records	4-51

Incomplete Recovery.....	4-51
Tablespace Point-in-Time Recovery	4-51
Database Duplication	4-52
Integrity Checks	4-54
Detection of Physical Block Corruption	4-54
Detection of Logical Block Corruption.....	4-55
Detection of Fractured Blocks During Open Backups	4-55

5 Getting Started with Recovery Manager

Setting Up Recovery Manager	5-2
Using Password Files	5-2
Setting NLS Environment Variables.....	5-2
Determining the Snapshot Control File Location	5-3
Deciding Whether to Use a Recovery Catalog	5-4
Consequences of Using the Recovery Catalog for RMAN Metadata	5-4
Consequences of Using the Control File for RMAN Metadata	5-6
Connecting to RMAN	5-8
Connecting to RMAN Without a Recovery Catalog	5-9
Connecting to RMAN with a Recovery Catalog.....	5-10
Connecting to an Auxiliary Database.....	5-11
Disconnecting from RMAN	5-12
Using Basic RMAN Commands	5-12
Connecting to RMAN	5-13
Mounting the Database.....	5-14
Reporting the Current Schema	5-14
Copying a Datafile.....	5-15
Backing Up a Tablespace.....	5-16
Listing Backups and Copies.....	5-17
Validating a Restore	5-18
Configuring a Media Manager	5-19
Linking with a Media Manager	5-19
Generating Unique Filenames	5-19
Limiting File Size	5-20
Sending Device-Specific Strings to the Media Manager	5-20
Troubleshooting the Media Manager	5-21

Using Sample Scripts and Scenarios.....	5-21
---	------

6 Managing Recovery Manager Metadata

Creating the Recovery Catalog	6-2
Maintaining RMAN Metadata	6-4
Registering a Database with the Recovery Catalog.....	6-4
Unregistering a Database from the Recovery Catalog.....	6-6
Resetting the Recovery Catalog.....	6-7
Changing the Availability of a Backup or File Copy.....	6-8
Crosschecking RMAN Metadata.....	6-9
Deleting Backups and Copies and Updating Their Status in the RMAN Metadata.....	6-13
Validating the Restore of Backups and Copies.....	6-18
Storing Scripts in the Recovery Catalog	6-20
Resynchronizing the Recovery Catalog.....	6-23
Managing Records in the Control File.....	6-26
Cataloging O/S Backups.....	6-27
Backing Up and Recovering the Recovery Catalog	6-30
Backing Up the Recovery Catalog.....	6-30
Recovering the Recovery Catalog.....	6-32
Re-Creating the Recovery Catalog.....	6-33
Upgrading the Recovery Catalog	6-34
Dropping the Recovery Catalog	6-35
Managing RMAN Metadata Without a Recovery Catalog	6-35
Maintaining the Control File Metadata.....	6-36
Backing Up the Control File.....	6-36

7 Generating Lists and Reports with Recovery Manager

Using Lists and Reports in Your Backup and Recovery Strategy	7-2
Generating Lists	7-2
Generating Reports	7-5
List and Report Scenarios	7-9
Making Lists of Backups and Copies.....	7-10
Using Lists to Determine Obsolete Backups and Copies.....	7-10
Reporting Datafiles Needing Backups.....	7-10
Reporting Unrecoverable Datafiles.....	7-11

Reporting Obsolete Backups and Copies.....	7-11
Deleting Obsolete Backups	7-11
Generating Historical Reports of Database Schema.....	7-12

8 Making Backups and Copies with Recovery Manager

Making Backups	8-2
Making Consistent and Inconsistent Backups.....	8-3
Making Whole Database Backups.....	8-3
Backing Up Tablespaces and Datafiles.....	8-4
Backing Up Control Files.....	8-6
Backing Up Archived Redo Logs	8-8
Making Incremental Backups	8-8
Making Image Copies	8-10
Monitoring Backup and Copy Operations	8-12
Correlating Server Sessions with Channels.....	8-12
Monitoring Progress.....	8-13
Monitoring Performance	8-14
Backup and Copy Scenarios	8-14
Reporting Datafiles Needing Backups	8-15
Skipping Files when Backing Up a Database	8-15
Spreading a Backup Across Multiple Disk Drives	8-15
Backing Up a Large Database to Multiple Filesystems.....	8-16
Specifying the Size of Backup Sets.....	8-16
Multiplexing Datafiles in a Backup.....	8-18
Backing Up Archived Redo Logs	8-19
Backing Up and Deleting Multiple Copies of an Archived Redo Log.....	8-20
Performing Differential Incremental Backups	8-21
Performing Cumulative Incremental Backups.....	8-22
Duplexing Backup Sets	8-22
Parallelizing Backup Sets.....	8-23
Backing Up in NOARCHIVELOG Mode.....	8-23
Backing Up in a Parallel Server Environment.....	8-25
Cataloging O/S Copies.....	8-25
Maintaining Backups and Copies	8-26
Handling Errors During Backups and Copies	8-26

9 Restoring and Recovering with Recovery Manager

Restoring Datafiles, Control Files, and Archived Redo Logs	9-2
Restoring a Database.....	9-2
Restoring Tablespaces and Datafiles.....	9-9
Restoring Control Files	9-11
Restoring Archived Redo Logs.....	9-13
Restoring in Preparation for Incomplete Recovery	9-15
Recovering Datafiles	9-15
Preparing for Media Recovery.....	9-17
Performing Complete Recovery	9-18
Performing Incomplete Recovery.....	9-23
Restore and Recovery Scenarios	9-30
Using Datafile Copies to Restore to a New Host	9-30
Restoring when Multiple Databases Share the Same Name	9-31
Performing an Irregular Restore of the Control File from a Backup Set	9-33
Recovering an Inaccessible Datafile in an Open Database	9-34
Recovering an Inaccessible Datafile Using Backups from Disk and Tape	9-35
Performing Recovery After a Total Media Failure	9-35
Recovering a Pre-RESETLOGS Backup.....	9-37
Recovering a Database in NOARCHIVELOG Mode.....	9-38

10 Creating a Duplicate Database with Recovery Manager

Creating a Duplicate Database: Overview	10-2
Obeying Restrictions	10-3
Generating Files for the Duplicate Database	10-3
Preparing the Auxiliary Instance for Duplication	10-7
Creating a Duplicate Database on a Local or Remote Host	10-10
Duplicating a Database on a Remote Host with the Same Directory Structure	10-10
Duplicating a Database on a Remote Host with a Different Directory Structure	10-12
Creating a Duplicate Database on the Local Host	10-17
Duplication Scenarios	10-17
Setting New Filenames Manually	10-17
Resynchronizing the Duplicate Database with the Target Database.....	10-19
Creating a Non-Current Duplicate Database	10-19

Part III Recovery Manager Reference

11 Recovery Manager Command Syntax

Conventions Used in this Reference	11-2
Command Entries	11-4
Summary of RMAN Commands	11-5
allocate	11-9
allocateForMaint	11-13
alterDatabase	11-15
archivelogRecordSpecifier	11-17
backup	11-21
catalog	11-32
change	11-35
cmdLine	11-39
completedTimeSpec	11-42
connect	11-44
connectStringSpec	11-46
copy	11-48
createCatalog	11-52
createScript	11-54
crosscheck	11-57
datafileSpec	11-60
debug	11-62
deleteExpired	11-63
deleteScript	11-65
deviceSpecifier	11-66
dropCatalog	11-68
duplicate	11-69
host	11-74
list	11-76
listObjList	11-84
printScript	11-86
recover	11-88
register	11-93
release	11-95

releaseForMaint	11-96
replaceScript	11-97
replicate	11-100
report	11-102
reset	11-110
restore	11-112
resync	11-118
rmanCmd	11-121
run	11-124
send	11-127
set	11-129
set_run_option	11-133
shutdown	11-137
sql	11-140
startup	11-142
switch	11-144
untilClause	11-146
upgradeCatalog	11-148
validate	11-150

12 Recovery Catalog Views

RC_ARCHIVED_LOG	12-2
RC_BACKUP_CONTROLFILE	12-3
RC_BACKUP_CORRUPTION	12-4
RC_BACKUP_DATAFILE	12-6
RC_BACKUP_PIECE	12-8
RC_BACKUP_REDOLOG	12-9
RC_BACKUP_SET	12-11
RC_CHECKPOINT	12-12
RC_CONTROLFILE_COPY	12-12
RC_COPY_CORRUPTION	12-14
RC_DATABASE	12-15
RC_DATABASE_INCARNATION	12-15
RC_DATAFILE	12-16
RC_DATAFILE_COPY	12-17

RC_LOG_HISTORY.....	12-18
RC_OFFLINE_RANGE.....	12-19
RC_PROXY_CONTROLFILE.....	12-20
RC_PROXY_DATAFILE.....	12-22
RC_REDO_LOG.....	12-24
RC_REDO_THREAD.....	12-24
RC_RESYNC.....	12-25
RC_STORED_SCRIPT.....	12-25
RC_STORED_SCRIPT_LINE.....	12-26
RC_TABLESPACE.....	12-26

Part IV Performing Operating System Backup and Recovery

13 Performing Operating System Backups

Listing Database Files Before Performing a Backup	13-2
Performing O/S Backups	13-3
Performing Whole Database Backups.....	13-3
Performing Tablespace and Datafile Backups.....	13-6
Performing Control File Backups.....	13-11
Recovering From a Failed Online Tablespace Backup	13-14
Using Export and Import for Supplemental Protection	13-16
Using Export.....	13-16
Using Import.....	13-18

14 Performing Operating System Recovery

What Is Media Recovery?	14-2
Restoring Files.....	14-2
Recovering Datafiles.....	14-2
Determining Which Files to Recover	14-3
Restoring Files	14-6
Restoring Backup Datafiles.....	14-6
Re-Creating Datafiles when Backups Are Unavailable.....	14-6
Restoring Necessary Archived Redo Log Files.....	14-7
Understanding Basic Media Recovery Procedures	14-8

Using Media Recovery Statements.....	14-9
Applying Archived Redo Logs.....	14-10
Recovering a Database in NOARCHIVELOG Mode.....	14-17
Recovering a Database in ARCHIVELOG Mode.....	14-19
Performing Media Recovery in Parallel.....	14-19
Performing Complete Media Recovery.....	14-20
Performing Closed Database Recovery.....	14-20
Performing Open Database Recovery.....	14-23
Performing Incomplete Media Recovery.....	14-26
Performing Cancel-Based Recovery.....	14-26
Performing Time-Based Recovery.....	14-29
Performing Change-Based Recovery.....	14-32
Opening the Database after Media Recovery.....	14-33
Deciding Whether to Specify RESETLOGS or NORESETLOGS.....	14-33
Guidelines for Opening in RESETLOGS Mode.....	14-34
Recovering a Pre-RESETLOGS Backup.....	14-35

15 Operating System Recovery Scenarios

Understanding the Types of Media Failures.....	15-2
Recovering After the Loss of Datafiles.....	15-2
Losing Datafiles in NOARCHIVELOG Mode.....	15-2
Losing Datafiles in ARCHIVELOG Mode.....	15-3
Recovering Through an ADD DATAFILE Operation.....	15-3
Recovering Transported Tablespaces.....	15-4
Recovering After the Loss of Online Redo Log Files.....	15-5
Recovering After Losing a Member of a Multiplexed Online Redo Log Group.....	15-5
Recovering After the Loss of All Members of an Online Redo Log Group.....	15-6
Recovering After the Loss of Archived Redo Log Files.....	15-10
Recovering After the Loss of Control Files.....	15-11
Losing a Member of a Multiplexed Control File.....	15-12
Losing All Copies of the Current Control File.....	15-13
Recovering from User Errors.....	15-14
Performing Media Recovery in a Distributed Environment.....	15-15
Coordinating Time-Based and Change-Based Distributed Database Recovery.....	15-16
Recovering a Database with Snapshots.....	15-17

16 Managing a Standby Database

Planning a Standby Database	16-2
Standby Database Advantages	16-2
Standby Database Requirements	16-3
Creating a Standby Database	16-3
Choosing the Standby Database Mode	16-5
Maintaining a Standby Database in Recovery Mode	16-6
Placing the Standby in Manual Recovery Mode	16-6
Placing the Standby Database in Managed Recovery Mode	16-7
Transmitting Archived Redo Logs to a Standby Database	16-9
Maintaining the Standby Database in Recovery Mode	16-15
Opening a Standby Database in Read-Only Mode	16-19
Activating a Standby Database	16-21
Altering the Physical Structure of the Primary Database	16-22
Adding Datafiles	16-23
Renaming Datafiles	16-24
Altering Redo Logs	16-24
Altering Control Files	16-25
Configuring Initialization Parameters	16-25
Taking Datafiles in the Standby Database Offline	16-27
Performing Direct Path Operations	16-27
Refreshing the Standby Database Control File	16-29
Using a Standby Database in an OPS Configuration	16-29

A Performing Tablespace Point-in-Time Recovery with Recovery Manager

Introduction to RMAN TSPITR	A-2
Planning for TSPITR	A-4
Performing TSPITR Without a Recovery Catalog	A-4
Understanding General Restrictions	A-5
Researching and Resolving Inconsistencies	A-6
Managing Data Relationships	A-7
Preparing the Auxiliary Instance for TSPITR	A-7
Performing TSPITR	A-10
Preparing the Target Database for Use After TSPITR	A-11
Responding to Unsuccessful TSPITR	A-12

Tuning TSPITR Performance	A-13
Specify a New Name for Datafiles in Auxiliary Set Tablespaces	A-13
Set the Auxiliary Name and Use a Datafile Copy for Recovery Manager TSPITR.....	A-14
Use the Converted Filename in the Auxiliary Control File	A-15
Summary: Datafile Naming Methods.....	A-15

B Performing Operating System Tablespace Point-in-Time Recovery

Introduction to O/S Tablespace Point-in-Time Recovery	B-2
TSPITR Advantages.....	B-2
TSPITR Methods	B-3
TSPITR Terminology.....	B-4
Planning for Tablespace Point-in-Time Recovery	B-4
TSPITR Limitations.....	B-5
TSPITR Requirements	B-6
Preparing the Databases for TSPITR	B-6
Step 1: Determine Whether Objects Will Be Lost	B-7
Step 2: Research and Resolve Dependencies on the Primary Database	B-7
Step 3: Prepare the Primary Database	B-9
Step 4: Prepare the Clone Parameter Files	B-10
Step 5: Prepare the Clone Database	B-11
Performing TSPITR	B-12
Step 1: Recover the Clone Database	B-12
Step 2: Open the Clone Database	B-13
Step 3: Prepare the Clone Database for Export	B-13
Step 4: Export the Metadata	B-13
Step 5: Copy the Recovery Set Clone Files to the Primary Database	B-13
Step 6: Import the Metadata into the Primary Database	B-13
Step 7: Prepare the Primary Database for Use	B-14
Step 8: Back Up the Recovered Tablespaces in the Primary Database	B-14
Performing Partial TSPITR of Partitioned Tables	B-15
Step 1: Create a Table on the Primary Database for Each Partition Being Recovered....	B-15
Step 2: Drop the Indexes on the Partition Being Recovered.....	B-16
Step 3: Exchange Partitions with Stand-Alone Tables.....	B-16
Step 4: Take the Recovery Set Tablespace Offline.....	B-16
Step 5: Create Tables at Clone Database.....	B-16

Step 6: Drop Indexes on Partitions Being Recovered	B-16
Step 7: Exchange Partitions with Stand-Alone Tables	B-16
Step 8: Export the Clone Database	B-17
Step 9: Copy the Recovery Set Datafiles to the Primary Database	B-17
Step 10: Import into the Primary Database	B-17
Step 11: Bring Recovery Set Tablespace Online	B-17
Step 12: Exchange Partitions with Stand-Alone Tables	B-17
Step 13: Back Up the Recovered Tablespaces in the Primary Database	B-18
Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped	B-18
Step 1: Find the Low and High Range of the Partition that Was Dropped.....	B-19
Step 2: Create a Temporary Table	B-19
Step 3: Delete Records From Partitioned Table.....	B-19
Step 4: Take Recovery Set Tablespaces Offline	B-19
Step 5: Create Tables at Clone Database	B-19
Step 6: Drop Indexes on Partitions Being Recovered	B-19
Step 7: Exchange Partitions with Stand-Alone Tables	B-20
Step 8: Export the Clone Database	B-20
Step 9: Copy the Recovery Set Datafiles to the Primary Database	B-20
Step 10: Import into the Primary Database.....	B-20
Step 11: Bring Recovery Set Tablespace Online	B-20
Step 12: Insert Stand-Alone Tables into Partitioned Tables	B-21
Step 13: Back Up the Recovered Tablespaces in the Primary Database	B-21
Performing TSPITR of Partitioned Tables When a Partition Has Split	B-22
Step 1: Drop the Lower of the Two Partitions at the Primary Database	B-22
Steps 2-13: Follow Same Steps as for Partial TSPITR of Partitioned Tablespaces.....	B-23
TSPITR Tuning Considerations	B-23
Recovery Set Location Considerations	B-23
Backup Control File Considerations	B-24
Performing TSPITR Using Transportable Tablespaces.....	B-25

Glossary

Index

Send Us Your Comments

Oracle8i Backup and Recovery Guide, 8.1.5

A67773-01

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

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

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

- email: infodev@us.oracle.com
- fax: (650) 506-7228 Attn: Server Technologies Documentation Manager
- letter: Server Technologies Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

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

Preface

Welcome to the world of Oracle backup and recovery. This guide includes the conceptual and task-oriented information you will need to perform backup, restore, and recovery procedures whether you use the Recovery Manager utility or operating system commands.

Note: The *Oracle8i Backup and Recovery Guide* contains information that describes the features and functionality of the Oracle8i Standard Edition and the Oracle8i Enterprise Edition products. The Standard Edition and Enterprise Edition have the same basic features, but several advanced features are available only with the Enterprise Edition, and some of these are optional. For example, to use Recovery Manager to perform automated tablespace point-in-time recovery, you must have the Enterprise Edition.

For information about the differences between Oracle8i Standard Edition and the Oracle8i Enterprise Edition and the features and options that are available to you, please refer to *Getting to Know Oracle8i*.

Structure

This book contains the following parts and chapters.

Part / Chapter	Contents
PART 1	Developing a Backup and Recovery Strategy
Chapter 1, "What Is Backup and Recovery?"	Offers a general overview of backup and recovery concepts and methods.
Chapter 2, "Managing Data Structures"	Describes how to manage control files, online redo logs, and archived redo logs for backup and recovery.
Chapter 3, "Developing a Backup and Recovery Strategy"	Provides guidelines for developing a backup and recovery strategy.
PART 2	Using Recovery Manager
Chapter 4, "Recovery Manager Concepts"	Describes the features and functionality of the Recovery Manager utility.
Chapter 5, "Getting Started with Recovery Manager"	Describes how to start using Recovery Manager, including whether you should use a recovery catalog, how you can execute commands, and how to connect to a target database.
Chapter 6, "Managing Recovery Manager Metadata"	Describes how to manage RMAN metadata using either a recovery catalog or the control file.
Chapter 7, "Generating Lists and Reports with Recovery Manager"	Describes how to generate lists of your backups and copies and reports describing which datafiles need to be backed up, which backups are obsolete, and the structure of the database schema at a specified time.
Chapter 8, "Making Backups and Copies with Recovery Manager"	Describes how to make backup and image copies of datafiles, control files, and archived redo logs using Recovery Manager.
Chapter 9, "Restoring and Recovering with Recovery Manager"	Describes how to restore backups and copies of datafiles, control files, and archived redo logs and perform media recovery on datafiles.
Chapter 10, "Creating a Duplicate Database with Recovery Manager"	Describes how to create a duplicate of your target database on either a local or remote host using backups of your target database datafiles.
PART 3	Recovery Manager Reference
Chapter 11, "Recovery Manager Command Syntax"	Provides syntax diagrams, parameter descriptions, and code samples for all RMAN commands.
Chapter 12, "Recovery Catalog Views"	Describes the views available with a recovery catalog.

Part / Chapter	Contents
PART 4	Using O/S Commands for Backup and Recovery
Chapter 13, "Performing Operating System Backups"	Provides step-by-step instructions for performing operating system backups.
Chapter 14, "Performing Operating System Recovery"	Provides step-by-step instructions for performing media recovery using operating system commands.
Chapter 15, "Operating System Recovery Scenarios"	Describes the procedures for performing media recovery in several common scenarios.
Chapter 16, "Managing a Standby Database"	Explains how to plan, create, and maintain a standby database.
Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager"	Provides planning guidelines and step-by-step instructions for performing tablespace point-in-time recovery with Recovery Manager.
Appendix B, "Performing Operating System Tablespace Point-in-Time Recovery"	Provides planning guidelines and step-by-step instructions for manually performing tablespace point-in-time recovery.

Audience

This guide is for DBAs who administer the backup, restore, and recovery operations of an Oracle database system.

Knowledge Assumed of the Reader

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

Conventions

This section explains the conventions used in this manual including the following:

- Text
- Syntax diagrams and notation
- Code examples

Text

This section explains the conventions used within the text:

UPPERCASE Characters

Uppercase text is used to call attention to tablespace names, initialization parameters, and SQL keywords.

For example, "If you create a private rollback segment, the name must be included in the `ROLLBACK_SEGMENTS` parameter of the `init.ora` file. You can view this information by issuing a `SHOW PARAMETER` statement in `SQL*Plus`."

Italicized Characters

Italicized words within text are book titles, new vocabulary, emphasized words, or variables in SQL or Recovery Manager syntax.

For example, "An *archived redo log* is an online redo log that has been copied offline. You *must* run your database in ARCHIVELOG mode to enable this feature. If you are using Recovery Manager, you can specify an archived redo log in a **backup** command by using the **archivelog like** `'/oracle/archive/arc_*` sub-clause."

Bold Characters

Bold words within text are Recovery Manager keywords or operating system-specific commands.

For example, "Use the Recovery Manager **backup** command to back up your database. Alternatively, use the UNIX **cp** command to copy files."

Monospaced Characters

Filenames and directories appear in a monospaced font. Also, monospaced characters in text preceding a code example indicates a filename or keyword used in the sample code.

For example, "This command backs up the tablespace `TBS_1`:

```
run {
    allocate channel c1 type disk;
    backup tablespace tbs_1;
}
```

Recovery Manager Syntax Diagrams and Notation

For information about Recovery Manager syntax conventions, see "[Conventions Used in this Reference](#)" on page 11-2.

Code Examples

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

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
run {
    allocate channel ch1 type disk;
    backup database;
}
```

When you run RMAN from the command line, the command prompt appears as RMAN>. When you issue commands from the SQL*Plus command line, the prompt appears as SQL>. These prompts are displayed in the code examples only when they are necessary to prevent confusion.

You can execute SQL, SQL*Plus, and RMAN commands in different environments on different platforms. As much as possible, this guide attempts to provide generic documentation, that is, documentation that is not specific to any operating system or interface. Nevertheless, it is sometimes necessary for illustrative purposes to show how the syntax works at the O/S level. In these cases, this book uses examples from a UNIX command-line interface and employs the % symbol to indicate the O/S prompt. For example:

```
% rman target / rcvcat rman/rman@inst2
RMAN> startup
```

How to Use This Guide

Every reader of this guide is presumed to have read:

- The beginning of the *Oracle8i Concepts* manual, which provides an overview of the concepts and terminology related to Oracle and a foundation for the more detailed information in this guide. The rest of the *Oracle8i Concepts* manual explains the Oracle architecture and features in detail.
- The chapters in the *Oracle8i Administrator's Guide* that deal with managing the control file, online redo logs, and archived redo logs.

You will often need to refer to the following reference guides:

- *Oracle8i SQL Reference*
- *Oracle8i Reference*

Part I

Developing a Backup and Recovery Strategy

What Is Backup and Recovery?

This chapter introduces database concepts that are fundamental to backup and recovery. It is intended as a general overview. Subsequent chapters explore backup and recovery concepts in greater detail.

You will learn about the following topics:

- [What Is Backup and Recovery?](#)
- [Which Data Structures Are Important for Backup and Recovery?](#)
- [Understanding Basic Backup Strategy](#)
- [Understanding Basic Recovery Strategy](#)

What Is Backup and Recovery?

Simply speaking, a *backup* is a copy of data. This copy includes important parts of your database such as the control file and datafiles. A backup is a safeguard against unexpected data loss and application errors; should you lose your original data, you can use the backup to make it available again.

Backups are divided into *physical backups* and *logical backups*. Physical backups, which are the primary concern of this guide, are copies of physical database files. In contrast, logical backups contain data that you extract using the Oracle Export utility and store in a binary file. You can use logical backups to supplement physical backups. You can make physical backups using either the Oracle8i Recovery Manager utility or O/S utilities.

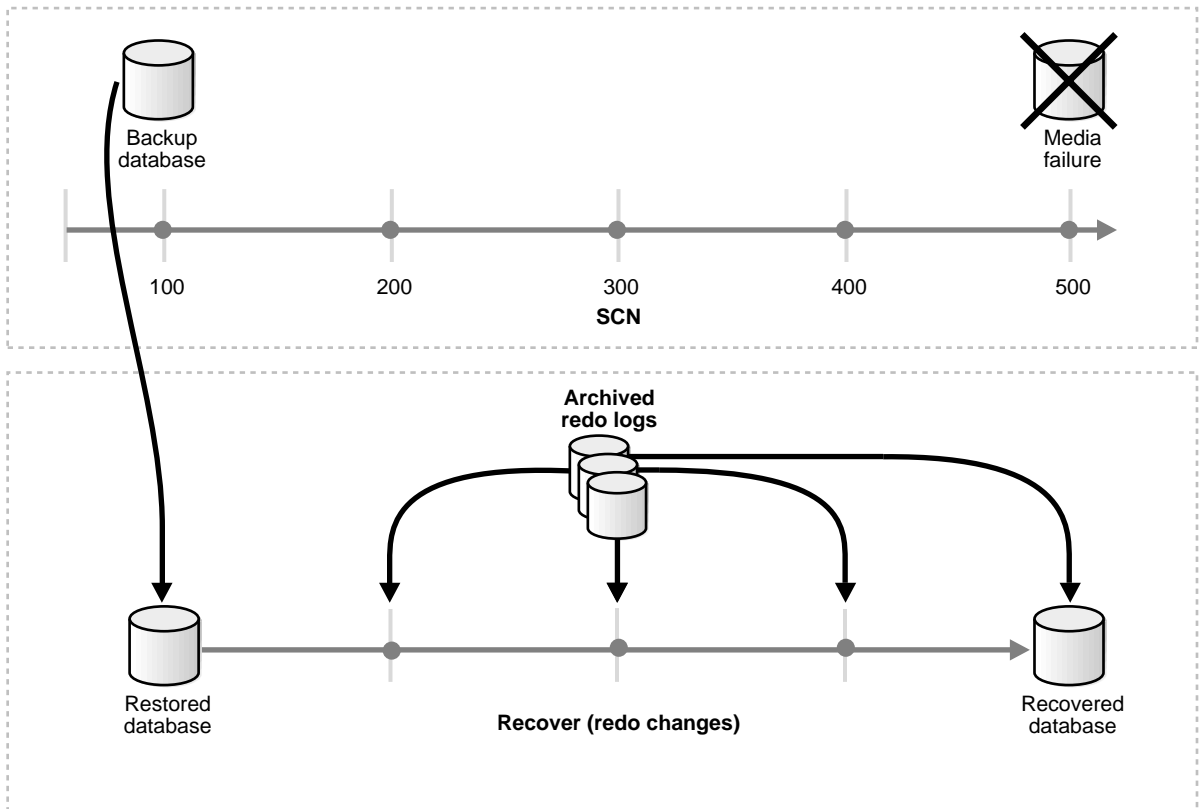
To *restore* a physical backup is to reconstruct it and make it available to the Oracle server. To *recover* a restored datafile is to update it using *redo records*, i.e., records of changes made to the database after the backup was taken. If you use Recovery Manager (RMAN), you can also recover restored datafiles with an *incremental backup*, which is a backup of a datafile that contain only changed data blocks.

Oracle performs *crash recovery* and *instance recovery* automatically after an instance failure. Instance recovery is an automatic procedure that involves two distinct operations: *rolling forward* the backup to a more current time by applying online redo records and *rolling back* all changes made in uncommitted transactions to their original state.

In contrast to instance recovery, *media recovery* requires you to issue recovery commands. If you use RMAN, then you issue the **recover** command to apply archived redo logs or incremental backups to the datafiles. RMAN automatically selects the appropriate incremental backups or redo logs and applies them. If you use SQL*Plus, you can issue the RECOVER or ALTER DATABASE RECOVER statements to apply the archived logs.

Figure 1-1 illustrates the basic principle of backing up, restoring, and performing media recovery on a database:

Figure 1–1 Restoring and Recovering a Database



Recovery in general refers to the various operations involved in restoring, rolling forward, and rolling back a backup. *Backup and recovery* in general refers to the various strategies and operations involved in protecting your database against data loss and reconstructing the data should that loss occur.

See Also: For an overview of Recovery Manager features, see [Chapter 4, "Recovery Manager Concepts"](#). To learn how to perform O/S backup and recovery, see [Chapter 13, "Performing Operating System Backups"](#) and [Chapter 14, "Performing Operating System Recovery"](#).

How Oracle Keeps Records of Database Transactions

To understand the basics of backup and recovery, you need to understand how Oracle records changes to a database. Every time a change is made against the database, Oracle generates a record of the change in the *redo log buffer* in memory. This record is called a redo record. Oracle records both committed and uncommitted changes in redo log buffers.

Oracle frequently writes the redo log buffers to the *online redo log*, which is on disk. The online redo log is constituted by at least two online redo log files. Oracle writes to these logs in a circular fashion: first it writes to one log file, then switches to the next available file when the current log is full.

Depending on whether Oracle runs in ARCHIVELOG or NOARCHIVELOG mode, the system begins *archiving* the redo information in the non-current online redo log file by copying the file to specified locations on disk. The online and archived redo logs are crucial for recovery since they contain records of all changes to the database.

See Also: For an overview of the online and archived redo logs, see *Oracle8i Concepts*. To learn the basics of administering these data structures, see the *Oracle8i Administrator's Guide*. To learn how to manage these structures for backup and recovery, see [Chapter 2, "Managing Data Structures"](#).

Backup and Recovery Operations

A physical backup is a snapshot of a datafile, tablespace, or database at a certain time. If you make periodic backups of a database, then should you lose some of the data on your original database, you can perform media recovery using the backups. During media recovery, you apply redo records or incremental backups to your latest backup to make the database current again. If the backup was a *consistent backup*, then you can also restore the backup without performing recovery.

Oracle enables you to restore an older backup and apply only some redo data, thereby recovering the database to a specified time or SCN. This type of recovery is called *incomplete recovery*. You must open your database with a RESETLOGS operation after performing incomplete recovery in order to reset the online redo logs.

An example will help illustrate the concept of media recovery. Assume that you make a backup of your database at noon. Starting at noon, one change to the database is made every second. The redo logs switch three times over the next hour, and because the database runs in ARCHIVELOG mode, these redo logs are archived to disk. At one p.m. your disk drive fails. Fortunately, you can restore your

noon whole database backup onto a functional disk drive and, using the archived redo logs to recover the database to one p.m., reconstruct the changes.

See Also: To learn how to make consistent backups using O/S methods, see ["Making Consistent Whole Database Backups"](#) on page 13-4. To learn how to perform media recovery, see [Chapter 14, "Performing Operating System Recovery"](#).

Elementary Backup and Recovery Strategy

Although backup and recovery operations can sometimes be complicated, the basic principles for developing an effective strategy are simple:

1. Maintain multiple identical copies of your online redo logs on different disks.
2. Archive your redo logs to multiple locations or make frequent backups of your archived redo logs.
3. Maintain multiple, concurrent copies of your control file. See ["Control Files"](#) on page 1-7.
4. Take frequent backups of your datafiles and control file and store them in a safe place (on more than one media, if possible).

Most backup and recovery techniques are variations on these principles. So long as you have backups of your datafiles, control files, and archived redo logs in safe storage, then even if a fire were to destroy your hardware, you can recreate your original database.

A sophisticated and effective disaster prevention technique is to maintain a standby database, which is an exact replicate of your production database that you can update automatically with archived redo logs propagated through a Net8 connection.

See Also: To learn more about managing important data structures such as the online redo logs, see [Chapter 2, "Managing Data Structures"](#). To learn more about developing a backup and recovery strategy, see [Chapter 3, "Developing a Backup and Recovery Strategy"](#). To learn how to maintain a standby database, see [Chapter 16, "Managing a Standby Database"](#).

Which Data Structures Are Important for Backup and Recovery?

Before you can begin thinking seriously about backup and recovery strategy, you need to understand the physical data structures that are relevant for backup and recovery operations. In this section we will briefly discuss:

- [Datafiles](#)
- [Control Files](#)
- [Rollback Segments](#)
- [Online Redo Log Files](#)
- [Archived Redo Log Files](#)

This manual will cover these topics in more detail in subsequent chapters.

See Also: For a complete overview of the Oracle8i architecture, see *Oracle8i Concepts*. To learn how to manage these data structures, see the *Oracle8i Administrator's Guide*.

Datafiles

Every Oracle database has one or more physical *datafiles*. A database's datafiles, which belong to logical structures called *tablespaces*, contain the database data. The datafile is divided into smaller units called *data blocks*. The data of logical database structures such as tables and indexes is physically located in the blocks of the datafiles allocated for a database.

The first block of every datafile is the header. The header includes important information such as file size, block size, tablespace, creation timestamp, and checkpoint SCN (see "[System Change Number \(SCN\)](#)" on page 1-9). Whenever you open a database, Oracle checks the datafile header information against the information stored in the control file to determine whether recovery is necessary.

The Use of Datafiles

Oracle reads the data in a datafile during normal operation and stores it in the in-memory buffer cache. For example, assume that a user wants to access some data in a table. If the requested information is not already in the buffer cache, Oracle reads it from the appropriate datafiles and stores it in memory.

The background process DBWn, known as the *database writer* or *db writer*, writes modified buffers to disk. Typically, there is a time lapse between the time when an Oracle server process changes a block in the buffer cache and the time when DBWn

writes it to disk. The more data that accumulates in memory without being written to disk, the longer the instance recovery time, since a crash or media failure will force Oracle to apply redo data from the current online log to recover those changes. Minimizing this time, known as the *mean time to recover (MTTR)*, is an important aspect of backup and recovery strategy.

See Also: For more information on parameters that you can use to influence the MTTR, see *Oracle8i Tuning*.

Control Files

Every Oracle database has a *control file*. A control file is an extremely important binary file that contains the operating system filenames of all other files constituting the database. It also contains consistency information that is used during recovery, such as:

- The database name.
- The timestamp of database creation.
- The names of the database's datafiles and online and archived redo log files.
- The *checkpoint*, i.e., a record indicating the point in the redo log such that all database changes prior to this point have been saved in the datafiles.
- Information on backups (when using the Recovery Manager utility).

When *multiplexing* the control file, you configure Oracle to write multiple copies of it in order to protect it against loss. If your operating system supports disk mirroring, you can also *mirroring* the control file, i.e., configure the O/S to write a copy of the control file to multiple disks.

See Also: For more information about multiplexing and mirroring the control file, see "[Maintaining Multiple Control Files](#)" on page 2-7. For general information about managing the control file, see *Oracle8i Administrator's Guide*.

The Use of Control Files

Every time you mount an Oracle database, its control file is used to identify the datafiles and online redo log files that must be opened for database operation. If the physical makeup of the database changes, e.g., a new datafile or redo log file is created, then Oracle modifies the database's control file to reflect the change.

Back up the control file whenever the set of files that makes up the database changes. Examples of structural changes include adding, dropping, or altering datafiles or tablespaces and adding or dropping online redo logs.

See Also: To learn how to back up the control file, see "[Backing Up the Control File After Structural Changes](#)" on page 2-5.

Rollback Segments

Every database contains one or more *rollback segments*, which are logical structures contained in datafiles. Whenever a transaction modifies a data block, a rollback segment records the state of the information before it changed.

The Use of Rollback Segments

Oracle uses rollback segments for a variety of operations. In general, the rollback segments of a database store the old values of data changed by uncommitted transactions. Oracle can use the information in a rollback segment during database recovery to undo any uncommitted changes applied from the redo log to the datafiles, putting the data into a consistent state.

For example, a user changes a row value in a table from 5 to 7. The redo log keeps a record of this change while the rollback segment stores the old value. Before the user can commit the transaction, the power goes out. After power is restored, the database performs a crash recovery operation: it uses the redo log to roll forward the value from 5 to 7, then uses the rollback segment to undo the uncommitted change from 7 to 5.

Online Redo Log Files

Every Oracle database has a set of two or more *online redo log files*. Oracle assigns every redo log file a unique *log sequence number* to identify it. The set of redo log files for a database is collectively known as the database's *redo log*. Oracle uses the redo log to record all changes made to the database.

Oracle records every change in a *redo record*, which is an entry in the redo buffer describing what has changed. For example, assume a user updates a column value in a payroll table from 5 to 7. Oracle records the old value in a rollback segment and the new value in a redo record. Because the redo log stores every change to the database, the redo record for this transaction actually contains three parts:

- The change to the transaction table of the rollback segment.
- The change to the rollback segment data block.
- The change to the payroll table data block.

If you then *commit* the update to the payroll table, Oracle generates another redo record and assigns the change an SCN. In this way, the system maintains a careful watch over everything that occurs in the database.

Circular Use of Redo Log Files

The log writer background process (LGWR) writes to online redo log files in a circular fashion: when it fills the *current online redo log*, LGWR writes to the next available *inactive redo log*. LGWR cycles through the online redo log files in the database, writing over old redo data. Filled redo log files are available for reuse depending on whether archiving is enabled:

- If archiving is disabled, a filled online redo log is available once the changes recorded in the log have been saved to the datafiles.
- If archiving is enabled, a filled online redo log is available once the changes have been saved to the datafiles and the file has been archived.

System Change Number (SCN)

The *system change number (SCN)* is an ever-increasing value that uniquely identifies a committed version of the database. Every time a user commits a transaction, Oracle records a new SCN. You can obtain SCNs in a number of ways, for example, from the alert log. You can then use the SCN as an identifier for purposes of recovery. For example, you can perform an incomplete recovery of a database up to SCN 1030.

Oracle uses SCNs in control files, datafile headers, and redo records. Every redo log file has both a log sequence number and *low* and *high* SCN. The low SCN records the lowest SCN recorded in the log file, while the high SCN records the highest SCN in the log file.

The Use of Online Redo Logs

Redo logs are crucial for recovery. For example, suppose that a power outage prevents Oracle from permanently writing modified data to the datafiles. In this situation, you can use an old version of the datafiles combined with the changes recorded in the online and archived redo logs to reconstruct what was lost.

To protect against redo log failure, Oracle allows the redo log to be *multiplexed*. When Oracle multiplexes the online redo log, it maintains two or more copies of the redo log on different disks. Note that you should not back up the online redo log files, nor should you ever need to restore them; you keep redo logs by archiving them.

See Also: For more information on managing the online redo log for backup and recovery, see "[Managing the Online Redo Log](#)" on page 2-11. For a general discussion see *Oracle8i Administrator's Guide*.

Archived Redo Log Files

An *archived redo log* is an online redo log that Oracle has filled with redo entries, rendered inactive, and copied to one or more destinations specified in the parameter file. You can run Oracle in either of two archive modes:

ARCHIVELOG	Oracle archives the filled online redo log files before reusing them in the cycle.
NOARCHIVELOG	Oracle does not archive the filled online redo log files before reusing them in the cycle.

Running the database in ARCHIVELOG mode has the following consequences:

- You can completely recover the database from both instance and media failure.
- You can perform a *hot backup*, i.e., a backup made while the database is open and available for use.
- You can transmit and apply archived redo logs to a *standby database*, which is an exact replica of your primary database.
- You have more recovery options, e.g., you can perform incomplete recovery and tablespace point-in-time recovery (TSPITR).
- You have to perform additional administrative operations to store the archived redo logs.
- You require extra disk space to store the archived logs.

Running the database in NOARCHIVELOG mode has the following consequences:

- You can only back up the database while it is completely closed after a clean shutdown.
- Typically, your only media recovery option is to restore the whole database. You will lose all changes since the last whole database backup.
- Because no archived redo log is created, no additional administration is necessary.

Note: The *only* time you can recover a database while operating in NOARCHIVELOG mode is when you have not already overwritten the online log files that were current at the time of the most recent backup.

See Also: For more information on archiving redo logs, see "[Managing the Archived Redo Logs](#)" on page 2-14 and *Oracle8i Administrator's Guide*. For more information about the standby database option, see [Chapter 16, "Managing a Standby Database"](#).

Understanding Basic Backup Strategy

A physical backup is a copy of a datafile, control file, or archived redo log that you store as a safeguard against data loss. When thinking about a backup strategy, consider these questions:

- [Why Are Backups Important?](#)
- [What Types of Failures Can Occur?](#)
- [What Should You Back Up?](#)
- [Which Backup Method Should You Use?](#)
- [Should You Make Consistent or Inconsistent Backups?](#)
- [How Often Should You Make Backups?](#)

Why Are Backups Important?

Imagine the magnitude of lost revenue—not to mention the degree of customer dissatisfaction—if the production database of a catalog company, express delivery service, bank, or airline suddenly became unavailable, even for just 5 or 10 minutes. Alternatively, imagine that you lose important payroll datafiles due to a hard disk crash and cannot restore or recover them because you do not have a backup. Depending on the size and value of the lost information, the results can be devastating.

By making frequent backups, you ensure that you can restore at least some of your lost data. If you run your database in ARCHIVELOG mode, you can apply redo to restored backups in order to reconstruct all lost changes.

What Types of Failures Can Occur?

A backup is a safeguard against data loss or corruption. Unfortunately, you can lose or corrupt data in a variety of ways; you should consider these various possibilities when developing your backup strategy. The most common types of failures causing data loss are:

Statement failure	A logical failure in the handling of a statement in an Oracle program. For example, a user issues an invalid SQL statement. When statement failure occurs, Oracle automatically undoes any effects of the statement and returns control to the user.
Process failure	A failure in a user process accessing Oracle, e.g., an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. If the user process fails while modifying the database, Oracle background processes undo the effects of uncommitted transactions.
Instance failure	<p>A problem that prevents an Oracle instance, i.e., the SGA and background processes, from continuing to function. Instance failure can result from a hardware problem such as a power outage or a software problem such as an operating system crash. When an instance fails, Oracle does not write the data in the buffers of the SGA to the datafiles.</p> <p>In a single server environment, Oracle automatically performs <i>crash recovery</i> when the database opens at the next startup. In an Oracle Parallel Server (OPS) environment, another instance can perform <i>instance recovery</i> immediately.</p>
User or application error	<p>A user or application problem that results in the loss of data. For example, a user can accidentally delete data from a payroll table.</p> <p>User errors can require a database to be recovered to a point in time before the error occurred. To allow recovery from user errors and accommodate other unique recovery requirements, Oracle provides for <i>database point-in-time recovery</i> (DBPITR) or <i>tablespace point-in-time recovery</i> (TSPITR). For example, if a user accidentally deletes payroll data, you can recover the database to the point in time before the data was deleted.</p>

Media failure

A physical problem that arises when Oracle tries to write or read a file that is required to operate the database. A common example is a disk head crash that causes the loss of all data on a disk drive. Disk failure can affect a variety of files, including the datafiles, redo log files, and control files. Because the database instance cannot continue to function properly, it cannot write the data in the database buffers of the SGA to the datafiles.

What Should You Back Up?

Your database contains a wide variety of types of data. When developing your backup strategy, you must decide what information you want to copy. You have the following basic backup types:

- Whole database
- Tablespace
- Datafile
- Control file
- Archived redo log

The basic principle you should use when deciding what to back up is to prioritize data depending on its importance and the degree to which it changes. Archived redo logs do not change, for example, but they are crucial for recovering your database, so you should maintain multiple copies if possible. A different case is a group of important expense account tables that users constantly update. You should probably back up this tablespace frequently; in this way, you will not have to apply as much redo data during recovery.

You can combine types of backups in a variety of ways. For example, you may prudently decide to take weekly whole database backups, which ensures that you have a relatively current copy of your original database information. You can then take daily backups of your most-accessed tablespaces. You can also multiplex the all-important control file and online redo logs as an additional safeguard.

Which Backup Method Should You Use?

You have a choice between three basic methods for making backups. You can:

- Use the Recovery Manager (RMAN), which is a utility that establishes a connection with a server session and manages the movement for data for backup and recovery operations.

Note: RMAN is only compatible with Oracle databases of release 8.0 or higher. Use the Enterprise Backup Manager (EBU) for Oracle7 databases.

- Back up your database manually by executing commands specific to your operating system.
- Use the Oracle Export utility to make logical backups. The utility writes data from an Oracle database to operating system files in a proprietary format. You can later import this data into a database.

Note: Logical backups are not a substitute for whole database physical backups. You should consider logical backups as an additional tool within your overall backup and recovery strategy.

See Also: To learn more about choosing backup methods, see "[Choosing Backup Methods](#)" on page 3-8. For an overview of Recovery Manager features, see [Chapter 4, "Recovery Manager Concepts"](#).

Making Recovery Manager Backups and Image Copies

RMAN is a powerful and versatile program that allows you to make a *backup* or *image copy* of your data. When you specify files or archived logs using the RMAN **backup** command, RMAN creates a *backup set* as output.

A backup set is one or more datafiles, control files, or archived redo logs that are written in an RMAN-specific format; it requires you to use the RMAN **restore** command for recovery operations. In contrast, when you use the **copy** command to create an *image copy* of a file, it is in an instance-usable format—you do not need to invoke RMAN to restore or recover it.

When you issue RMAN commands such as **backup** or **copy**, RMAN establishes a connection to an Oracle server session. The server session then backs up the specified datafile, control file, or archived log from the target database.

RMAN obtains the information it needs from either the control file or the optional *recovery catalog*. The recovery catalog is a central repository containing a variety of information useful for backup and recovery. Conveniently, RMAN automatically establishes the names and locations of all the files that you need to back up.

RMAN provides several advantages. One crucial advantage to using RMAN is its *incremental backup* feature. In traditional backup methods, you must perform a *full backup* in which you back up all the data blocks ever used in a datafile. The incremental backup feature allows you to back up only those data blocks that have changed since a previous backup.

Using RMAN, you can perform two types of incremental backups: a *differential backup* or a *cumulative backup*. In a differential level n incremental backup, you back up all blocks that have changed since the most recent level n or lower backup. For example, in a differential level 2 backup, RMAN determines which level 1 or level 2 backup occurred most recently and backs up all blocks modified since that backup.

In a cumulative level n backup, RMAN backs up all the blocks used since the most recent backup at level $n-1$ or less. For example, in a cumulative level 3 backup, RMAN determines which level 2 or level 1 backup occurred most recently and backs up all blocks used since that backup.

Media Management You can easily integrate RMAN with a *media manager*. A media manager is a vendor-supplied software package that allows you to back up to archival media such as tape. RMAN coordinates with the media manager to move data between the disk and storage devices. If the media manager is capable of making a *proxy copy*, then it can perform the data transfer in backup and restore operations.

See Also: For an introduction to RMAN, see [Chapter 4, "Recovery Manager Concepts"](#). To learn how to make backups and copies with RMAN, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#). For information about incremental backups, see ["Incremental Backups"](#) on page 4-38.

Making O/S Backups

If you do not want to use RMAN, you can use operating system commands such as the UNIX `cp` command to make backups. You can also automate backup operations by writing scripts.

You can make a backup of the whole database at once or supplement a whole database backup with backups of individual tablespaces, datafiles, control files, and archived logs. You can use O/S commands to perform these backups.

See Also: To learn how to make O/S backups, see [Chapter 13, "Performing Operating System Backups"](#).

Using the Export Utility for Supplemental Backup Protection

You can supplement physical backups by using the Export utility to make logical backups of your data. Logical backups store information about the schema objects created for a database. The Export utility writes data from a database into Oracle files in a proprietary format. You can then import this data into a database using the Import utility.

See Also: To learn how to use the Export and Import utilities, see *Oracle8i Utilities*.

Should You Make Consistent or Inconsistent Backups?

You can use RMAN or O/S commands to make an *inconsistent backup* or a *consistent backup*. An inconsistent backup is a backup of one or more database files that you make while the database is open or after the database has been shut down abnormally. A consistent backup is a backup of one or more database files that you make after the database has been closed cleanly. Unlike an inconsistent backup, a consistent backup does not require instance recovery after it is restored.

Whether you make consistent or inconsistent backups depends on a number of factors. If your database must be open and available all the time, then inconsistent backups are your only option. If there are recurring periods of minimal use, then you may decide to take regular consistent backups of the whole database and supplement them with online backups of often-used tablespaces.

Consistent Backups

When you back up the entire database after shutting it down cleanly, it is called a consistent *whole database backup*. The basic procedure for a consistent whole database backup is:

1. Shut down the database normally.
2. Back up all datafiles, control files, and parameter files.
3. Re-start Oracle in normal mode.

If you are operating in NOARCHIVELOG mode, shut down the database cleanly before making a cold backup. If you do not, the database is inconsistent with respect to an SCN and requires instance recovery to become consistent. The backup may be rendered unusable.

Inconsistent Backups

You can make backups of tablespaces, datafiles, control files, and archived redo logs while the database is open. If possible, take the desired tablespace offline and back up the datafiles. If the tablespace must remain online, and you are using O/S methods to back up the datafiles, first place the tablespace in *hot backup mode* by issuing the ALTER TABLESPACE ... BEGIN BACKUP command. Take it out of hot backup mode by issuing ALTER TABLESPACE ... END BACKUP. You do not need to issue this command when using RMAN.

See Also: To learn how to make online backups using O/S methods, see ["Backing Up Online Tablespaces and Datafiles"](#) on page 13-6. To learn how to use RMAN to make backups, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#).

How Often Should You Make Backups?

Tailor your backup strategy to the needs of your business. For example, if you can afford to lose data in the event of a disk failure, you may not need to perform backups very often. The advantage of taking infrequent backups is that you free Oracle's resources for other operations. The disadvantage is that you may end up losing data or increasing recovery time.

In a different scenario, if your database must be available twenty-four hours a day, seven days a week, then you should make online backups of your database frequently. In this case, you may decide to take daily hot backups, *multiplex* (i.e., have multiple copies of) your online redo logs, and archive your redo logs to several different locations. You can even maintain a standby database in a different city that is a constantly updated replica of your original database.

See Also: To learn important considerations for an effective backup strategy, see [Chapter 3, "Developing a Backup and Recovery Strategy"](#).

Understanding Basic Recovery Strategy

As we have seen, basic media recovery involves two parts: restoring a physical backup and updating it with database changes. The most important aspect of recovery is making sure that all datafiles are consistent with respect to the same SCN. Oracle has integrity checks that prevent you from opening the database until all datafiles are consistent with one another.

When preparing a recovery strategy, make sure you understand the answers to these questions:

- [How Does Recovery Work?](#)
- [What Are the Types of Recovery?](#)
- [What Is Media Recovery?](#)
- [Which Recovery Method Should You Use?](#)

How Does Recovery Work?

In every type of instance or media recovery (except media recovery using RMAN incremental backups), Oracle sequentially applies redo data to data blocks. Oracle uses information in the control file and datafile headers to ascertain whether recovery is necessary.

Recovery has two parts: *rolling forward* and *rolling back*. When Oracle rolls forward, it applies redo records to the corresponding data blocks. Oracle systematically goes through the redo log to determine which changes it needs to apply to which blocks, and then changes the blocks. For example, if a user adds a row to a table, but the server crashes before it can save the change to disk, Oracle can use the redo record for this transaction to update the data block to reflect the new row.

During the rolling back phase, Oracle applies rollback segments to the datafiles. The rollback information is stored in *transaction tables*. Oracle searches through the table for uncommitted transactions, undoing any that it finds. For example, if the user never committed the SQL statement that added the row, then Oracle will discover this fact in a transaction table and undo the change.

If the database is mounted during recovery, rollback occurs only when the database is opened. If the database is open and a tablespace is offline during recovery, rollback for that tablespace occurs when the tablespaces is brought online.

See Also: To learn more about the mechanics of Oracle recovery, see *Oracle8i Concepts*.

What Are the Types of Recovery?

There are three basic types of recovery: *instance recovery*, *crash recovery*, and *media recovery*. Oracle performs the first two types of recovery automatically at instance startup; only media recovery requires you to issue commands.

Instance Recovery

Instance recovery, which is only possible in an OPS configuration, occurs in an open database when one instance discovers that another instance has crashed. A surviving instance automatically uses the redo log to recover the committed data in the database buffers that was lost when the instance failed. Further, Oracle undoes any transactions that were in progress on the failed instance when it crashed and then clears any locks held by the crashed instance after recovery is complete.

Crash Recovery

Crash recovery occurs when either a single-instance database crashes or all instances of a multi-instance database crash. In crash recovery, an instance must first open the database and then execute recovery operations. In general, the first instance to open the database after a crash or SHUTDOWN ABORT automatically performs crash recovery.

Media Recovery

Unlike crash and instance recovery, media recovery is executed on your command. In media recovery, you use online and archived redo logs and (if using RMAN) incremental backups to make a restored backup current or to update it to a specific time. It is called media recovery because you usually perform it in response to media failure.

What Is Media Recovery?

Media recovery uses redo records or (if you use RMAN) incremental backups to recover restored datafiles either to the present or to a specified non-current time. When performing media recovery, you can recover the whole database, a tablespace, or a datafile. In any case, you always use a restored backup to perform the recovery. The principal division in media recovery is between *complete recovery* and *incomplete recovery*.

Complete Recovery

Complete recovery involves using redo data or incremental backups combined with a backup of a database, tablespace, or datafile to update it to the most current point in time. It is called *complete* because Oracle applies *all* of the redo changes to the backup. Typically, you perform media recovery after a media failure damages datafiles or the control file.

Requirements for Complete Recovery You can perform complete recovery on a database, tablespace, or datafile. If you are performing complete recovery on the whole database, then you must:

- Mount the database.
- Ensure that all datafiles you want to recover are online.
- Restore a backup of the whole database or the files you want to recover.
- Apply online or archived redo logs, or a combination of the two.

If you are performing complete recovery on a tablespace or datafile, then you must:

- Take the tablespace or datafile to be recovered offline if the database is open.
- Restore a backup of the datafiles you want to recover.
- Apply online or archived redo logs, or a combination of the two.

See Also: To learn how to perform complete recovery with RMAN, see "[Performing Complete Recovery](#)" on page 9-18. To learn how to perform complete media recovery using O/S methods, see "[Performing Complete Media Recovery](#)" on page 14-20.

Incomplete Recovery

Incomplete recovery uses a backup to produce a non-current version of the database. In other words, you do not apply all of the redo data generated since the most recent backup. You usually perform incomplete recovery when:

- Media failure destroys some or all of the online redo logs.
- A user error causes data loss, e.g., a user inadvertently drops a table.
- You cannot perform complete recovery because an archived redo log is missing.
- You lose your current control file and must use a backup control file to open the database.

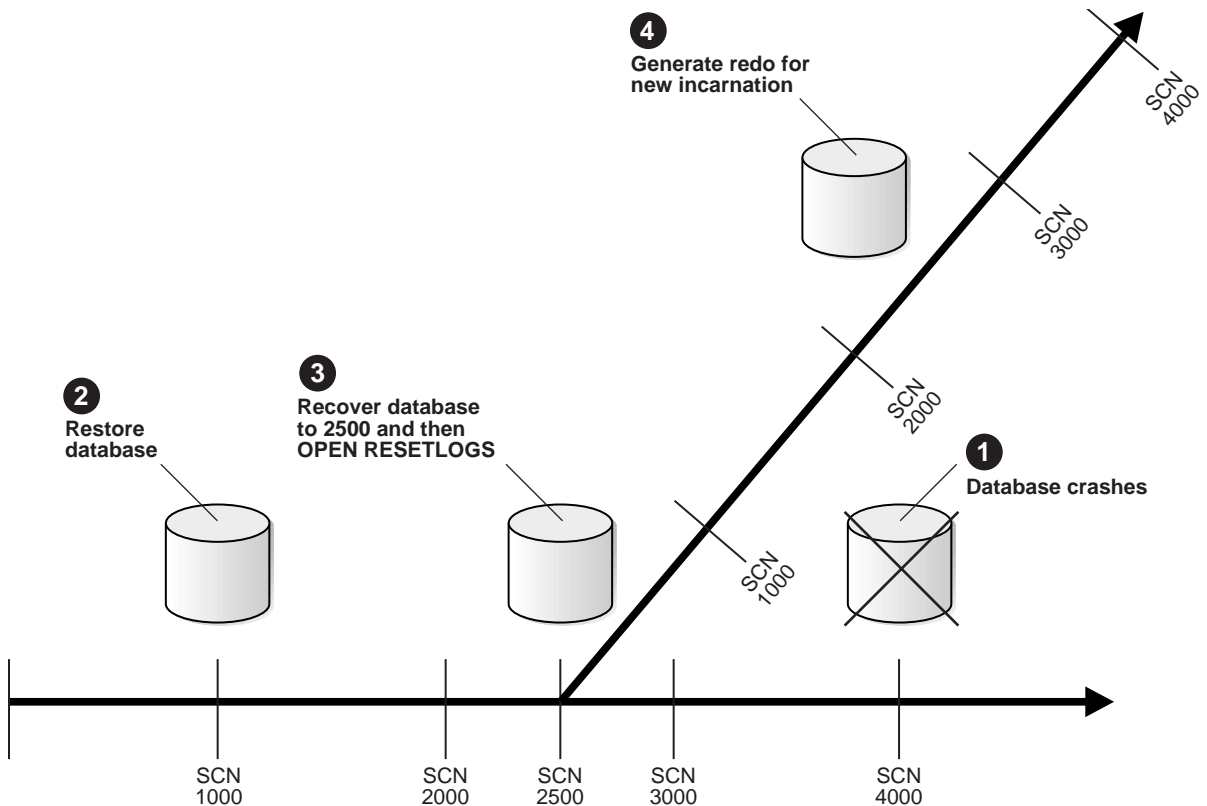
To perform incomplete media recovery, you must restore all datafiles from backups created prior to the time to which you want to recover and then open the database with the RESETLOGS option when recovery completes. The RESETLOGS operation creates a new incarnation of the database. All archived redo logs generated after the point of the RESETLOGS on the old incarnation are invalid on the new incarnation.

[Figure 1-2](#) shows the case of a database that can only be recovered to SCN 2500 because an archived redo log is missing. At SCN 4000, the database crashes. You restore the SCN 1000 backup and prepare for complete recovery. Unfortunately, one

of your archived redo logs is corrupted. The log before the missing log contains SCN 2500, so you recover to this point and open with the RESETLOGS option.

As the diagram illustrates, you generate new changes in the new incarnation of the database, eventually reaching SCN 4000. The changes between SCN 2500 and SCN 4000 for the *new* incarnation of the database will be completely different from the changes between SCN 2500 and SCN 4000 for the *old* incarnation. Oracle will not allow you to apply logs from an old incarnation to the new incarnation. You cannot restore backups from before SCN 2500 in the old incarnation to the new incarnation.

Figure 1–2 Creating a New Database Incarnation



Tablespace Point-in-Time Recovery The *tablespace point-in-time recovery (TSPITR)* feature enables you to recover one or more tablespaces to a point-in-time that is different from the rest of the database. TSPITR is most useful when you want to:

- Recover from an erroneous drop or truncate table operation.
- Recover a table that has become logically corrupted.
- Recover from an incorrect batch job or other DML statement that has affected only a subset of the database.
- Recover one logical database to a point different from the rest of a physical database (in cases where there are multiple logical databases in separate tablespaces of one physical database).
- Recover a tablespace on a very large database (VLDB) rather than restore the whole database from a backup and perform a complete database roll-forward (see "[Planning for Tablespace Point-in-Time Recovery](#)" on page B-4 before making any decisions).

See Also: To learn how to perform TSPITR using RMAN, see [Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager"](#). To learn how to perform O/S TSPITR, see [Appendix B, "Performing Operating System Tablespace Point-in-Time Recovery"](#).

Recovery Options Since you are not completely recovering the database to the most current time, you must tell Oracle when to terminate recovery. You can perform the following types of recovery.

Type of Recovery	Function
time-based recovery	Recovers the data up to a specified point in time.
cancel-based recovery	Recovers until you issue the CANCEL command.
change-based recovery	Recovers until the specified SCN.
log sequence recovery	Recovers until the specified log sequence number.

See Also: To learn how to perform incomplete recovery with RMAN, see "[Performing Incomplete Recovery](#)" on page 9-23. To learn how to perform incomplete media recovery using O/S methods, see "[Performing Incomplete Media Recovery](#)" on page 14-26.

Opening the Database with the RESETLOGS Option Whenever you perform incomplete recovery, you must reset the online redo logs when you open the database. The new version of the reset database is called a new *incarnation*.

Opening the database with the RESETLOGS option informs Oracle that you want to discard some redo and prevents Oracle from ever applying the discarded redo in any recovery you might do in the future. For example, if the latest log sequence number for your database is 100, and you then recover to log sequence number 50 and do *not* open RESETLOGS, eventually your database will reach log sequence number 100 again. If you then try to recover the database using the *old* archived log 100, you run the risk of corrupting your datafiles or generating internal errors.

Whenever you open the database with the RESETLOGS option, all datafiles get a new SCN and timestamp. Archived redo logs also have these two values in their header. Because Oracle will not apply an archived redo log to a datafile unless the SCN and timestamps match, the RESETLOGS operations prevents you from corrupting your datafiles with old archived logs.

After resetting the online redo logs, make a whole database backup. In general, backups made before a RESETLOGS operation are not legal in the new incarnation. There is, however, an exception to the rule: you can restore a pre-RESETLOGS backup *only if* Oracle does not need to access archived redo logs from before the RESETLOGS to perform recovery.

It is possible to restore these pre-RESETLOGS backups in a new incarnation:

- Backups of a tablespace made after it was made read-only (only if it was not made read-write again before the RESETLOGS)
- Backups of a tablespace after it was taken offline-normal (only if it was not brought online again before the RESETLOGS)
- Backups of a read-write tablespace made after recovery ends and before you open RESETLOGS, i.e., you do not perform further recovery or alter the datafiles between the backup and the RESETLOGS

Note that you are prevented from restoring backups of read-write tablespaces that were *not* made immediately before the RESETLOGS. This restriction obtains even if no changes were made to the datafiles in the read-write tablespace between the backup and the RESETLOGS. Because the checkpoint in the datafile header of a backup will be older than the checkpoint in the control file, Oracle has to search the archived logs to determine whether changes need to be applied—and the pre-RESETLOGS archived logs are not valid in the new incarnation.

See Also: To learn how to restart the database in RESETLOGS mode, see "[Opening the Database after Media Recovery](#)" on page 14-33.

Which Recovery Method Should You Use?

You have a choice between two basic methods for recovering physical files. You can:

- Use the RMAN utility to automate recovery.
- Recover your database manually by executing SQL/SQL*Plus commands.

Recovering with RMAN

The basic RMAN recovery commands are **restore** and **recover**. Use RMAN to restore datafiles from backup sets or from image copies on disk, either to their current location or to a new location. You can also restore backup sets containing archived redo logs.

If you use a recovery catalog, RMAN has a record containing all the essential metadata concerning every backup you have taken. If you do not use a recovery catalog, RMAN uses the control file for necessary metadata.

Use the RMAN **recover** command to perform media recovery and apply incremental backups. You can use the **set until** command to perform incomplete media recovery. RMAN completely automates the procedure for recovering and restoring your backups and copies.

See Also: To learn how to restore database files, see ["Restoring Datafiles, Control Files, and Archived Redo Logs"](#) on page 9-2. To learn how to perform recovery with RMAN, see ["Recovering Datafiles"](#) on page 9-15. For RMAN syntax, see [Chapter 11, "Recovery Manager Command Syntax"](#).

Recovering with SQL*Plus

If you do not use RMAN, use the SQL*Plus utility to restore and recover your files. You can execute:

- The SQL*Plus RECOVER command (recommended)
- The SQL ALTER DATABASE RECOVER statement

If your operating system supports Oracle Enterprise Manager, you can execute restore and recovery operations in a GUI environment. See the *Oracle Enterprise Manager Administrator's Guide* for more information.

In each case you can recover a database, tablespace, or datafile. Before performing recovery, you need to:

1. Determine which datafiles to recover. Often you can use the fixed view `VS$RECOVER_FILE`.

2. Restore backups of files permanently damaged by media failure. If you do not have a backup, it is sometimes possible to perform recovery if you have the necessary redo logs dating from the time when the datafiles were first created and the control file contains the name of the damaged file.

If you cannot restore a datafile to its original location, relocate the restored datafile and inform the control file of the new location.

3. Restore any necessary archived redo log files.

After these steps are completed, issue either the RMAN **recover** command or the SQL*Plus RECOVER statement.

See Also: To learn how to perform recovery with RMAN, see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#). To learn how to perform operating system recovery, see [Chapter 14, "Performing Operating System Recovery"](#).

To learn how to use the SQL*Plus utility, see the *SQL*Plus User's Guide and Reference*. For more information on the syntax of SQL commands, see the *Oracle8i SQL Reference*.

Managing Data Structures

This chapter describes how to manage data structures that are crucial for successful backup and recovery. It includes the following topics:

- [Overview of Backup and Recovery Data Structures](#)
- [Managing the Control File](#)
- [Managing the Online Redo Log](#)
- [Managing the Archived Redo Logs](#)

See Also: For a conceptual overview of these data structures, see *Oracle8i Concepts*. For detailed administration information, see the *Oracle8i Administrator's Guide*. If you are using Oracle with the Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

Overview of Backup and Recovery Data Structures

The single most useful strategy in backup and recovery is planning ahead. To prevent data loss, you must foresee the various ways that data can be lost and develop your defense accordingly.

An important aspect of planning ahead is the intelligent management of database data structures. For example, what can you do to prevent a database crash if your control file becomes corrupted? What can you do to prevent the loss of archived redo logs if a disk failure occurs? Besides the datafiles, the data structures that are most important for developing a backup and recovery strategy are:

- Control files
- Online redo logs
- Archived redo logs (if you run in ARCHIVELOG mode)

If these structures become corrupted or unavailable, you may find yourself unable to recover lost data.

If you have sufficient resources, you can help protect yourself from data loss by following this basic data management strategy:

- Mirror your datafiles at the O/S or hardware level.
- Maintain at least two current control files on different disks by multiplexing or mirroring them. Use O/S or hardware mirroring to ensure that you can recover from any one media failure while the system is fully available.
- Maintain at least three online redo log groups with two members each. Place each member of the group on a different disk and on a different controller.
- Archive your redo logs to multiple destinations and back them up frequently to different media. It is advisable to take multiple backups to multiple different media devices.

Note: This chapter assumes that you understand the function of the control file, online redo logs, and archived redo logs as well as the basics of how to administer them. If you do not, refer to the relevant chapters in *Oracle8i Concepts* and the *Oracle8i Administrator's Guide*.

Managing the Control File

The control file is a small binary file containing a record of the database schema. It is one of the most essential files in the database because it is necessary for the database to start and operate successfully. Oracle updates a control file continuously during database use, so it must be available for writing whenever the database is mounted. If for some reason the control file is not accessible, then the database cannot be mounted and recovery is difficult.

A control file contains information about the associated database that is required for the database to be accessed by an instance, both at startup and during normal operation. Only the Oracle server can modify a control file's information; no user can edit a database's control file.

The control file has various properties that make it crucial for backup and recovery. For example, the control file:

- Identifies the database name (taken from either the name specified by the initialization parameter `DB_NAME` or the name used in the `CREATE DATABASE` statement).
- Records the names and locations of associated datafiles and online redo log files.
- Records the names and locations of archived redo logs.
- Stores checkpoint and log sequence information required for the synchronization of the database.
- Stores information on Recovery Manager backups (if you use Recovery Manager). The recovery catalog optionally used by RMAN obtains its essential information from the control file.
- Must be accessible for the database to be mounted, opened, and maintained.

This section addresses the following topics relating to control file management:

- [Displaying Control File Information](#)
- [Backing Up the Control File After Structural Changes](#)
- [Maintaining Multiple Control Files](#)
- [Recovering from the Loss of Control Files](#)

Displaying Control File Information

Your first step in managing the control file is learning how to gain information about it. The following data dictionary views contain useful information:

Views	Description
V\$CONTROLFILE	Lists the control file filenames.
V\$DATABASE	Indicates whether the control file is current or a backup, when the control file was created, and the last timestamp in the control file if it is a backup

For example, the following query displays the database control files:

```
SELECT name FROM v$controlfile;
```

```
NAME
```

```
-----
```

```
/vobs/oracle/dbs/cf1.f
```

```
/vobs/oracle/dbs/cf2.f
```

```
2 rows selected.
```

To display the control file type, query the V\$DATABASE view:

```
SELECT controlfile_type FROM v$database;
```

```
CONTROL
```

```
-----
```

```
BACKUP
```

The following useful command displays all control files, datafiles, and online redo log files for the database:

```
SELECT member FROM v$logfile
```

```
UNION ALL
```

```
SELECT name FROM v$datafile
```

```
UNION ALL
```

```
SELECT name FROM v$controlfile;
```

```
MEMBER
```

```
-----
```

```
/vobs/oracle/dbs/rdo_log1.f
```

```
/vobs/oracle/dbs/rdo_log2.f
```

```
/vobs/oracle/dbs/tbs_01.f
```

```
/vobs/oracle/dbs/tbs_02.f
```

```
/vobs/oracle/dbs/tbs_11.f
```

```
/vobs/oracle/dbs/tbs_12.f
```

```
/vobs/oracle/dbs/tbs_21.f
```

```
/vobs/oracle/dbs/tbs_22.f  
/vobs/oracle/dbs/tbs_13.f  
/vobs/oracle/dbs/cf1.f  
/vobs/oracle/dbs/cf2.f  
11 rows selected.
```

See Also: For more information on the dynamic performance views, see the *Oracle8i Reference*.

Backing Up the Control File After Structural Changes

Each time that a user adds, renames, or drops a datafile or an online redo log file from the database, Oracle updates the control file to reflect this physical structure change. Oracle records these changes so that it can identify:

- The datafiles and online redo log files that it needs to open during database startup.
- The files that are required or available in case database recovery is necessary.

Therefore, if you make a change to your database's physical structure, *immediately* back up your control file. If you do not, and your control file is corrupted or destroyed, then your backup control file will not accurately reflect the state of the database at the time of the failure.

Generate a binary copy of the control file or back up to a text trace file (with the destination specified by the USER_DUMP_DEST initialization parameter). You can run the script in the text trace file to re-create the control file. Back up the control file after you issue any of the following commands:

- ALTER DATABASE [ADD | DROP] LOGFILE
- ALTER DATABASE [ADD | DROP] LOGFILE MEMBER
- ALTER DATABASE [ADD | DROP] LOGFILE GROUP
- ALTER DATABASE [ARCHIVELOG | NOARCHIVELOG]
- ALTER DATABASE RENAME FILE
- CREATE TABLESPACE
- ALTER TABLESPACE [ADD | RENAME] DATAFILE
- ALTER TABLESPACE [READ WRITE | READ ONLY]
- DROP TABLESPACE

To create a binary backup of the control file using a SQL command:

1. If the database is not mounted or open, use SQL*Plus to mount or open it:

```
SQL> ALTER DATABASE MOUNT;
```

2. Issue the following ALTER DATABASE statement, specifying the backup control file destination:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/cf.f';
```

To back up the control file to a text trace file:

You can back up the control file to a trace file and then use the script in this file to re-create the control file.

1. If the database is not mounted or open, use SQL*Plus to mount or open it:

```
SQL> ALTER DATABASE MOUNT;
```

2. Issue the following command:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

3. Run the script in the trace file located in the USER_DUMP_DEST directory. Depending on whether you have offline or read-only tablespaces, you may need to edit the script first.

For example, to edit trace `rman_ora_839.trc` on UNIX enter:

```
% vi rman_ora_839.trc

*** SESSION ID:(8.1) 1998.12.09.13.26.36.000
*** 1998.12.09.13.26.36.000
# The following commands will create a new control file and use it
# to open the database.
# Data used by the recovery manager will be lost. Additional logs may
# be required for media recovery of offline data files. Use this
# only if the current version of all online logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "RMAN" NORESETLOGS ARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 1012
LOGFILE
    GROUP 1 '/oracle/dbs/t1_log1.f' SIZE 200K,
    GROUP 2 '/oracle/dbs/t1_log2.f' SIZE 200K
DATAFILE
```

```

'/oracle/dbs/tbs_01.f',
'/oracle/dbs/tbs_02.f',
'/oracle/dbs/tbs_11.f',
'/oracle/dbs/tbs_12.f',
'/oracle/dbs/tbs_21.f',
'/oracle/dbs/tbs_22.f',
CHARACTER SET WE8DEC
;
# Configure snapshot controlfile filename
EXECUTE SYS.DBMS_BACKUP_RESTORE.CFILESETSNAPSHOTNAME('/oracle/dbs/snapcf_rman.f');
# Recovery is required if any of the datafiles are restored backups,
# or if the last shutdown was not normal or immediate.
RECOVER DATABASE
# All logs need archiving and a log switch is needed.
ALTER SYSTEM ARCHIVE LOG ALL;
# Database can now be opened normally.
ALTER DATABASE OPEN;
# No tempfile entries found to add.

```

See Also: For more information on managing the control file, see the *Oracle8i Administrator's Guide*. For a sample scenario involving editing a trace file, see ["Backing Up the Control File to a Trace File"](#) on page 13-12.

Maintaining Multiple Control Files

As with online redo log files, Oracle allows you to *multiplex* control files, i.e., configure Oracle to open and write to multiple, identical copies. Oracle writes the same data to each copy of the control file. You can also *mirror* them, i.e., allow the O/S to write a copy of a control file to two or more physical disks.

Mirroring at the O/S level is often better than multiplexing at the Oracle level, since O/S mirroring usually tolerates failure of one of the mirrors, whereas Oracle does not. With Oracle multiplexing, if any one of the mirror sides fail, then the instance shuts down. The user then has to either:

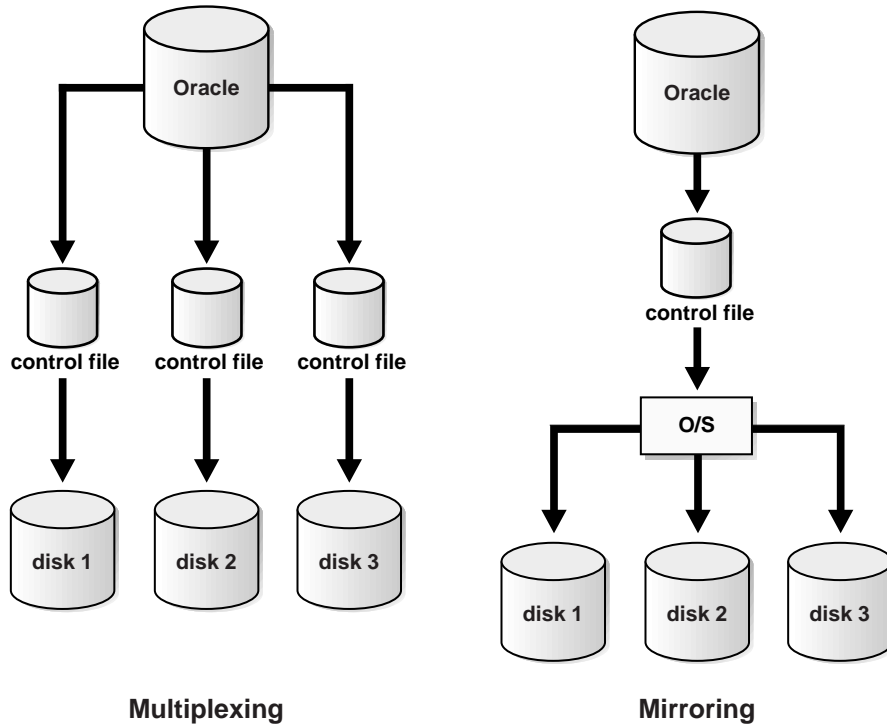
- Repair the failed disk, copy a good control file into the old location, and restart.
- Copy a good control file to a new location and edit the `CONTROL_FILES` parameter of the `init.ora` file accordingly.

With O/S or hardware mirroring, you achieve the same redundancy that you do with multiplexing, but in many cases you do not have to pay with a loss in availability when a failure occurs.

The permanent loss of all copies of a database's control file is a serious problem. If any copy of a control file fails during database operation, then the instance aborts and media recovery is required. If you do not multiplex or mirror the control file,

then recovery will be more complex. Therefore, you should use multiplexed or mirrored control files with each database.

Figure 2-1 Multiplexing and Mirroring the Control File



Multiplexing the Control File

By storing multiple control files for a single database on different disks, you safeguard against a single point of failure. If a single disk containing a control file crashes, the instance fails when Oracle attempts to access the damaged control file.

If the control file is multiplexed, other copies of the current control file are available on different disks. After repairing the bad disk, you can then copy a good control file to the old location and restart the instance easily without having to perform media recovery. If you cannot repair the disk, then you can edit the `CONTROL_FILES` initialization parameter to specify a new location and copy the good control file to this location.

The only disadvantage of multiplexing control files is that operations that update the control files (such as adding a datafile or checkpointing the database) can take slightly longer. This increase in performance overhead is usually insignificant, however, especially for operating systems that can perform multiple, concurrent writes. A slight performance loss does not justify using only a single control file.

Note: Oracle strongly recommends that you maintain a minimum of two control files on different disks.

Note the following characteristics of multiplexed control files:

- At least two filenames are listed for the initialization parameter `CONTROL_FILES` in the database's parameter file.
- The first file listed in the `CONTROL_FILES` parameter is the only file read by the Oracle Server during database operation.
- If any of the control files becomes unavailable during database operation, the instance becomes inoperable and should be aborted. Copy a good control file to the bad control file's location or, if media failure makes the disk inaccessible, copy a good control file to a new location and edit the `init.ora` file.

Mirroring the Control File

If your operating system supports disk mirroring, then the O/S allows for *mirrored disk storage*. Mirrored disk storage makes several physical disks look like a single disk to Oracle. Oracle writes the data once, then the O/S writes it to each of the underlying physical disks. Each file is a *mirror*, i.e., an exact duplicate, of the others.

The advantage of disk mirroring is that if one of the disks becomes unavailable, then the other disk or disks can continue to function without interruption. Therefore, your control file is protected against a single point of failure. Note that if you store your control file on a mirrored disk system, then you only need Oracle to write one active copy of the control file.

Recovering from the Loss of Control Files

Following are scenarios where you may need to recover or re-create the control file:

- The control file is corrupted or lost due to media failure. If you multiplex or mirror your control files on different disks, you significantly minimize this risk.

- You change the name of a database. Because the control file records the name of the database, you will need to re-create it.

To recover from control file corruption using a current control file copy:

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is corrupted, the control file directory is still accessible, and you have a current multiplexed or mirrored copy.

1. With the instance shut down, use an O/S command to overwrite the bad control file with a good copy:

```
% cp '/disk2/copy/cf.f' '/disk1/oracle/dbs/cf.f';
```

2. Start SQL*Plus and mount or open the database:

```
SQL> STARTUP MOUNT;
```

To recover from permanent media failure using a current control file copy:

This procedure assumes that one of the control files specified in the CONTROL_FILES parameter is inaccessible due to a permanent media failure, and you have a current multiplexed or mirrored copy.

1. With the instance shut down, use an O/S command to copy the current copy of the control file to a new, accessible location:

```
% cp '/disk2/copy/cf.f' '/disk3/copy/cf.f';
```

2. Edit the CONTROL_FILES parameter in the `init.ora` file to replace the bad location with the new location:

```
CONTROL_FILES = '/oracle/dbs/cf1.f','/disk3/copy/cf.f'
```

3. Start SQL*Plus and mount or open the database:

```
SQL> STARTUP MOUNT;
```


Managing the Online Redo Log

Perhaps the most crucial structure for recovery operations is the online *redo log*, which consists of two or more pre-allocated files that store all changes made to the database as they occur. Every instance of an Oracle database has an associated online redo log to protect the database in case of an instance failure.

WARNING: Oracle recommends that you do not back up a current online log, because if you restore that backup, the backup will appear at the end of the redo thread. Since additional redo may have been generated in the thread, when you attempt to execute recovery by supplying the redo log copy, recovery will erroneously detect the end of the redo thread and prematurely terminate, possibly corrupting the database.

Each database instance has its own online *redo log groups*. These online redo log groups, multiplexed or not, are called an instance's *thread* of online redo. In typical configurations, only one database instance accesses an Oracle database, so only one thread is present. When running the Oracle Parallel Server, however, two or more instances concurrently access a single database; each instance has its own thread.

Note: This manual describes how to configure and manage the online redo log when the Oracle Parallel Server is *not* used. Thus, the thread number can be assumed to be 1 in all discussions and examples of commands. For complete information about configuring the online redo log with the Oracle Parallel Server, see *Oracle8i Parallel Server Concepts and Administration*.

See Also: For a conceptual overview of the online redo log, see *Oracle8i Concepts*. For detailed information about managing the online redo logs, see the *Oracle8i Administrator's Guide*.

Displaying Online Redo Log Information

The following data dictionary views contain useful information about the archived redo logs:

Views	Description
V\$LOG	Identifies the online redo log groups, the number of members per group, and which logs have been archived.
V\$LOGFILE	Displays filenames and status information about the redo log group members.

For example, the following query displays which online redo log group requires archiving:

```
SELECT group#, sequence#, status, archived FROM v$log;
```

GROUP#	SEQUENCE#	STATUS	ARC
1	43	CURRENT	NO
2	42	INACTIVE	YES

2 rows selected.

To display the members for each log group, query the V\$LOGFILE view:

```
SELECT group#, member FROM v$logfile;
```

GROUP#	MEMBER
1	/oracle/dbs/t1_log1.f
2	/oracle/dbs/t1_log2.f

2 rows selected.

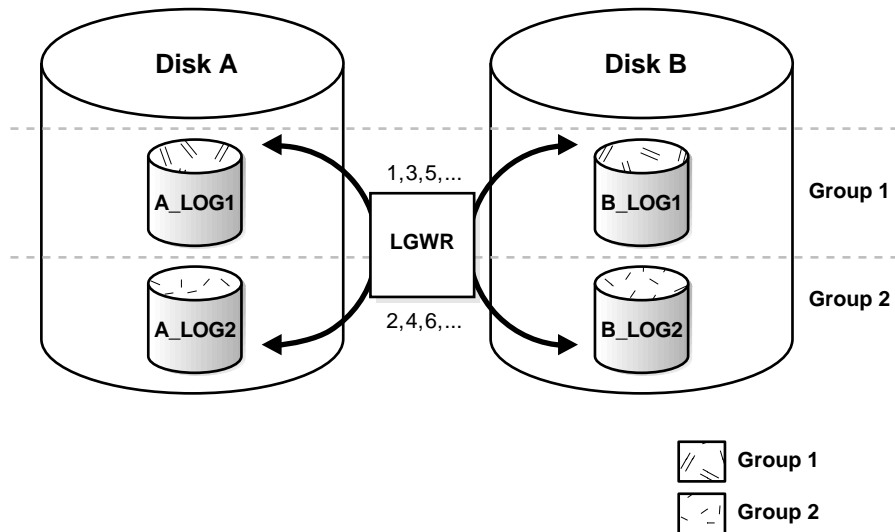
See Also: For more information on the data dictionary views, see the *Oracle8i Reference*.

Multiplexing Online Redo Log Files

Oracle provides the capability to *multiplex* an instance's online redo log files to safeguard against damage. When **multiplexing** online redo log files, LGWR concurrently writes the same information to multiple identical online redo log files, thereby eliminating a single point failure. You can also mirror your redo logs at the O/S level, but in so doing you run the risk of O/S or hardware induced corruption. In most cases, multiplexing of online logs is best.

WARNING: Oracle strongly recommends that you multiplex your redo log files or mirror them at the O/S level; the loss of the redo data can be catastrophic if recovery is required.

Figure 2–2 Multiplexed Online Redo Log Files



The corresponding online redo log files are called *groups*. Each online redo log file in a group is called a *member*. In [Figure 2–2](#), files 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 exact same size.

Notice that each member of a group is concurrently active, i.e., concurrently written to by LGWR, as indicated by the identical log sequence numbers assigned by LGWR. In [Figure 2–2](#), first LGWR writes to file A_LOG1 in conjunction with B_LOG1, then A_LOG2 in conjunction with B_LOG2, etc. LGWR never writes concurrently to members of different groups, e.g., to A_LOG1 and B_LOG2.

Responding to Online Redo Log Failure

Whenever LGWR cannot write to a member of a group, Oracle marks that member as stale and writes an error message to the LGWR trace file and to the database's alert log to indicate the problem with the inaccessible files. LGWR reacts differently

when certain online redo log members are unavailable, depending on the reason for the unavailability.

If	Then
LGWR can successfully write to at least one member in a group	Writing proceeds as normal; LGWR simply writes to the available members of a group and ignores the unavailable members.
LGWR cannot access the next group at a log switch because the group needs to be archived	Database operation temporarily halts until the group becomes available, i.e., until the group is archived.
All members of the next group are inaccessible to LGWR at a log switch because of disk failures	Oracle returns an error and the database instance shuts down. In this case, you may need to perform media recovery on the database from the loss of an online redo log file.
All members of the next group are inaccessible and the database checkpoint has moved beyond the lost redo log	Media recovery is not necessary because Oracle has saved the data recorded in the redo log to the datafiles. Simply drop the inaccessible redo log group.
You want to drop an unarchived redo log when in ARCHIVELOG mode	Issue ALTER DATABASE CLEAR UNARCHIVED LOG to disable archiving before the log can be dropped.
All members of group become inaccessible to LGWR while it is writing to them	Oracle returns an error and the database instance immediately shuts down. In this case, you may need to perform media recovery. If the media containing the log is not actually lost — for example, if the drive for the log was inadvertently turned off — media recovery may not be needed. In this case, you only need to turn the drive back on and let Oracle perform instance recovery.

See Also: For more information about configuring multiplexed online redo logs, see the *Oracle8i Administrator's Guide*.

Managing the Archived Redo Logs

If you run your database in ARCHIVELOG mode, Oracle allows you to save filled groups of online redo log files, known as *archived redo logs*, to one or more offline destinations. *Archiving* is the operation of turning online redo logs into archived redo logs.

Use archived logs to:

- Recover a database.
- Update a standby database.
- Gain information about the history of a database via the LogMiner utility.

An archived redo log file is a copy of one of the identical filled members of an online redo log group: it includes the redo entries present in the identical members of a group and also preserves the group's unique log sequence number. For example, if you are multiplexing your online redo logs, and if Group 1 contains member files A_LOG1 and B_LOG1, then the ARC*n* process will archive one of these identical members. Should A_LOG1 become corrupted, then ARC*n* can still archive the identical B_LOG1.

If you enable archiving, LGWR is not allowed to re-use and hence overwrite an online redo log group until it has been archived. Therefore, the archived redo log contains a copy of every online redo group created since you enabled archiving. The best way to back up the contents of the current online log is always to archive it, then back up the archived log.

By archiving your online redo logs, you save a copy of every change made to the database since you enabled archiving. If you suffer a media failure, you can recover the lost data by using the archived redo logs.

See Also: For complete procedures for managing archived redo logs as well as for using the LogMiner, see the *Oracle8i Administrator's Guide*. To learn how to manage a standby database see [Chapter 16, "Managing a Standby Database"](#).

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 includes the following topics:

- [Running a Database in NOARCHIVELOG Mode](#)
- [Running a Database in ARCHIVELOG Mode](#)

Running a Database in NOARCHIVELOG Mode

When you run your database in NOARCHIVELOG mode, you disable the archiving of the online redo log. The database's control file indicates that filled groups are not required to be archived. Therefore, after a filled group becomes inactive after a log switch, the group is available for reuse by LGWR.

The choice of whether to enable the archiving of filled groups of online redo log files depends on the availability and reliability requirements of the application running on the database. If you cannot afford to lose any data in your database in the event of a disk failure or operator error, use ARCHIVELOG mode. Note that the archiving of filled online redo log files can require you to perform extra administrative operations.

When you do not archive your logs, you lose all redo contained in a log when Oracle switches into it and begins writing. Running your database in NOARCHIVELOG mode has the following consequences:

- You can only *restore* (not recover) the database to the point of the most recent full database backup. You cannot apply archived logs to the backup because there are no archived logs to be applied.
- You can only perform an operating system backup of the database when it is shut down cleanly.
- You can only restore a whole database backup and then open the database when the backup was taken while the database was closed cleanly. The only time you can restore an inconsistent backup is when the redo logs are undamaged and contain all redo generated since the backup was made.
- You cannot perform online tablespace backups. Furthermore, you cannot use online tablespace backups previously taken while the database operated in ARCHIVELOG mode.

Therefore, if you decide to operate a database in NOARCHIVELOG mode, take full database backups at regular, frequent intervals. Otherwise, you may end up in the situation of having to restore an old backup and lose days, weeks, or even months worth of changes.

Running a Database in ARCHIVELOG Mode

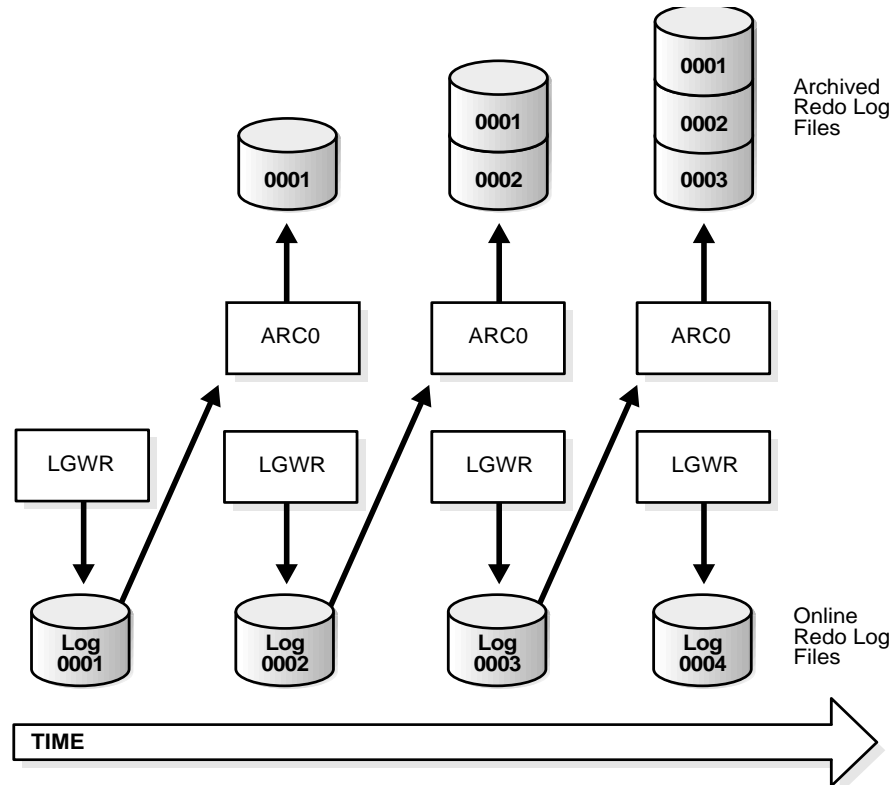
When you run a database in ARCHIVELOG mode, Oracle requires the online redo log to be archived. You can either perform the archiving manually or enable automatic archiving.

In ARCHIVELOG mode, the database control file indicates that a group of filled online redo log files cannot be used by LGWR until the group is archived. A filled group is immediately available to ARC*n* after a log switch occurs. After the group has been successfully archived, Oracle can reuse the group.

The archiving of filled groups has these advantages:

- A database backup in conjunction with backups of online and archived logs guarantees that you can recover all committed transactions.
- You can recover the database to a specified time, SCN, or log sequence number.
- You can take tablespace backups while the database is open.
- You can recover tablespaces while the database is open.
- You can keep a standby database current with its original database by continually applying the original's archived redo logs to the standby.

Figure 2-3 Online Redo Log File Use in ARCHIVELOG Mode



See Also: To learn how to enable ARCHIVELOG mode and enable automatic archiving, see the *Oracle8i Administrator's Guide*. For information about the managed recovery option for standby databases, see "[Maintaining the Standby Database in Recovery Mode](#)" on page 16-15.

Displaying Archived Redo Log Information

The following data dictionary views contain useful information about the archived redo logs:

Views	Description
V\$DATABASE	Identifies whether the database is in ARCHIVELOG or NOARCHIVELOG mode.
V\$ARCHIVED_LOG	Displays archived log information from the control file.
V\$ARCHIVE_DEST	Describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.
V\$LOG	Displays all online redo log groups for the database and indicates which need to be archived.
V\$LOG_HISTORY	Contains log history information such as which logs have been archived and the SCN range for each archived log.

For example, the following query displays which online redo log group requires archiving:

```
SELECT group#, archived FROM sys.v$log;
```

```
GROUP#      ARC
-----  ---
1           YES
2           NO
```

To see the current archiving mode, query the V\$DATABASE view:

```
SELECT log_mode FROM sys.v$database;
```

```
LOG_MODE
-----
NOARCHIVELOG
```


The SQL*Plus statement ARCHIVE LOG LIST also shows archiving information for the connected instance:

```
ARCHIVE LOG LIST;

Database log mode                ARCHIVELOG
Automatic archival                ENABLED
Archive destination                /oracle/log
Oldest online log sequence        30
Next log sequence to archive      31
Current log sequence number       33
```

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 destination of the archived redo log (operating system specific).
- The oldest filled online redo log group has a sequence number of 30.
- The next filled online redo log group to be archived has a sequence number of 31.
- The current online redo log file has a sequence number of 33.

You must archive all redo log groups with a sequence number equal to or greater than the *Next log sequence to archive*, yet less than the *Current log sequence number*. For example, the display above indicates that the online redo log groups with sequence numbers 31 and 32 need to be archived.

See Also: For more information on the data dictionary views, see the *Oracle8i Reference*.

Archiving Redo Logs to Multiple Locations

You can specify a *single* destination or multiple destinations for the archived redo logs. Oracle recommends archiving your logs to different disks to guard against file corruption and media failure.

Specify the number of locations for your archived logs by setting either of two mutually exclusive sets of initialization parameters:

- LOG_ARCHIVE_DEST_1 (where *n* is an integer from 1 to 5)
- LOG_ARCHIVE_DEST in conjunction with the optional initialization parameter LOG_ARCHIVE_DUPLEX_DEST

The first method is to use the LOG_ARCHIVE_DEST_ *n* parameter to specify from one to five different destinations for archival. Each numerically-suffixed parameter uniquely identifies an individual destination, e.g., LOG_ARCHIVE_DEST_1, LOG_ARCHIVE_DEST_2, etc. Use the LOCATION keyword to specify a pathname or the SERVICE keyword to specify a net service name (for use in conjunction with a standby database).

The second method, which allows you to specify a maximum of two locations, is to use the LOG_ARCHIVE_DEST parameter to specify a *primary* archive destination and the LOG_ARCHIVE_DUPLEX_DEST to determine an optional *secondary* location. Whenever Oracle archives a redo log, it archives it to every destination specified by either set of parameters.

Note: You can also mirror your archived logs at the O/S level.

To set the archiving destination using LOG_ARCHIVE_DEST_ *n*:

1. Edit the LOG_ARCHIVE_DEST_ *n* parameter to specify from one to five archiving locations. For example, enter:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION=/disk1/arc/'
LOG_ARCHIVE_DEST_2 = 'LOCATION=/disk2/arc/'
LOG_ARCHIVE_DEST_3 = 'LOCATION=/disk3/arc/'
```

2. Edit the LOG_ARCHIVE_FORMAT parameter, using %s to include the log sequence number as part of the filename and %t to include the thread number. Use capital letters (%S and %T) to pad the filename to the left with zeros. For example, enter:

```
LOG_ARCHIVE_FORMAT = arch%t_%s.arc
```

For example, this setting will result in the following files for log sequence numbers 100-102 on thread 1:

```
/disk1/arc/arch1_100.arc, /disk1/arc/arch1_101.arc, /disk1/arc/arch1_102.arc,
/disk2/arc/arch1_100.arc, /disk2/arc/arch1_101.arc, /disk2/arc/arch1_102.arc,
/disk3/arc/arch1_100.arc, /disk3/arc/arch1_101.arc, /disk3/arc/arch1_102.arc
```

3. If the database is open, start a SQL*Plus session and shut down the database. For example, enter:

```
SHUTDOWN IMMEDIATE;
```

4. Mount or open the database to enable the settings. For example, enter:

```
STARTUP;
```

To set archiving destinations with LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST:

1. Edit the `init.ora` file, specifying destinations for the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` parameter. If the database is open, you can also edit the parameter dynamically using the `ALTER SYSTEM` command.

For example, change the parameter to read:

```
LOG_ARCHIVE_DEST = '/disk1/arc'
LOG_ARCHIVE_DUPLEX_DEST_2 = '/disk2/arc'
```

2. Edit the `LOG_ARCHIVE_FORMAT` parameter, using `%s` to include the log sequence number as part of the filename and `%t` to include the thread number. Use capital letters (`%S` and `%T`) to pad the filename to the left with zeroes. If the database is open, you can alter the parameter using the `ALTER SYSTEM` command.

For example, enter:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc
```

For example, this setting will result in the following files for log sequence numbers 300-302 on thread 1:

```
/disk1/arc/arch_1_300.arc, /disk1/arc/arch_1_301.arc, /disk1/arc/arch_1_302.arc,
/disk2/arc/arch_1_300.arc, /disk2/arc/arch_1_301.arc, /disk2/arc/arch_1_302.arc
```

3. If the database is open, start a SQL*Plus session and shut down the database. For example, enter:

```
SHUTDOWN IMMEDIATE;
```

4. Mount or open the database to enable the settings in the `init.ora` file. For example, enter:

```
STARTUP;
```

Oracle provides you with a number of useful archiving options. See the *Oracle8i Administrator's Guide* for a complete account of how to:

- Obtain status information for each archive destination. For example, you can determine whether there was a problem archiving to a specific destination.
- Specify archiving in *normal archiving transmission* or *standby transmission* mode. Normal transmission involves archiving files to a local disk. Standby transmission involves archiving files via a network to either a local or remote standby database.

- Specify the minimum number of destinations to which Oracle must successfully archive.
- Specify when and how often *ARCn* attempts to re-archive to a failed destination.
- Specify up to ten *ARCn* processes for each database instance to parallelize archiving operations.
- Use the LogMiner utility to analyze the contents of online and archived redo logs.

Developing a Backup and Recovery Strategy

This chapter offers guidelines and considerations for developing an effective backup and recovery strategy. It includes the following topics:

- [Choosing Backup Types](#)
- [Choosing Backup Methods](#)
- [Choosing Backup Formats](#)
- [Developing a Backup Strategy](#)
- [Developing a Recovery Strategy](#)

Choosing Backup Types

When developing your backup strategy, you need to know which types of backups you can perform. In each type of physical backup you either back up a file or group of files. This section defines and describes:

- [Whole Database Backups](#)
- [Tablespace Backups](#)
- [Datafile Backups](#)
- [Control File Backups](#)

Logical Backups, also known as *exports*, are described in detail in *Oracle8i Utilities*.

Whole Database Backups

A *whole database backup* should include backups of the control file along with all database files that belong to a database. Whole database backups are the most common type of backup.

Whole database backups do not require you to operate the database in a specific archiving mode. Before performing whole database backups, however, be aware of the implications of backing up in ARCHIVELOG and NOARCHIVELOG modes (see "[Choosing Between NOARCHIVELOG and ARCHIVELOG Mode](#)" on page 4-4).

Whole database backups can be consistent or inconsistent. This section contains the following topics:

- [Consistent Backups](#)
- [Inconsistent Backups](#)

See Also: For detailed procedures for backing up a database using RMAN commands, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#).

For detailed procedures for backing up a database using O/S commands, see [Chapter 13, "Performing Operating System Backups"](#).

Consistent Backups

A *consistent backup* of a whole database is a backup in which all read-write datafiles and control files have been checkpointed with respect to the same SCN. In addition, all the online, read-write datafiles are not "fuzzy," i.e., do not contain changes beyond the SCN in the header. Oracle determines whether a restore backup is

consistent by checking all the datafile headers against the datafile header information contained in the control file.

The control files and datafiles are made consistent during a database checkpoint. The only tablespaces in a consistent backup that are allowed to have older SCNs are read-only and offline normal tablespaces, which are still consistent with the other datafiles in the backup because no changes have been made to these tablespaces and so no recovery is required. If the offline datafile's checkpoint SCN matches the offline-SCN in the control file, then Oracle know the datafile needs no redo.

The important point is that you can open the database after restoring a consistent whole database backup *without applying redo logs* because the data is already consistent: no action is required to make the data in the restored datafiles correct. Consequently, you can restore a year-old consistent backup of your database without performing media recovery and without Oracle performing instance recovery.

WARNING: Only use a backup control file created during a consistent whole database backup if you are restoring the whole database backup and do not intend to perform recovery. If you intend to perform recovery and have a current control file, do not restore an older control file—unless performing point-in-time recovery to a time when the database structure was different from the current structure.

The only way to make a consistent whole database backup is to shut down the database cleanly and make the backup while the database is closed. If a database is not shut down cleanly, e.g., an instance fails or you issue a SHUTDOWN ABORT statement, the database's datafiles are always inconsistent—unless you opened the database in read-only mode. Instance recovery will be required at open time.

A consistent whole database backup is the only valid backup option for databases running in NOARCHIVELOG mode, because otherwise redo will need to be applied to create consistency—and in NOARCHIVELOG mode Oracle overwrites redo records without archiving them first.

To make a consistent database backup current or to take it to a non-current point in time, perform media recovery. If you use a current control file for recovery, Oracle starts media recovery beginning at the lowest checkpoint SCN in the datafile headers; if you use a backup control file, Oracle starts media recovery using the lowest of the following: the control file SCN and the lowest SCN in the datafile headers.

To perform media recovery either apply archived redo logs or, if you are using Recovery Manager, apply incremental backups and/or archived logs. All redo data is located in the archived and online redo logs.

Inconsistent Backups

An *inconsistent backup* of a whole database is a backup in which all read-write datafiles and control files have not been checkpointed with respect to the same SCN. For example, one datafile header may contain an SCN of 100 while others contain an SCN of 95. Oracle will not open the database until these SCNs are made consistent, i.e., until all changes recorded in the online redo logs have been made to the datafiles.

If your database must be up and running 24 hours a day, 7 days a week, you have no choice but to perform inconsistent backups of a whole database. For example, a backup of an offline tablespace in an open database is inconsistent with other tablespaces because portions of the database are being modified and written to disk while the backup of the tablespace is progressing. The datafile headers for the online and offline datafiles may contain inconsistent SCNs. You must run your database in ARCHIVELOG mode to make open backups.

If you run the database in ARCHIVELOG mode, you can construct a whole database backup using backups of datafiles taken at different times. For example, if your database contains seven tablespaces, and you back up the control file as well as a different tablespace each night, in a week you will back up all tablespaces in the database as well as the control file. You can consider this backup as a whole database backup.

Inconsistent Closed Backups You have the option of making inconsistent closed backups if a database is backed up after a system crash or SHUTDOWN ABORT. This type of backup is valid if the database is running in ARCHIVELOG mode, because both online and archived redo logs are available to make the backup consistent.

Note: Oracle recommends that you do not make inconsistent, closed database backups in NOARCHIVELOG mode.

If you run your database in NOARCHIVELOG mode, only back it up when you have closed it cleanly using the IMMEDIATE or NORMAL options. Inconsistent whole database backups of databases running in NOARCHIVELOG mode are

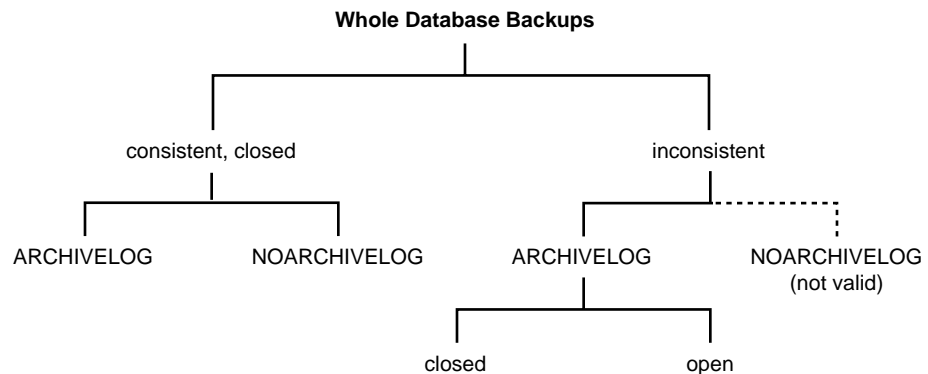
usable *only if* the redo logs containing the changes made prior to the backup are available when you restore it—an unlikely occurrence.

The reason that NOARCHIVELOG inconsistent backups are not recommended is that the datafile headers of the backed up files contain different SCNs (a normal shutdown will guarantee the consistency of these SCNs), and because the database is in NOARCHIVELOG mode, no archived redo logs are available to apply the lost changes. For this reason, RMAN does not allow you to backup a database that has been running in NOARCHIVELOG mode and shut down abnormally because the backup is not usable for recovery.

The moral is: if you run your database in NOARCHIVELOG mode, always aim to have a backup that is usable without performing any recovery. This aim is defeated if you need to apply redo from logs to recover a backup. If you need redo data to make a database consistent, operate in ARCHIVELOG mode.

Figure 3–1 illustrates the various options available to you when perform whole database backups:

Figure 3–1 Whole Database Backup Options



Archiving Unarchived Redo Log Files After open or inconsistent closed backups, always guarantee that you will have the redo necessary to recover the backup by archiving the unarchived redo logs. When the database is open, issue the following SQL statement to force Oracle to switch out of the current log and archive it as well as all other unarchived logs:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

When the database is mounted, open, or closed, issue the following SQL statement to force Oracle to archive all non-current redo logs:

```
SQL> ALTER SYSTEM ARCHIVE LOG ALL;
```

When the database is mounted, open, or closed, issue the following SQL statement to archive a specified log group, where *integer* is the number of the group:

```
SQL> ALTER SYSTEM ARCHIVE LOG GROUP integer;
```

Afterwards, back up all archived redo logs produced since the backup began. This operation ensures that you can use the backup and also allows you to delete the original archived logs from disk. If you do not have all archived redo logs produced during the backup, you cannot recover the backup because you do not have all the redo records necessary to make it consistent.

Backing Up the Control File Unless you are making a consistent whole database backup, make a binary backup up your control file using the ALTER DATABASE command with the BACKUP CONTROLFILE option, specifying the backup destination in quotes. For example, enter:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/copy/cf.f';
```

You can also back up the control file to a trace file. Use the script in the trace file, which is located in the location specified by the USER_DUMP_DEST parameter, to re-create the control file should it become necessary. Use the following syntax:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

Note that trace backups have one major disadvantage: they have no record of any previous backups made with the old control file.

See Also: For more information about backing up the control file using SQL statements, see "[Performing Control File Backups](#)" on page 13-11.

Tablespace Backups

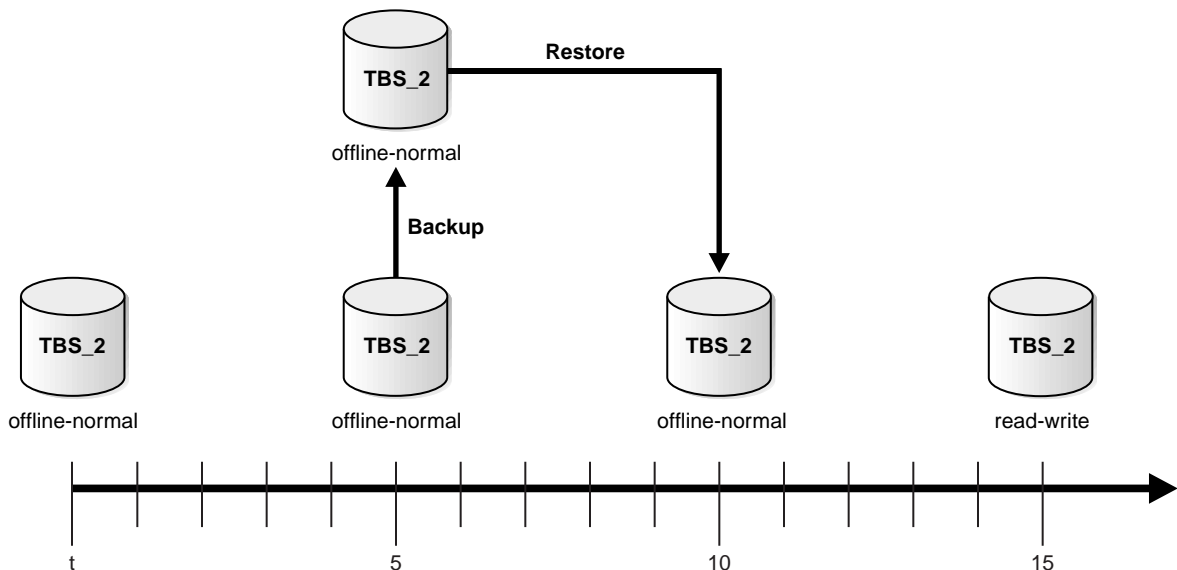
A tablespace backup is a backup of the datafiles that constitute the tablespace. The reason is that a tablespace is a logical grouping, whereas a datafile is a physical file—and only physical files can be physically backed up. For example, if tablespace TBS_2 contains datafiles 2, 3, and 4, then a backup of tablespace TBS_2 backs up those three datafiles.

Tablespace backups, whether online or offline, are only valid if the database is operating in ARCHIVELOG mode. The reason is that redo will be required to make the restored tablespace consistent with the other tablespaces in the database.

The only time a tablespace backup is valid for a database running in NOARCHIVELOG mode is when the tablespace is currently read-only or offline-normal. These cases are exceptions because no redo is required to recover them. For example, take the scenario depicted in Figure 3-2:

1. You take tablespace TBS_2 offline normal at some time during day t .
2. You make a backup of TBS_2 at day $t + 5$.
3. You restore tablespace TBS_2 at day $t + 10$ using the backup made at day $t + 5$.
4. You make tablespace TBS_2 read-write at day $t + 15$.

Figure 3-2 *Tablespace Backups in NOARCHIVELOG Mode*



Because there were necessarily no changes to the offline tablespace between $t + 5$ and $t + 10$, Oracle does not require media recovery. If you make the tablespace read-write at $t + 15$ and then subsequently attempt to restore the $t + 5$ backup, however, Oracle requires media recovery for the changes after $t + 15$. Consequently, you will only be able to open the database if all necessary redo is located in the online redo logs.

Datafile Backups

A datafile backup is a backup of a single datafile. Datafile backups, which are not as common as tablespace backups, are valid in ARCHIVELOG databases.

The only time a datafile backup is valid for a database in NOARCHIVELOG mode is if every datafile in a tablespace is backed up. You cannot restore the database unless all datafiles are backed up. The datafiles must be read-only or offline-normal.

Control File Backups

A control file backup is a copy of a database's control file. If the database is mounted, you can issue the following SQL statement, where *controlfile_location* is the name for the backup:

```
ALTER DATABASE BACKUP CONTROLFILE TO 'controlfile_location';
```

You can also back up to a trace file: the trace file contains a script for re-creating the control file. The statement is as follows:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

You can also use the RMAN **backup current controlfile** command.

See Also: To learn how to make control file backups using SQL*Plus, see "[Backing Up the Control File After Structural Changes](#)" on page 2-5. To learn how to make control file backups using RMAN, see "[Backing Up Control Files](#)" on page 8-6.

Choosing Backup Methods

You can make physical backups using Recovery Manager or O/S commands and logical backups using a utility such as Export:

Table 3–1 Requirements for Different Backup Methods

Backup Method	Type	Version Available	Requirements
Recovery Manager (RMAN)	Physical	Oracle version 8.0 and higher	Media manager (if backing up to tape)
O/S	Physical	All versions	O/S backup utility (for example, UNIX dd)
Export	Logical	All versions	N/A
Enterprise Backup Utility (EBU)	Physical	Oracle7 only	Media manager

Recovery Manager

The Recovery Manager (RMAN) utility manages Oracle backup and recovery operations. RMAN uses information about the database stored in the control file to automatically locate, back up, restore, and recover database files—including datafiles, control files, and archived redo logs. For example, RMAN can:

- Back up and restore the database, tablespaces, datafiles, control files, and archived redo logs.
- Configure frequently executed backup operations.
- Generate a printable log of all backup and recovery actions.
- Use the recovery catalog or the target database control file to automate both media restore and recovery operations.
- Perform automatic parallelization of backups and restores.
- Crosscheck a media manager to ensure that backed up files are still available.
- Find datafiles that require a backup based on user-specified limits on the amount of redo that must be applied.
- Use the Oracle Enterprise Manager GUI interface.
- Schedule backups automatically via Oracle Enterprise Manager.

See Also: For a conceptual overview of Recovery Manager, see ["Overview of Recovery Manager"](#) on page 4-2.

Recovery Manager Metadata

Recovery Manager obtains necessary information for its backup and recovery operations from either the control file or from an optional repository called a *recovery catalog*. The recovery catalog, which is maintained by RMAN, is a warehouse of information obtained from the control files of its target databases. RMAN refers to the data in the recovery catalog when directing server processes during restore and recover operations.

See Also: For a conceptual overview of the Recovery Manager metadata, see ["Recovery Manager Metadata"](#) on page 4-13. To learn how to manage the RMAN metadata, see [Chapter 6, "Managing Recovery Manager Metadata"](#).

Media Management

To utilize tape storage for your Oracle database backups, RMAN requires a [media manager](#). A media manager is a third-party software utility that loads, labels, and unloads sequential media such as tape drives for the purpose of backing up and recovering data.

The Oracle Backup Solutions Program (BSP) allows third-party vendors to easily integrate their products with the Recovery Manager; you can access online information about BSP via the following URL:

<http://www.oracle.com/st/products/features/backup.html>

Although media management software offers you the additional flexibility of backing up to both disk and tertiary storage, a media manager is not required if you only intend to back up to disk.

See Also: For a conceptual overview of media management, see "[Media Management](#)" on page 4-16. To learn what you need to do to configure a media manager for use with RMAN, see "[Configuring a Media Manager](#)" on page 5-19.

Enterprise Manager

Although RMAN is commonly used as a command-line utility, you can also perform RMAN backups using Oracle Enterprise Manager. Oracle Enterprise Manager-Backup Manager is a GUI interface to Recovery Manager that enables you to perform backup and recovery via a point-and-click method.

See Also: For information about performing backup and recovery using Oracle Enterprise Manager, see the *Oracle Enterprise Manager Administrator's Guide*.

Operating System (O/S)

You can perform an O/S backup with a native command on your O/S. For example, you can use the `cp` command to back up files in UNIX. In this case, you must write and maintain UNIX scripts to control the O/S backup.

See Also: For information about the utilities available on your operating system, see your operating system-specific documentation.

Export

The Oracle Export utility writes data from an Oracle database to operating system files in an Oracle-specific format. Export files store information about schema objects created for a database. Because the Oracle Export utility can selectively

export specific objects, you may consider exporting portions of or all of a database for supplemental protection and flexibility. Database exports are not a substitute for whole database backups.

See Also: For information about using exports to supplement your backup strategy, see *Oracle8i Utilities*.

Enterprise Backup Utility

The Oracle Enterprise Backup Utility (EBU) is a utility that automates backups of Oracle7 databases; it is not compatible with Oracle8i databases.

Feature Comparison of Backup Methods

Table 3–2 compares the features of the backup methods described in this chapter:

Table 3–2 Feature Comparison of Backup Methods

Feature	Recovery Manager	Operating System	Export
Closed database backups	Supported. Requires instance to be mounted.	Supported.	Not supported.
Open database backups	Do not use BEGIN/END BACKUP commands.	Use BEGIN/END BACKUP commands.	Requires RBS to generate consistent backups.
Incremental backups	Supported.	Not supported.	Not supported.
Corrupt block detection	Supported. Identifies corrupt blocks and writes to V\$BACKUP_CORRUPTION or V\$COPY_CORRUPTION.	Not supported.	Supported. Identifies corrupt blocks in the export log.
Automatic backup	Supported. Establishes the name and locations of all files to be backed up (whole database, tablespace, datafile or control file backup).	Not supported. Files to be backed up must be specified manually.	Supported. Performs either full, user, or table backups.
Backup catalogs	Supported. Backups are cataloged in the recovery catalog and in the control file, or just in the control file.	Not supported.	Not supported.

Table 3–2 Feature Comparison of Backup Methods (Cont.)

Feature	Recovery Manager	Operating System	Export
Backups to sequential media	Supported. Interfaces with a media manager. RMAN also supports proxy copy, a feature that allows the media manager to manage the transfer of data.	Supported. Backup to tape is manual or managed by a media manager.	Supported.
Backs up <code>init.ora</code> and password files	Not supported.	Supported.	Not supported.
Operating system independent language	Supported.	Not supported.	Supported.

Choosing Backup Formats

The format of your backup is contingent on the method you use to make it. You can back up files in the following formats:

- [Backup Sets](#)
- [Image Copies](#)
- [Operating System Backups](#)
- [Logical Backups](#)

Backup Sets

When you issue the RMAN **backup** command (and do not specify the **proxy** option), you create a backup set. A backup set is a logical structure containing one or more physical *backup pieces*. Typically, a backup set contains one backup piece. Backup sets can:

- Contain either archived redo logs or datafiles, but not both.
- Be written to disk or tertiary storage.
- Constitute full or incremental backups.
- Span multiple O/S files.

Backup sets are in an Oracle proprietary format, which means that an Oracle instance cannot use files in a backup set until RMAN restores them to an instance-usable format. For example, a tablespace backup in a backup set is a

compressed version of each file in the tablespace; you must use an RMAN command to restore the datafiles in the backup set.

When you specify datafiles that you want to back up, an Oracle server session reads the files and creates the backup set. You do *not* need to precede an RMAN backup with the ALTER TABLESPACE BEGIN BACKUP statement (for information about how blocks are read in RMAN backups, see "[Detection of Logical Block Corruption](#)" on page 4-55).

RMAN can include datafiles, control files, or archived redo logs in a backup set. If you perform a whole database backup, then RMAN automatically backs up every file in that database as well as the control file. You can also tell RMAN to include a control file backup in any datafile backup set.

Note: You cannot combine archived redo logs and datafiles in the same backup set.

See Also: For a conceptual overview of Recovery Manager backups, see "[Backup Sets](#)" on page 4-28. To learn how to make RMAN backups, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#). For reference information for the RMAN **backup** command, see "[backup](#)" on page 11-21.

Image Copies

Create an *image copy* using the RMAN **copy** command of any of the following objects:

- Datafiles
- Control files
- Archived redo logs

When you issue the command, an Oracle server session—not an O/S routine—reads the file and writes the copy out to disk. You do *not* need to precede an RMAN copy with the ALTER TABLESPACE BEGIN BACKUP statement.

Image copies can be used immediately by an Oracle instance, i.e., they are already in an instance-usable format. You can only make image copies to disk.

Note: You can make proxy copies to tape. Proxy copies are not image copies, but a type of media-managed backup generated with the **proxy** option of the **backup** command.

See Also: For an overview of RMAN image copies, see "[Image Copies](#)" on page 4-45. To learn how to make RMAN image copies, see "[Making Image Copies](#)" on page 8-10. For information on the RMAN **copy** and **backup** commands, see "[copy](#)" on page 11-48 and "[backup](#)" on page 11-21.

Operating System Backups

Create operating system (O/S) backups using an operating system command such as the UNIX **dd**. You can write O/S backups to disk or tape in any format that your O/S utilities support. Recovery Manager can catalog and use O/S backups that are image backups on disk.

See Also: To learn how to make operating system backups, see [Chapter 13, "Performing Operating System Backups"](#).

Logical Backups

Logical backups store information about the schema objects created for a database. Use the Export utility to write data from an Oracle database to operating system files that have an Oracle database format.

Because the Oracle Export utility can selectively export specific objects or portions of an object, you can export portions or all of a database for supplemental protection and flexibility in a database's backup strategy. Database exports are not a substitute for physical backups and cannot provide the same complete recovery advantages that the built-in functionality of Oracle offers.

See Also: For more information about the Export Utility, see *Oracle8i Utilities*.

Developing a Backup Strategy

Before you create an Oracle database, decide how to protect the database against potential media failures. If you do not develop a backup strategy before creating your database, you may not be able to perform recovery should a disk failure damage the datafiles, online redo log files, or control files

This section describes general guidelines that can help you decide when to perform database backups and which parts of a database you should back up. Of course, the specifics of your strategy will depend on the constraints under which you are operating. No matter which backup strategy you implement, however, follow these guidelines whenever possible:

- [Decide Whether to Run in ARCHIVELOG or NOARCHIVELOG Mode](#)
- [Multiplex Control Files, Online Redo Logs, and Archived Redo Logs](#)
- [Perform Backups Frequently and Regularly](#)
- [Perform Backups When You Make Structural Changes](#)
- [Perform Frequent Backups of Often-Used Tablespace](#)
- [Perform Backups After Unrecoverable/Unlogged Operations](#)
- [Store Older Backups](#)
- [Know the Constraints for Distributed Database Backups](#)
- [Export Database Data for Added Protection and Flexibility](#)
- [Do Not Back Up Online Redo Logs](#)

Decide Whether to Run in ARCHIVELOG or NOARCHIVELOG Mode

Before you create an Oracle database, decide how you plan to protect it against potential failures. Answer the following questions:

- **Is it acceptable to lose any data if a disk failure damages some of the files that constitute a database?** If not, run the database in ARCHIVELOG mode, ideally with a multiplexed online redo log, a multiplexed control file, and multiplexed archive redo logs. If you can afford to lose a limited amount of data, you can operate in NOARCHIVELOG mode and avoid the extra maintenance chores.
- **Will you need to recover to a non-current time?** If you need to perform incomplete recovery to correct an erroneous change to the database, run in ARCHIVELOG mode and perform control file backups whenever making structural changes. Incomplete recovery is helped by having a backup control file reflecting the database structure at the desired time.
- **Does the database need to be available at all times?** If so, do not operate the database in NOARCHIVELOG mode because the required whole database backups, taken while the database is shutdown, cannot be made frequently, if at all. Therefore, high-availability databases always operate in ARCHIVELOG mode to take advantage of open datafile backups.

Once you have answered these questions and have determined which mode to use, follow the guidelines for either:

- [Backing Up in NOARCHIVELOG Mode](#)
- [Backing Up in ARCHIVELOG Mode](#)

Backing Up in NOARCHIVELOG Mode

If you operate your database in NOARCHIVELOG mode, Oracle does not archive filled groups of online redo log files. Therefore, the only protection against a disk failure is the most recent whole backup of the database. Follow these guidelines:

- Plan to make whole database backups regularly, according to the amount of work that you can afford to lose. For example, if you can afford to lose the amount of work accomplished in one week, make a consistent whole database backup once per week. If you can afford to lose only a day's work, make a consistent whole database backup every day. For large databases with a high amount of activity, you usually cannot afford to lose work. In this case, you should operate the database in ARCHIVELOG mode.
- Whenever you alter the physical structure of a database operating in NOARCHIVELOG mode, immediately take a consistent whole database backup. A whole database backup fully reflects the new structure of the database.

Backing Up in ARCHIVELOG Mode

If you run your database in ARCHIVELOG mode, *ARCn* archives groups of online redo log files. Therefore, the archived redo log coupled with the online redo log and datafile backups can protect the database from a disk failure, providing for complete recovery from a disk failure to the instant that the failure occurred (or, to the desired non-current time). Following are common backup strategies for a database operating in ARCHIVELOG mode:

- Perform a whole database backup of the entire database after you create it. This initial whole database backup is the foundation of your backups because it provides backups of all datafiles and the control file of the associated database.

Note: When you perform this initial whole database backup, make sure that the database is in ARCHIVELOG mode first. Otherwise, the backup control files will contain the NOARCHIVELOG mode setting.

- Make open database or tablespace backups to keep your database backups up-to-date. Subsequent whole database backups are not required, and if a database must remain open at all times, whole database backups while the database is closed are not feasible.
- Make open or closed datafile backups to update backed up information for the database. Doing so supplements the initial whole database backup. In particular, back up the datafiles of extensively used tablespaces frequently to reduce database recovery time. If a more recent datafile backup restores a damaged datafile, you need to apply less redo data (or incremental backups) to the restored datafile to roll it forward to the time of the failure.

Whether you should take open or closed datafile backups depends on the availability requirements of the data. Open datafile backups are the only choice if the data being backed up must always be available.

You can also use a datafile copy taken while the database is open and the tablespace is online to restore datafiles. You must apply the appropriate redo log files to these restored datafiles to make the data consistent and bring it forward to the specified point in time.

- Make a control file backup every time you make a structural change to the database. If you are operating in ARCHIVELOG mode and the database is open, use either RMAN or the ALTER DATABASE statement with the BACKUP CONTROLFILE option.

Multiplex Control Files, Online Redo Logs, and Archived Redo Logs

The control file, online redo log, and archived redo log are crucial files for backup and recovery operations. The loss of any of these files can cause you to lose data irrevocably. You should maintain:

- At least two copies of the control file on different disks mounted under different disk controllers. You can either use Oracle to multiplex the copies or your O/S to mirror them.
- Two or more copies of your online redo log on different disks. The online redo data is crucial for instance, crash, and media recovery.
- Two or more copies of your archived redo log on different disks and, if possible, different media.

See Also: To learn how to integrate data structure management into your backup and recovery strategy, see [Chapter 2, "Managing Data Structures"](#). For a thorough conceptual overview of all Oracle data structures, see *Oracle8i Concepts*.

Perform Backups Frequently and Regularly

Frequent backups are essential for any recovery scheme. Base the frequency of backups on the rate or frequency of database changes such as:

- Addition and deletion of tables.
- Insertions and deletions of rows in existing tables.
- Updates to data within tables.

If users generate a significant amount of DML, database backup frequency should be proportionally high. Alternatively, if a database is mainly read-only, and updates are issued only infrequently, you can back up the database less frequently.

Use either Recovery Manager or O/S methods to create backup scripts. RMAN scripts, which are stored in the recovery catalog, are an especially efficient method for performing routine, repetitive backup operations.

See Also: For an overview of RMAN stored scripts, see ["Stored Scripts"](#) on page 4-10. To learn how to create, delete, replace, and print stored scripts, see ["Storing Scripts in the Recovery Catalog"](#) on page 6-20.

Perform Backups When You Make Structural Changes

Administrators as well as users make changes to a database. If you make any of the following structural changes, perform a backup of the appropriate portion of your database immediately before and after completing the alteration:

- Create or drop a tablespace.
- Add or rename a datafile in an existing tablespace.
- Add, rename, or drop an online redo log group or member.

The part of the database that you should back up depends on your archiving mode:

Mode	Action
ARCHIVELOG	Make a control file backup (using the ALTER DATABASE statement with the BACKUP CONTROLFILE option) before and after a structural alteration. Of course, you can back up other parts of the database as well.
NOARCHIVELOG	Make a consistent whole database backup immediately before and after the modification.

Perform Frequent Backups of Often-Used Tablespaces

Many administrators find that regular whole database backups are not in themselves sufficient for a robust backup strategy. If you run in ARCHIVELOG mode, then you can back up the datafiles of an individual tablespace or even a single datafile. This option is useful if a portion of a database is used more extensively than others, e.g., the SYSTEM tablespace and tablespaces that contain rollback segments. By making more frequent backups of the extensively used datafiles of a database, you gather more recent copies of the datafiles.

For example, you may make a whole database backup once a week on Sunday. If your database experiences heavy traffic during the week, a media failure on Friday can force you to apply a tremendous amount of redo data during recovery. If you had backed up your most frequently accessed tablespaces three times a week, you would have to apply a smaller number of changes to data to roll the restored file forward to the time of the failure.

Perform Backups After Unrecoverable/Unlogged Operations

If users are creating tables or indexes using the UNRECOVERABLE option, consider taking backups after the objects are created. When tables and indexes are created as UNRECOVERABLE, Oracle does not log redo data, which means that you cannot recover these objects from existing backups.

Note: If using RMAN, you can take an incremental backup.

See Also: For information about the UNRECOVERABLE option, see the CREATE TABLE ... AS SELECT and CREATE INDEX commands in the *Oracle8i SQL Reference*.

Perform Whole Database Backups After Using the RESETLOGS Option

After you have opened a database with the RESETLOGS option, Oracle recommends that you immediately perform a whole database backup. If you do not, and a disaster occurs, then you will lose all work performed since opening the database.

When you open a database with the RESETLOGS option, Oracle automatically:

- Creates a new incarnation of the database, putting a new RESETLOGS SCN in the headers of all database files.

- Resets the log sequence number to 1.
- Reformats the online redo log files if they exist, and otherwise creates them.

Oracle performs these actions so that it can identify which archived redo logs apply to which incarnations of the database.

The resetting of the online redo logs has severe consequences for your backup and recovery strategy. As a rule, you cannot restore pre-RESETLOGS backups, so all your old backups are worthless in the new incarnation. The only exceptions are:

- Backups of read-only tablespaces that were not made read-write again before the RESETLOGS.
- Backups of offline normal tablespaces that were not brought online again before the RESETLOGS.
- Backups of read-write tablespaces made after an incomplete recovery and *immediately* before the RESETLOGS.

See Also: For more information, see ["Recovering a Pre-RESETLOGS Backup"](#) on page 14-35.

Backup Options While in NOARCHIVELOG Mode

If you are operating in NOARCHIVELOG mode, your only option is to make a cold, consistent, whole database backup. If you do not, then your backups are inconsistent and require recovery before the database can be opened. Because no redo data is available for recovery, Oracle will prevent you from opening the database.

Backup Options in ARCHIVELOG mode

If you are operating in ARCHIVELOG mode, you can either perform a consistent, closed, whole database backup or an inconsistent, open database backup. The option you choose depends on:

- The amount of time before the database must be available.
- The importance of data entered after you open the database with the RESETLOGS option.

If your most important criterion is getting the database up and running, then you must perform open database backups. You run a risk when you perform an open database backup—if the backups do not complete before you have another media failure, then you will lose all changes made since opening the database with the

RESETLOGS option. You cannot use a backup taken before opening the database with RESETLOGS to recover this incarnation of the database.

Note: There is one and only one exception to this rule. See ["Recovering a Pre-RESETLOGS Backup"](#) on page 14-35.

If the most important criterion is to restore in case of another failure, then you may elect to take a consistent, closed database backup. Oracle recommends that you perform a consistent whole database backup whenever possible.

Store Older Backups

You should store older backups for two basic reasons:

- An older backup is necessary to perform incomplete recovery to a time before your most recent backup.
- Your most recent backup is corrupted.

If you want to recover to a non-current time, you need a database backup that completed before the desired time. For example, if you make backups on the 1st and 14th of February, then decide at the end of the month to recover your database to February 7th, you must use the February 1st backup.

For a database operating in NOARCHIVELOG mode, the backup you use should be a consistent whole database backup. Of course, you will not be able to perform media recovery using this backup. For a database operating in ARCHIVELOG mode, your whole database backup:

- Does not need to be consistent because redo is available to recover it.
- Should have completed before the intended recovery time (the control file should reflect the database's structure at the point-in-time of recovery).
- Should have all archived logs necessary to recover the datafiles to the required point-in-time.

For added protection, keep two or more database backups (with associated archived redo logs) previous to the current backup. Thus, if your most recent backups are not usable, you will not lose all of your data.

WARNING: After you open the database with the **RESETLOGS** option, you cannot use existing backups for subsequent recovery beyond the time when the logs were reset. You should therefore shut down the database and make a consistent whole database backup. Doing so will enable recovery of database changes after using the **RESETLOGS** option.

Know the Constraints for Distributed Database Backups

If your database is a node in a distributed database, all databases in the distributed database system should operate in the same archiving mode. Note the following consequences and constraints:

Mode	Constraint	Consequence
ARCHIVELOG	Closed cleanly.	Backups at each node can be performed autonomously, i.e., individually and without time coordination.
NOARCHIVELOG	Closed cleanly.	Consistent whole database backups must be performed at the same global time to plan for global distributed database recovery. For example, if a database in New York is backed up at midnight EST, the database in San Francisco should be backed up at 9 PM PST.

See Also: To learn about performing media recovery in distributed systems, see ["Opening the Database after Media Recovery"](#) on page 14-33.

Export Database Data for Added Protection and Flexibility

Because the Oracle Export utility can selectively export specific objects, consider exporting portions or all of a database for supplemental protection and flexibility in a database's backup strategy. This strategy is especially useful for recovery catalog backups when using RMAN.

Note that Database exports are not a substitute for whole database backups and cannot provide the same complete recovery advantages that the built-in functionality of Oracle offers.

See Also: For a complete account of the Export utility, see the *Oracle8i Utilities* guide.

Do Not Back Up Online Redo Logs

Although it may seem that you should back up online redo logs along with the datafiles and control file, this technique is dangerous. You should not back up online redo logs for the following reasons:

- The best method for protecting the online logs against media failure is by multiplexing them, i.e., having multiple log members per group, on different disks and disk controllers.
- If your database is in ARCHIVELOG mode, ARCHIVELOG mode is already archiving the redo logs.
- If your database is in NOARCHIVELOG mode, the only type of backups that you should perform are closed, consistent, whole database backups. The files in this type of backup are all consistent and do not need recovery, so the online logs are not needed.

The danger in backing up online redo logs is that you may accidentally restore them while not intending to. There are a number of situations where restoring the online logs would cause very significant problems in the recovery process. Following are two scenarios that illustrate how restoring backed up online logs severely compromises recovery.

Scenario 1: Unintentionally Restoring Online Redo Logs When a crisis occurs, it is easy to make a simple mistake. DBAs and system administrators frequently encounter dangers during a database restore. When restoring the whole database, it is also possible to accidentally restore the online redo logs, thus overwriting the current logs with the older, useless backups. This action forces the DBA to perform an incomplete recovery instead of the intended complete recovery, thereby losing the ability to recover valuable transactions contained in your overwritten redo logs.

Scenario 2: Erroneously Creating Multiple Parallel Redo Log Time Lines You can unintentionally create multiple parallel redo log timelines for a single instance database. You can avoid this mistake, however, by making it so that the online logs cannot be restored. You must open the database with the RESETLOGS option, which effectively creates the new redo logs, and also a new database incarnation.

If you face a problem where the best course of action is to restore the database from a consistent backup and not perform any recovery, then you may think it is safe to restore the online logs and thereby avoid opening the database with the RESETLOGS option. If you do so, however, the database will eventually generate a log sequence number that was already generated by the database during the previous timeline.

If you then face another disaster and need to restore from this backup and roll forward, you will find it difficult to identify which log sequence number is the correct one. If you had reset the logs, then you would have created a new incarnation of the database. You could only apply archived redo logs created by this new incarnation to this incarnation.

Note: Recovery Manager and EBU do not back up online redo logs.

Developing a Recovery Strategy

Oracle provides a variety of procedures and tools to assist you with recovery. To develop an effective recovery strategy, learn to do the following:

- [Test Backup and Recovery Strategies](#)
- [Handling Non-Media Failures](#)
- [Recovering from Media Failure](#)

Test Backup and Recovery Strategies

Practice your backup and recovery strategies in a test environment before and after you move to a production system. In this way you can measure the thoroughness of your strategies and minimize problems before they occur in a real situation. Performing test recoveries regularly ensures that your archiving, backup, and recovery procedures work. It also helps you stay familiar with recovery procedures, so that you are less likely to make a mistake in a crisis.

If you use Recovery Manager, use the **duplicate** command to create a test database using backups of your production database. To learn how to duplicate a database, see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#).

Handling Non-Media Failures

Although media recovery is your primary concern when developing your recovery strategy, you should understand the basic types of non-media failures as well as the causes and solutions for each:

- Statement failure
- User process failure
- User error

- Instance failure
- Media failure

Statement Failure

A statement failure is a logical failure in the handling of a statement in an Oracle program. Oracle or the O/S usually returns an error code and a message when a statement failure occurs. [Table 3-3](#) shows typical causes and resolutions for statement failures.

Table 3-3 *Typical Causes and Resolutions for Statement Failures*

Problem	Solution
A logical error occurred in an application.	Fix the program that generated the error so that its logic flows correctly. You may need to enlist the aid of developers to solve this type of problem.
A user attempted to enter illegal data into a table.	Modify the illegal SQL statement and reissue it.
A user attempted an operation with insufficient privileges, e.g., attempting to insert data into a table when the user has only SELECT privileges.	Provide the necessary database privileges for the user to complete the statement successfully.
A user attempted to create a table but exceed the allotted quota limit.	Issue an ALTER USER command to change the quota limit.
A user attempted a table INSERT or UPDATE, causing an extent to be allocated without sufficient free space in the tablespace.	Add space to the tablespace. You can also use the RESIZE and AUTOEXTEND options for datafiles.

User Process Failure

A user process failure is any failure in a user program accessing an Oracle database. User processes can fail for a wide variety of reasons. Some typical scenarios include:

- A user performed an abnormal disconnection in the session.
- The user's session was abnormally terminated. For example, the user rebooted the client while connected to a database in a client-server configuration.
- The user's program raised an address exception that terminated the session.

In most cases, you do not need to act to resolve user process failures: the user process simply fails to function but does not affect Oracle and other user process.

The PMON background process is usually sufficient for cleaning up after an abnormally terminated user process.

User Error

Users errors are any mistakes that users make in adding data to or deleting data from the database. Typical causes of user error are:

- A user accidentally drops or truncates a table.
- A user deletes all rows in a table.
- A user commits data, but discovers an error after the data is committed.

If you have a logical backup of a table from which data has been lost, sometimes you can simply import it back into the table. Depending on the scenario, however, you will probably have to perform some type of incomplete media recovery to correct such errors.

You can perform either database point-in-time recovery (DBPITR) or tablespace point-in-time recovery (TSPITR). The following table explains the difference between these types of incomplete recovery:

Type	Description	Procedure
DBPITR	<ol style="list-style-type: none"> 1. Restore backup database. 2. Roll forward to the time just before the error. 3. Open RESETLOGS. 	<p>For RMAN recovery, see "Performing Incomplete Recovery" on page 9-23.</p> <p>For O/S recovery, see "Performing Incomplete Media Recovery" on page 14-26.</p>
TSPITR	<ol style="list-style-type: none"> 1. Create auxiliary instance. 2. Recover the tablespace on the auxiliary to the time just before the error. 3. Import data back into the primary database. 	<p>For RMAN TSPITR, see Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager".</p> <p>For O/S TSPITR, see Appendix B, "Performing Operating System Tablespace Point-in-Time Recovery".</p>

Instance Failure

Instance failure occurs when an instance abnormally terminates. An instance failure can occur because:

- A power outage causes the server to crash.
- The server becomes unavailable because of hardware problems.

- The O/S crashes.
- One of the Oracle background processes fails.
- You issue a SHUTDOWN ABORT statement.

Fortunately, Oracle performs instance recovery automatically: all you need to do is start the database. Oracle automatically detects that the database was not shut down cleanly, then applies committed and uncommitted redo records in the redo log to the datafiles and rolls back uncommitted data. Finally, Oracle synchronizes the datafiles and control file and opens the database.

Recovering from Media Failure

Media failure is the biggest threat to your data. A media failure is a physical problem that occurs when a computer attempts to read from or write to a file necessary to operate the database. Common types of media problems include:

- A disk drive that held one of the database files experiences a head crash.
- A datafile, online or archived redo log, or control file is accidentally deleted, overwritten, or corrupted.

The technique you use to recover from media failure depends heavily on the type of media failure that occurred. For example, the strategy you use to recover from a corrupted datafile is different from the strategy for recovering from the loss of your control file.

The basic steps for media recovery are:

- Determine which files to recover.
- Determine the type of media recovery required: complete or incomplete, open database or closed database.
- Restore backups or copies of necessary files: datafiles, control files, and the archived redo logs necessary to recover the datafiles.

Note: If you do not have a backup, you can still perform recovery if you have the necessary redo log files and the control file contains the name of the damaged file. If you cannot restore a file to its original location, then you must relocate the restored file and inform the control file of the new location.

- Apply redo records (and/or incremental backups when using Recovery Manager) to recover the datafiles.
- Re-open the database. If you perform incomplete recovery, you must open the database in RESETLOGS mode.

See Also: To learn how to perform media recovery using RMAN, see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#). To learn how to perform media recovery using O/S methods, see [Chapter 14, "Performing Operating System Recovery"](#).

Determine Which Files to Recover

First, you need to determine what to recover. Some types of failure are obvious: for example, the hardware crashes and you need to recover the entire database. In other cases, a single datafile becomes corrupted. Often you can use the table `V$RECOVER_FILE` to determine what requires recovery.

Choose a Type of Recovery

When you perform media recovery, you choose either *complete recovery* or *incomplete recovery*. Following is a list of media recovery operations:

- Complete media recovery

Complete media recovery includes the application of all necessary redo or incremental backups ever generated for the particular incarnation of the database being recovered. Complete media recovery can be performed on offline datafiles while the database is open. Types of complete media recovery are:

 - Closed database recovery
 - Open-database, offline-tablespace recovery
 - Open-database, offline-tablespace, individual datafile recovery
- Incomplete Media Recovery

Incomplete media recovery, also called point-in-time recovery (PITR), produces a version of the database as it was at some time in the past. Incomplete media recovery must either continue on to become complete media recovery, or be terminated by a RESETLOGS operation that creates a new incarnation of the database. The database must be closed for incomplete media recovery operations. You can perform:

 - *time-based recovery*, which recovers the data up to a specified point in time.

- *cancel-based recovery*, which recovers until you issue the CANCEL command.
- *change-based recovery*, which recovers up to a specified SCN.
- *log sequence recovery*, which recovers up to a specified log sequence number.

One important and special type media recovery is *tablespace point-in-time recovery* (TSPITR). TSPITR enables you to recover one or more tablespaces to a point-in-time that is different from the rest of the database.

The type of recovery method you use depends on the situation. [Table 3-4](#) displays typical scenarios and strategies.

Table 3-4 Typical Media Failures and Recovery Strategies

Lost Files	Archiving Mode	Status	Strategy
One or more datafiles	NOARCHIVELOG	Closed	Restore whole database from a consistent database backup. The control file and all datafiles are restored from a consistent backup and the database is opened. All changes made after the backup are lost. Note: The <i>only</i> time you can recover a database in NOARCHIVELOG is when you have not already overwritten the online log files that were current at the time of the most recent backup.
One or more datafiles and an online redo log	NOARCHIVELOG	Closed	Restore whole database from consistent backup. You will lose all changes made since the last backup.
One or more datafiles and all control files	NOARCHIVELOG	Closed	Restore whole database and control file from consistent backup. You will lose all changes made since the last backup.
One or more datafiles	ARCHIVELOG	Open	Perform tablespace or datafile recovery while the database is open. The tablespaces or datafiles are taken offline, restored from backups, recovered, and placed online. No changes are lost and the database remains available during the recovery.
One or more datafiles and an online redo log required for recovery	ARCHIVELOG	Closed	Perform incomplete recovery of the database up to the point of the lost online redo log.

Table 3–4 Typical Media Failures and Recovery Strategies

Lost Files	Archiving Mode	Status	Strategy
One or more datafiles and an archived redo log required for recovery	ARCHIVELOG	Open	Perform TSPITR on the tablespaces containing the lost datafiles up to the point of the latest available redo log.
One or more datafiles and/or all control files	ARCHIVELOG	Not open	Restore the lost files from backups and recover the datafiles. No changes are lost, but the database is unavailable during recovery.
One or more datafiles and/or all control files, as well as an archived or online redo log required for recovery	ARCHIVELOG	Not open	Perform incomplete recovery of the database. You will lose all changes contained in the lost log and in all subsequent logs.

Restore Backups of Datafiles and Necessary Archived Redo Logs

The method you use to restore backups depends on whether you use RMAN or O/S methods to back up your data. If you use RMAN, then issue a **restore** command and let RMAN take over the transfer of data. If you use O/S methods, then you need to identify which files need to be restored and manually copy the backups to their necessary location.

If the database is shut down and you restore one of the datafiles with a backup, either because of a media failure or because for some reason you want to recover the database to a non-current time, Oracle detects an inconsistency between the checkpoint SCN in the datafile headers and the datafile header checkpoint SCNs recorded in the control file at database open time. Oracle then selects the lowest checkpoint SCN recorded in the control file and datafile headers and asks you to begin media recovery starting from a specified log sequence number. You cannot open the database if any of the online datafiles needs media recovery.

There is only one case in which Oracle will tell you that you need to perform media recovery when you do not. If a tablespace is in *hot backup mode* because you issued the ALTER TABLESPACE ... BEGIN BACKUP command and the system crashes, then on the next startup Oracle will issue a message stating that media recovery is required. Media recovery is not really required here, however, since you did not restore a backup; in this case, avoid media recovery by issuing the ALTER DATAFILE ... END BACKUP command. Note that RMAN backups do not have this problem.

See Also: To learn how to restore backups using RMAN, see "[Restoring Datafiles, Control Files, and Archived Redo Logs](#)" on page 9-2. To learn how to restore datafiles using O/S methods, see "[Restoring Files](#)" on page 14-6.

Begin Media Recovery

You have a choice between two basic methods for recovering physical files. You can:

- Use RMAN to automate recovery.
- Execute SQL or SQL*Plus statements.

Obviously, the recovery method you choose is contingent on which backup method you use. For example, if you backed up your files using:

- The RMAN **backup** command, then you must use the RMAN **restore** and **recover** commands for recovery, since these backups are in an RMAN-specific format.
- The RMAN **copy** command, then you can use either RMAN or O/S methods for recovery.
- O/S commands, then you must use your O/S utility for restoring files and the RECOVER (SQL*Plus) or ALTER DATABASE RECOVER (SQL) statements for recovering them.

Note: The only exception is for O/S backups that are image copies on disk. If you register these image copies as datafile copies with RMAN, then RMAN can restore them.

Recovering with RMAN RMAN is a powerful tool that can aid in backup and recovery operations. Using RMAN for recovery allows you to:

- Restore and recover both file copies and RMAN-specific backup sets.
- Minimize administration errors by using the recovery catalog.
- Use RMAN's incremental backup feature to minimize recovery time.
- Generate comprehensive reports on previous backups, datafile copies, unrecoverable files, etc.
- Use scripts stored in the file system or recovery catalog to automate jobs.
- Cooperate with a media manager to restore backups from tape.

See Also: For a conceptual overview of Recovery Manager recovery, see "[Restoring Files](#)" on page 4-47 and "[Media Recovery](#)" on page 4-49. To learn how to perform RMAN recovery, see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#). For reference information for the RMAN **recover** command, see "[recover](#)" on page 11-88.

Recovering with the SQL*Plus RECOVER Command If you do not use RMAN, then you can use O/S methods to restore your backups and SQL*Plus commands to perform media recovery. You can use SQL*Plus commands to:

- Recover both O/S copies and files generated with the RMAN **copy** command.
- Maintain manual control over media recovery.

You can use three basic SQL*Plus commands for recovery:

- RECOVER DATABASE
- RECOVER TABLESPACE
- RECOVER DATAFILE

Note that each of these is also a sub-clause of an ALTER DATABASE statement. Oracle recommends using the SQL*Plus RECOVER command rather than the ALTER DATABASE statement with the RECOVER clause. For more information about the SQL*Plus RECOVER command, see *SQL*Plus User's Guide and Reference*.

Each command uses the same criteria to determine whether files are recoverable. If Oracle cannot get the lock for a file it is attempting to recover, it signals an error. This signal prevents two recovery sessions from recovering the same file and prevents media recovery of a file that is in use. In each case you can recover a database, tablespace, or datafile.

See Also: To learn about the differences between the SQL ALTER DATABASE RECOVER and SQL*Plus RECOVER statements, see "[Using Media Recovery Statements](#)" on page 14-9.

Part II

Using Recovery Manager

Recovery Manager Concepts

This chapter describes the basic concepts for the Oracle Recovery Manager (RMAN) utility, and includes the following topics:

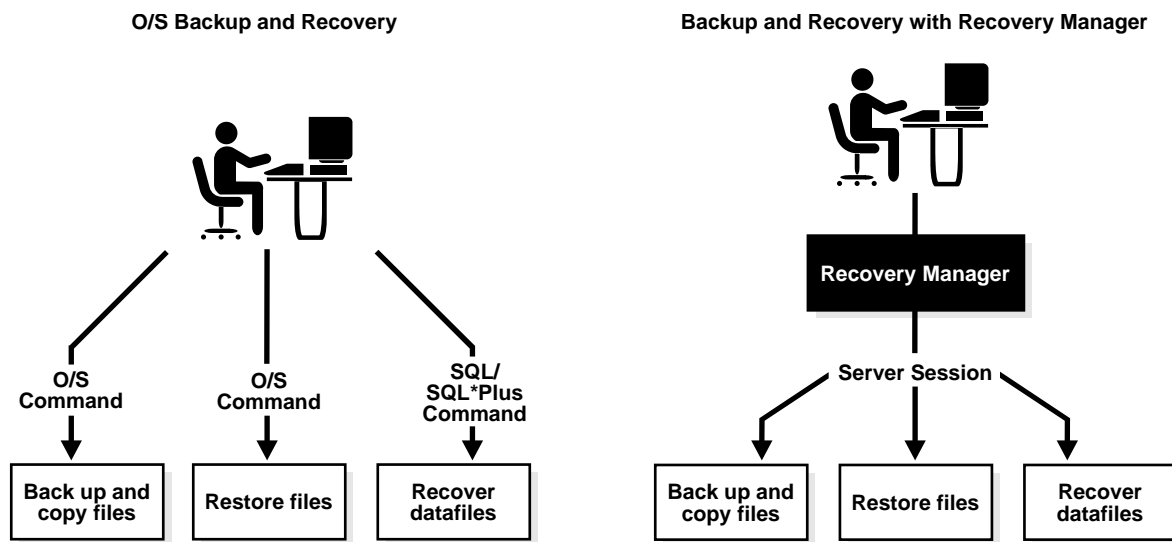
- [Overview of Recovery Manager](#)
- [Recovery Manager Commands](#)
- [Recovery Manager Metadata](#)
- [Media Management](#)
- [Lists and Reports](#)
- [Channel Allocation](#)
- [Backup Sets](#)
- [Backup Types](#)
- [Image Copies](#)
- [Restoring Files](#)
- [Media Recovery](#)
- [Database Duplication](#)
- [Integrity Checks](#)

Overview of Recovery Manager

Recovery Manager (RMAN) is an Oracle tool that allows you to back up, copy, restore, and recover datafiles, control files, and archived redo logs. You can invoke RMAN as a command line utility from the O/S prompt or use the GUI-based Enterprise Manager Backup Manager.

RMAN automates many of the backup and recovery tasks that were formerly performed manually. For example, instead of requiring you to locate appropriate backups for each datafile, copy them to the correct place using O/S commands, and choose which archived logs to apply, RMAN manages these tasks automatically.

Figure 4–1 Comparison of RMAN and O/S Backup and Recovery Procedures



When you start RMAN, the following operations occur:

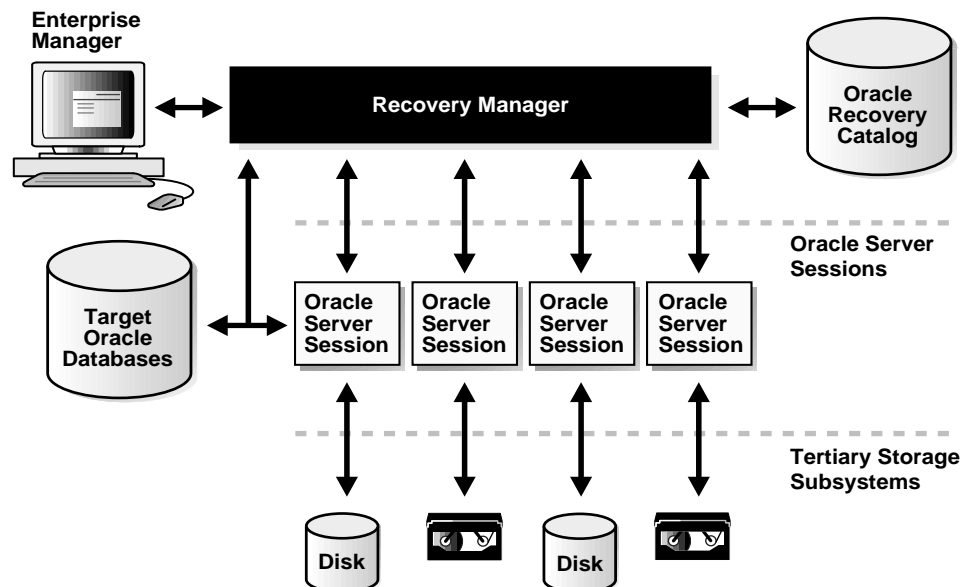
- The RMAN user session starts on the client.
- RMAN creates two default server sessions that connect to the target database. The *target database* is the database that you want to back up or restore.
- If you perform I/O on disk or tape, RMAN requires that you allocate one channel for each disk or device. A channel corresponds to a server session.
- If you connect to a recovery catalog, RMAN creates a server session on the recovery catalog database.

When you use RMAN to connect to your target database, RMAN uses server sessions to perform the backup and recovery operations through a PL/SQL interface. RMAN physically stores its backups and copies on disk or, if you use media management software, on tape.

RMAN stores metadata about its backup and recovery operations in the recovery catalog, which is a centralized repository of information, or exclusively in the control file. Typically, the recovery catalog is stored in a separate database. If you do not use a recovery catalog, RMAN uses the control file as its repository of metadata.

Note: Recovery Manager is only compatible with Oracle databases of release 8.0 or higher. It is *not* compatible with the Oracle7 Enterprise Backup Manager (EBU).

Figure 4–2 Recovery Manager with Optional Recovery Catalog



See Also: For information about RMAN compatibility and upgrade issues, see *Oracle8i Migration*.

Recovery Manager Features

RMAN automates important backup and recovery procedures. For example, Recovery Manager is able to:

- Use server sessions to back up and copy the database, tablespaces, datafiles, control files, and archived redo logs.
- Compress backups of datafiles so that only those data blocks that have been written to are included in a backup.
- Store frequently executed backup and recovery operations in scripts.
- Perform incremental backups, which back up only those data blocks that have changed since a previous backup.
- Create a duplicate of your production database for testing purposes.
- Use third-party media management software.
- Generate a printable message log of all backup and recovery operations.
- Use the recovery catalog to automate both restore and recovery operations.
- Perform automatic parallelization of backup and restore operations.
- Restore a backup using a backup control file and automatically adjust the control file to reflect the structure of the restored datafiles.
- Find datafiles that require a backup based on user-specified limits on the amount of redo that must be applied for recovery.
- Perform crosschecks to determine whether archived materials in the media management catalog are still available.
- Test whether specified backups can be restored.

See Also: For a description of RMAN features that are new to this version of Oracle, see *Getting to Know Oracle8i*.

What Recovery Manager Is Not

RMAN is not:

- The only option for performing backup and recovery operations. You can also back up and restore files using O/S commands, and use SQL/SQL*Plus statements to recover them.
- A recovery catalog. A recovery catalog is an optional schema containing information about RMAN operations. Note also that RMAN does not store

backups or copies in the recovery catalog; rather, it stores information *about* backups and copies in the recovery catalog.

- Compatible with Oracle databases *before* release 8.0.
- The Enterprise Backup Utility (EBU), which is an Oracle7 utility. RMAN is *not* compatible with EBU.
- An O/S-specific backup utility. RMAN is a generic utility that can run on a number of different operating systems.

Recovery Manager Commands

RMAN provides commands that you can use to manage all aspects of backup and recovery operations. Note that all RMAN commands for Oracle release 8.0 work in release 8.1.

This section contains the following topics:

- [Recovery Manager PL/SQL Packages](#)
- [How Recovery Manager Compiles and Executes Commands](#)
- [Types of Recovery Manager Commands](#)
- [User Execution of Recovery Manager Commands](#)
- [Recovery Manager Command Errors](#)

See Also: For a complete account of RMAN commands and their syntax, see [Chapter 11, "Recovery Manager Command Syntax"](#).

Recovery Manager PL/SQL Packages

The RMAN executable uses PL/SQL procedures to interpret commands. The PL/SQL packages perform two main functions:

- Maintain the RMAN metadata in the control file or recovery catalog.
- Communicate with Oracle and the O/S to create, restore, and recover backup sets and image copies.

The DBMS_RCVCAT and DBMS_RCVMAN packages are created by the **create catalog** command. RMAN uses DBMS_RCVCAT to maintain information in the recovery catalog and DBMS_RCVMAN to query the recovery catalog or control file.

The DBMS_BACKUP_RESTORE package is created by the `dbmsbkrs.sql` and `prvrbkrs.plb` scripts. This package is automatically installed in every Oracle

database when the `catproc.sql` script is run. This package interfaces with the Oracle server and the operating system to provide the I/O services for backup and restore operations as directed by Recovery Manager.

See Also: For more information about the `DBMS_RCVCAT`, `DBMS_RCVMAN`, and `DBMS_BACKUP_RESTORE` packages, see the *Oracle8i Supplied Packages Reference*.

How Recovery Manager Compiles and Executes Commands

Recovery Manager processes commands in two phases:

- [Compilation](#)
- [Execution](#)

When you issue a Recovery Manager command such as **backup**, RMAN generates output alerting you to the various phases of command processing. Following is sample output for a **backup tablespace** command. Note the RMAN messages that begin `RMAN-03xxx`:

```
RMAN-03022: compiling command: backup
RMAN-03023: executing command: backup
RMAN-08008: channel chl: starting full datafile backupset
RMAN-08502: set_count=48 set_stamp=346765191 creation_time=15-OCT-98
RMAN-08010: channel chl: specifying datafile(s) in backupset
RMAN-08522: input datafile fno=00017 name=/oracle/dbs/tbs_14.f
RMAN-08522: input datafile fno=00003 name=/oracle/dbs/tbs_11.f
RMAN-08522: input datafile fno=00004 name=/oracle/dbs/tbs_12.f
RMAN-08522: input datafile fno=00007 name=/oracle/dbs/tbs_13.f
RMAN-08013: channel chl: piece 1 created
RMAN-08503: piece handle=/oracle/dbs/lgaamds7_1_1 comment=NONE
RMAN-08525: backup set complete, elapsed time: 00:00:04
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete
RMAN-08031: released channel: chl
```

Compilation

During the compilation phase, RMAN performs two operations:

- Name translation
- Construction of PL/SQL job steps

Name Translation Most RMAN commands require you to identify what object the command should operate on. *Name translation* is the operation whereby RMAN translates what you specified in the command into a list of one or more entities that

it really operates on. An entity in this sense can be a datafile, an archived redo log, a datafile copy, a control file copy, and so forth. For example, when you enter the command **backup database**, RMAN translates the keyword **database** into a list of all the datafiles that are in the database. Similarly, when you issue the following:

```
backup archivelog from time='xxx' until time='yyy'
```

RMAN translates the *archivelogRecordSpecifier* clause (the **from ... until ...** part of the command) into a list of archived redo logs.

Construction of PL/SQL Job Steps RMAN performs its work by dynamically generating one or more PL/SQL programs and then executing them. In essence, RMAN compiles the commands you issue into PL/SQL programs. RMAN itself contains a copy of the PL/SQL engine, and executes the PL/SQL programs internally during the execution phase. These programs make remote procedure calls to the target database to perform backup, restore, and other necessary operations.

A single RMAN command can result in the generation of multiple PL/SQL programs. For example, a **backup database** command generates one PL/SQL program for each backup set that is to be created. Similarly, **restore database** generates one PL/SQL program for each backup set that needs to be restored.

Execution

During the execution phase, RMAN schedules and runs the PL/SQL programs it compiled during the compilation phase. RMAN assigns one PL/SQL program to each channel (that is, each server session) that you have allocated. The channels execute their PL/SQL program concurrently. For example, if you allocate three channels, then RMAN executes three of the PL/SQL programs at the same time.

RMAN is able to execute concurrent PL/SQL programs because the remote procedure calls (RPCs) that the programs make to the target database use the non-blocking User Program Interface (UPI) feature, thereby allowing RMAN to switch to another channel when one channel makes a non-blocking RPC call. RMAN uses an internal polling mechanism to detect when a non-blocking RPC call has completed. After it completes, RMAN resumes the PL/SQL program.

Types of Recovery Manager Commands

RMAN uses two basic types of commands: *stand-alone commands* and *job commands*. With the exception of the **change**, **crosscheck**, and **delete** commands, stand-alone commands are self-contained. In contrast, job commands must appear within the brackets of a **run** command.

After you connect to the target and optional recovery catalog, you will execute most of your RMAN commands within **run**. Following is a typical example of a **run** statement:

```
run {
    allocate channel c1 type 'sbt_tape';
    restore database;
    recover database;
}
```

See Also: See [Chapter 11, "Recovery Manager Command Syntax"](#) for an exhaustive description of RMAN syntax.

Stand-Alone Commands

Unlike job commands, stand-alone commands do not appear as sub-commands within **run**. Following are some of the commands that can appear by themselves:

- **catalog**
- **change**
- **create catalog, drop catalog, upgrade catalog**
- **create script, delete script, replace script**
- **crosscheck**
- **delete expired backupset**
- **list**
- **report**

Some of these commands are not strictly stand-alone, however, since they must be preceded by an **allocate channel for maintenance** command.

See Also: To learn about stand-alone command syntax, see "[rmanCmd](#)" on page 11-121.

Job Commands

Unlike stand-alone commands, job commands must appear within the brackets of a **run** command. Following are examples of job commands:

- **allocate channel**
- **backup**
- **copy**

- **duplicate**
- **recover**
- **restore**
- **switch**

RMAN executes the job commands inside of a **run** command block sequentially. If any command within the block fails, RMAN ceases processing—no further commands within the block are executed. In effect, the **run** command defines a unit of command execution. When the last command within a **run** block completes, Oracle releases any server-side resources such as I/O buffers or I/O slave processes allocated within the block.

See Also: To learn about job command syntax, "[run](#)" on page 11-124.

Command Exceptions

Most commands are either stand-alone commands or job commands. If you try to issue a job command outside of a **run** block or issue a stand-alone command inside a **run** block, RMAN issues a syntax error message. The following exceptions, however, can function as both stand-alone and job commands:

- **@**
- **@@**
- **host**
- **send**
- **shutdown**
- **startup**
- **sql**

Command-Line Arguments

RMAN supports a number of command-line arguments that you can specify when you connect to RMAN. You can specify most of these arguments only on the command line; the exceptions are **target** and **catalog**, which you can specify either on the command line or via the **connect** command after RMAN has started. The **connect** command allows you to avoid putting passwords on the command line, since this practice sometimes poses a security problem.

See Also: To learn about command-line options, see "[cmdLine](#)" on page 11-39.

User Execution of Recovery Manager Commands

RMAN uses a command language interpreter (CLI) that allows you to execute commands in interactive or batch mode. You can also specify the **log** option at the command line to write RMAN output into a log file.

Interactive Mode

To run RMAN commands interactively, start RMAN and then type commands into the command line interface. For example, you can start RMAN from the UNIX command shell and then execute interactive commands as follows:

```
% rman target sys/sys_pwd@prodl catalog rman/rman@rcat
RMAN> run {
2> allocate channel d1 type disk;
3> backup database;
4> }
```

Batch Mode

You can type RMAN commands into a file, and then run the *command file* by specifying its name on the command line. The contents of the command file should be identical to commands entered at the command line.

When running in batch mode, RMAN reads input from a command file and writes output messages to a log file (if specified). RMAN parses the command file in its entirety before compiling or executing any commands. Batch mode is most suitable for performing regularly scheduled backups via an operating system job control facility.

In this example, the RMAN sample script from "[Interactive Mode](#)" on page 4-10 has been placed into a command file called `b_whole_10.rcv`. You can run this file from the O/S command line and write the output into the log file `rman_log.f` as follows:

```
% rman target / catalog rman/rman@rcat @b_whole_10.rcv log rman_log.f
```

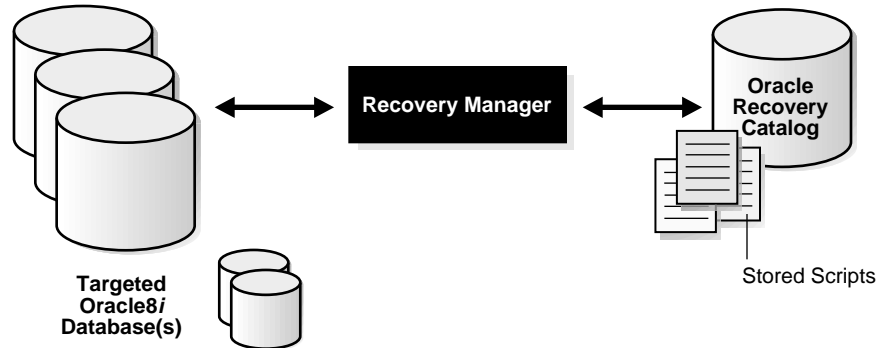
See Also: For more information about RMAN command line options, see "[cmdLine](#)" on page 11-39.

Stored Scripts

A *stored script* is a set of RMAN commands that is enclosed in braces and stored in the recovery catalog. Maintaining a stored script allows you to plan, develop, and test a set of commands for backing up, restoring, or recovering the database. Stored

scripts also minimize the potential for operator errors. Each stored script relates to only one database.

Figure 4–3 Recovery Manager Stored Scripts



To create a stored script, either enter your script interactively into the RMAN command-line interface, or type your RMAN commands into a command file and run the command file.

Following is an example of a stored script:

```
replace script b_whole_10 {
    # back up whole database and archived logs
    allocate channel d1 type disk;
    allocate channel d2 type disk;
    allocate channel d3 type disk;
    backup
        incremental level 0
        tag b_whole_10
        filesperset 6
        format '/dev/backup/prod1/df/df_t%t_s%s_p%p'
        (database);
    sql 'ALTER SYSTEM ARCHIVE LOG CURRENT';
    backup
        filesperset 20
        format '/dev/backup/prod1/al/al_t%t_s%s_p%p'
        (archivelog all
        delete input);
}
```

View your stored scripts by querying the recovery catalog view RC_STORED_SCRIPT:

```
SQL> SELECT * FROM rc_stored_script;
```

DB_KEY	DB_NAME	SCRIPT_NAME
1	RMAN	full_backup
1	RMAN	incr_backup_0
1	RMAN	incr_backup_1
1	RMAN	incr_backup_2
1	RMAN	log_backup

See Also: For more information on scripts, see ["Storing Scripts in the Recovery Catalog"](#) on page 6-20. Also refer to the sample scripts stored in your /demo directory.

Recovery Manager Command Errors

On various occasions it may be important for you to determine whether RMAN successfully executed your command. For example, if you are trying to write a script that will perform an unattended backup using RMAN, you may want to know whether the backup was a success or failure.

The simplest way to determine whether RMAN encountered an error is to examine its return code. RMAN will return 0 to the operating system if no errors occurred, 1 otherwise. For example, if you are running UNIX and using the C shell, RMAN outputs the return code into a shell variable called `$status`.

The second easiest way is to search the Recovery Manager output for the string RMAN-00569, which is the message number for the error stack banner. All RMAN errors are preceded by this error message. If you do not see an RMAN-00569 message in the output, then there are no errors. Following is sample output for a syntax error:

```
RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-00558: error encountered while parsing input commands
RMAN-01005: syntax error: found "}": expecting one of: "archivelog, backup, backupset,
channel, comma, controlfilecopy, current, database, datafile, datafilecopy, delete,
diskratio, filesperset, format, include, (, parms, pool, ;, skip, setsize, tablespace,
tag"
RMAN-01007: at line 1 column 58 file: standard input
```

Recovery Manager Metadata

The RMAN metadata is the information about your target databases that RMAN uses to conduct its backup, recovery, and maintenance operations. You can either create a *recovery catalog* in which to store this information, or let RMAN store it exclusively in the target database control file. Although RMAN can conduct all major backup and recovery operations using just the control file, some RMAN commands function only when you use a recovery catalog (see "[Consequences of Using the Control File for RMAN Metadata](#)" on page 5-6 for a list of the catalog-only commands).

The recovery catalog is maintained solely by RMAN; the target database never accesses it directly. RMAN propagates information about the database structure, archived redo logs, backup sets, and datafile copies into the recovery catalog from the target database's control file.

See Also: To learn how to manage RMAN metadata, see [Chapter 6, "Managing Recovery Manager Metadata"](#).

Storage of Metadata in the Recovery Catalog

The recovery catalog is an optional repository of information about your target databases that RMAN uses and maintains. Conveniently, you do not need an additional database for storing the recovery catalog, since you can put the recovery catalog in an existing database. RMAN uses the information in the recovery catalog, which is obtained from the control file, to determine how to execute requested backup and recovery operations.

The recovery catalog contains information about:

- Datafile and archived redo log backup sets and backup pieces.
- Datafile copies.
- Archived redo logs and their copies.
- Tablespaces and datafiles on the target database.
- Stored scripts, which are named user-created sequences of RMAN and SQL commands.

See Also: For information about recovery catalog compatibility for different releases of the database, see *Oracle8i Migration*.

Resynchronization of the Recovery Catalog

The recovery catalog obtains crucial RMAN metadata from the target database control file. Resynchronization of the recovery catalog ensures that the metadata that RMAN obtains from the control file stays current.

Resynchronizations can be *full* or *partial*. In a partial resynchronization, RMAN reads the current control file to update changed data, but does not resynchronize metadata about the database *physical schema*: datafiles, tablespaces, redo threads, rollback segments (only if the database is open), and online redo logs. In a full resynchronization, RMAN updates all changed records, including schema records.

When you issue certain commands in RMAN, the program automatically detects when it needs to perform a full or partial resynchronization and executes the operation as needed. You can also force a full resynchronization by issuing a **resync catalog** command. It is a good idea to run RMAN once a day or so and issue the **resync catalog** command to ensure that the catalog stays current.

RMAN generates a *snapshot control file*, which is a temporary backup control file, each time it resynchronizes. This snapshot control file ensures that RMAN has a consistent view of the control file either when refreshing the recovery catalog or when querying the control file. Because the snapshot control file is intended for RMAN's short-term use, it is not registered in the recovery catalog. RMAN records the snapshot control file checkpoint in the recovery catalog to indicate how current the recovery catalog is.

The Oracle8i server ensures that only one RMAN session accesses a snapshot control file at any point in time. This safeguard is necessary to ensure that two RMAN sessions do not interfere with each other's use of the snapshot control file.

Note: You can specify the name and location of a snapshot control file. For instructions, see "[Determining the Snapshot Control File Location](#)" on page 5-3.

See Also: To learn how to resynchronize the recovery catalog, see "[Resynchronizing the Recovery Catalog](#)" on page 6-23. For **resync catalog** syntax, see "[resync](#)" on page 11-118.

Backups of the Recovery Catalog

A single recovery catalog is able to store information for multiple target databases. Consequently, loss of the recovery catalog can be disastrous. You should back up the recovery catalog frequently.

Because the recovery catalog resides in an Oracle database, you can use RMAN to back it up by reversing the roles of the recovery catalog database and the target database. In other words, the target database can become the recovery catalog database and the recovery catalog database can be treated as a target database. Several other backup options are available.

If the recovery catalog is destroyed and no backups of it are available, then you can partially reconstruct the catalog from the current control file or control file backups. Nevertheless, you should always aim to have a valid, recent backup of your recovery catalog.

See Also: To learn how to back up the recovery catalog, see "[Backing Up the Recovery Catalog](#)" on page 6-30.

Storage of Metadata Exclusively in the Control File

Because most information in the recovery catalog is also available in the target database's control file, RMAN supports an operational mode in which it uses the target database control file instead of a recovery catalog. This mode is especially appropriate for small databases where installation and administration of another database for the sole purpose of maintaining the recovery catalog is burdensome.

Oracle does not support the following features in this operational mode:

- Stored scripts
- Restore and recovery when the control file is lost or damaged

To restore and recover your database without using a recovery catalog, Oracle recommends that you:

- Use a minimum of two multiplexed or mirrored control files, each on separate disks.
- Keep excellent records of which files were backed up, the date they were backed up, and the names of the backup pieces that each file was written to (see [Chapter 7, "Generating Lists and Reports with Recovery Manager"](#)). Keep all Recovery Manager backup logs.

WARNING: It is difficult to restore and recover if you lose your control files and do not use a recovery catalog. The only way to restore and recover when you have lost all control files and need to restore and recover datafiles is to call Oracle WorldWide Support (WWS). WWS will need to know:

- The current schema of the database.
 - Which files were backed up.
 - When the files were backed up.
 - The names of the backup pieces containing the files.
-
-

See Also: For a complete list of commands that are disabled unless you use a recovery catalog, see ["Consequences of Using the Control File for RMAN Metadata"](#) on page 5-6.

Media Management

To utilize tape storage for your database backups, RMAN requires a [media manager](#). A media manager is a utility that loads, labels, and unloads sequential media such as tape drives for the purpose of backing up and recovering data.

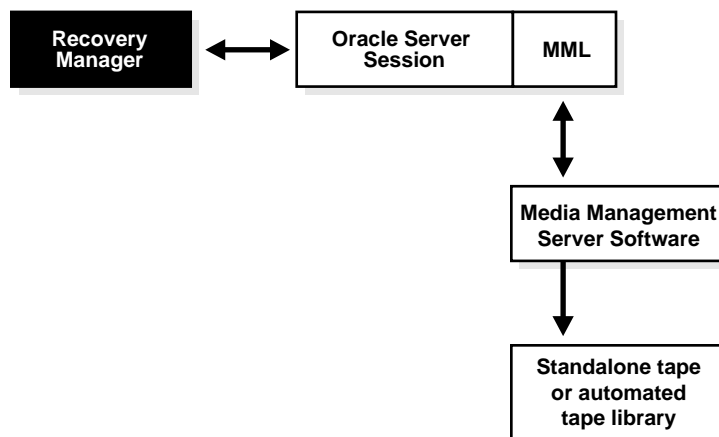
Some media management products can manage the entire data movement between Oracle datafiles and the backup devices. Such products may use technologies such as high-speed connections between storage and media subsystems, which can remove much of the backup load from the primary database server.

To use RMAN to make backups to sequential media such as tape, integrate media management software with your Oracle software. Note that Oracle does not need to connect to the media management library (MML) software when it backs up to disk.

Because RMAN uses the same media management API as the Oracle7 Enterprise Backup Utility (EBU), RMAN is compatible with the same media managers as EBU. Check with your media vendor, however, to determine whether the version of the media software you own has been certified to be compatible with RMAN (see ["Backup Solutions Program"](#) on page 4-20).

Figure 4-4 shows the architecture for a media manager integrated with Oracle.

Figure 4-4 Architecture for MML Integrated with Oracle



The Oracle executable is the same used when a user connects to the database. The MML in the diagram above represents vendor-supplied media management software that can interface with Oracle. Oracle calls MML software routines to back up and restore datafiles to and from media controlled by the media manager.

Backup and Restore Operations Using a Media Manager

The following Recovery Manager script performs a datafile backup to a tape drive controlled by a media manager:

```

run {
  # Allocating a channel of type 'sbt_tape' specifies a media management device
  allocate channel chl type 'sbt_tape';
  backup datafile 10;
}
  
```

When Recovery Manager executes this command, it sends the backup request to the Oracle server session performing the backup. The Oracle server session identifies the output channel as a media management device and requests the media manager to load a tape and write the output.

The media manager labels and keeps track of the tape and names of files on each tape. If your site owns an automated tape library with robotic arm, the media

manager will automatically load and unload the tapes required by Oracle; if not, the media manager will request an operator to load a specified tape into the drive.

The media manager handles restore as well as backup operations. When you restore a file, the following steps occur:

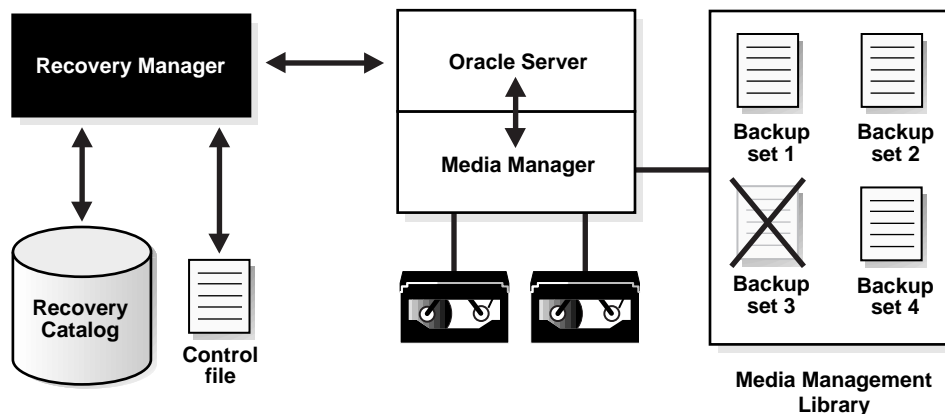
1. Oracle requests the restore of a particular file.
2. The media manager identifies the tape containing the file and reads the tape.
3. The media manager passes the information back to the Oracle server session.
4. The Oracle session writes the file to disk.

Media Manager Crosschecks

Sometimes tapes in the media management library become unavailable. RMAN can perform crosschecks to determine that backup pieces are still available; in this way, the recovery catalog stays synchronized with the media management catalog.

For example, you can issue a **crosscheck backup** command to check all backups on tape. If RMAN cannot find a particular backup, it will change its status to **expired** in the RMAN metadata. Issue a **list** command or access the recovery catalog views to determine the status of backups and copies.

Figure 4-5 Crosschecks



See Also: To learn how to perform crosschecks, see "[Crosschecking RMAN Metadata](#)" on page 6-9. For **crosscheck** and **change ... crosscheck** command syntax, see "[crosscheck](#)" on page 11-57 and "[change](#)" on page 11-35.

Proxy Copy

Oracle has integrated *proxy copy* functionality into its media management API. Vendors can use this API to develop media management software that takes control of backup and restore operations. RMAN provides a list of files requiring backup or recovery to the media manager, which in turn makes all decisions regarding how and when to move the data.

For each file that you attempt to back up using the **backup proxy** command, RMAN queries the media manager to determine whether it can perform proxy copy. If the media manager cannot proxy copy the file, then RMAN uses conventional backup sets to perform the backup. An exception occurs when you use the **proxy only** option, which causes Oracle to issue an error message when it cannot proxy copy.

Oracle records a record of each proxy-copied file in the control file. RMAN uses this information to resynchronize the recovery catalog. Access the `V$PROXY_DATAFILE` and `V$PROXY_ARCHIVEDLOG` dynamic performance views to obtain the proxy copy information. Use the **change ... proxy** command to delete or change the status of a proxy backup.

Note: If a proxy version of RMAN is used with a non-proxy target database, RMAN will *not* use proxy copy to create backup sets. If you make backups using proxy copy and then downgrade Oracle to a non-proxy version, RMAN will not use proxy copy backups when restoring and will issue a warning when the best available file is a proxy copy.

See Also: For more information about `V$PROXY_DATAFILE` and `V$PROXY_ARCHIVEDLOG`, see the *Oracle8i Reference*. For **backup** command syntax, see "**backup**" on page 11-21.

Media Manager Testing

A new client program, `sbttest`, is a stand-alone test of the media management software that is linked with Oracle to perform backups to tape. Use it when Oracle is unable to create or restore backups using either the bundled Legato Storage Manager or another vendor's media management product. Only use the `sbttest` program at the direction of Oracle support.

Backup Solutions Program

The Oracle Backup Solutions Program (BSP) provides a range of media management products that are compliant with Oracle's Media Management Library (MML) specification. Software that is compliant with the MML interface enables an Oracle server session to issue commands to the media manager to back up or restore a file. The media manager responds to the command by loading, labeling, or unloading the requested tape.

One MML-compliant product is the Legato Storage Manager (LSM), which is available for several common platforms. Oracle includes the LSM with software that it ships for these platforms. If your version of the Oracle software includes the LSM, refer to the *Legato Storage Manager Administrator's Guide* to learn about its features. If your shipment includes other BSP media management products, then refer to your platform-specific documentation for information.

Several other products may be available for your platform from media management vendors. For a current list of available products, refer to the BSP web site at:

<http://www.oracle.com/st/products/features/backup.html>

You can also contact your Oracle representative for a complete list.

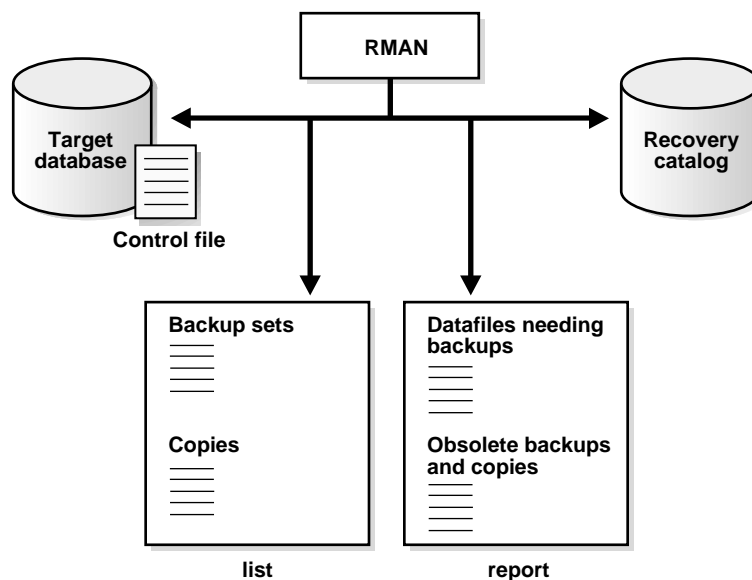
If you wish to use a specific media management product, contact the media management vendor directly to determine whether it is a member of the Oracle BSP. Note that Oracle does not certify media vendors for compatibility with RMAN, so any questions about availability, version compatibility, and functionality should be directed to the media vendor, not Oracle.

Lists and Reports

Use RMAN's **report** and **list** commands to gain information about backups and image copies. RMAN obtains the information from your metadata repository: either the control file or the recovery catalog.

The **list** command lists the contents of RMAN's metadata repository, whereas the **report** command performs a more detailed analysis. RMAN writes the output from these commands to the screen or to a log file.

Figure 4-6 RMAN Lists and Reports



Lists of Backups and Copies

The **list** command queries the recovery catalog or control file and produces a record of its contents. Use it to list:

- Backups of a specified list of datafiles.
- Image copies of a specified list of datafiles.
- Backups of any datafile that is a member of a specified list of tablespaces.
- Image copies of any datafile that is a member of a specified list of tablespaces.
- Backups of any archived redo logs with a specified name and/or within a specified range.
- Image copies of any archived redo log with a specified name and/or within a specified range.
- Incarnations of a specified database.

See Also: To learn how to generate lists of backups and image copies, see "[Generating Lists](#)" on page 7-2. For reference material on the **list** command, see "[list](#)" on page 11-76.

Reports on Backups, Copies, and Database Schema

RMAN reports are intended to provide analysis of your backup and recovery situation. An RMAN report can answer questions such as:

- Which datafiles need a backup?
- Which datafiles have not been backed up recently?
- Which datafiles need to be backed up since fewer than *n* number of backups or copies are available?
- Which backups and copies can be deleted?
- Which datafiles are not recoverable because of unrecoverable operations performed on them?
- What is the current physical schema of the database or what was it at some previous time?
- Which backups are *orphaned*, i.e., unusable in a restore operation, because they belong to incarnations of the database that are not direct predecessors of the current incarnation?

Issue the **report need backup** and **report unrecoverable** commands regularly to ensure that the necessary backups are available to perform media recovery, as well as to ensure that you can perform media recovery within a reasonable amount of time.

The **report** command lists backup sets and datafile copies that can be deleted either because they are redundant or because they are unrecoverable. A datafile is considered *unrecoverable* if an unrecoverable operation has been performed against an object residing in the datafile subsequent to the last backup.

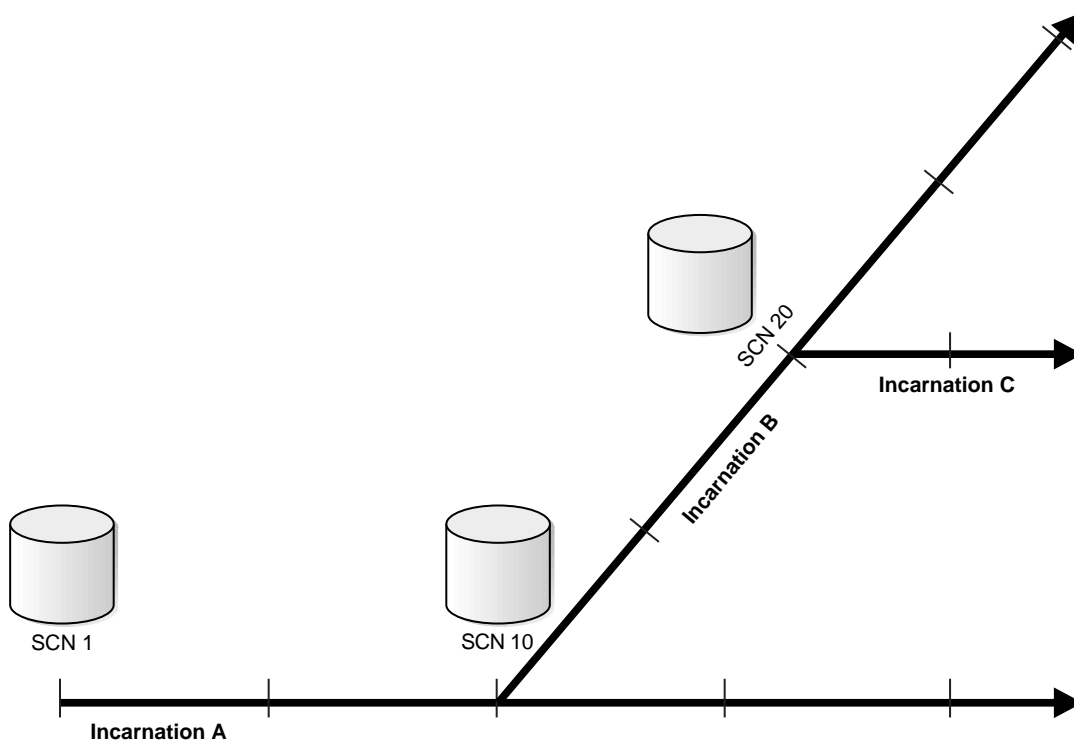
Note: A datafile that does not have a backup is not considered unrecoverable. You can recover such datafiles via the CREATE DATAFILE command provided that redo logs starting from when the file was created still exist.

Reporting on Orphaned Backups

The **report** command allows you to list *orphaned backups*. Orphaned backups are backups that are unusable because they belong to incarnations of the database that are not direct ancestors of the current incarnation.

For example, see the incarnation scenario depicted in [Figure 4-7](#):

Figure 4-7 Orphaned Backups



Incarnation A of the database started at SCN 1. At SCN 10, assume that you performed a RESETLOGS operation and created incarnation B. At SCN 20, you performed another RESETLOGS operation on incarnation B and created a new incarnation C.

The following table explains which backups are orphans depending on which incarnation is current:

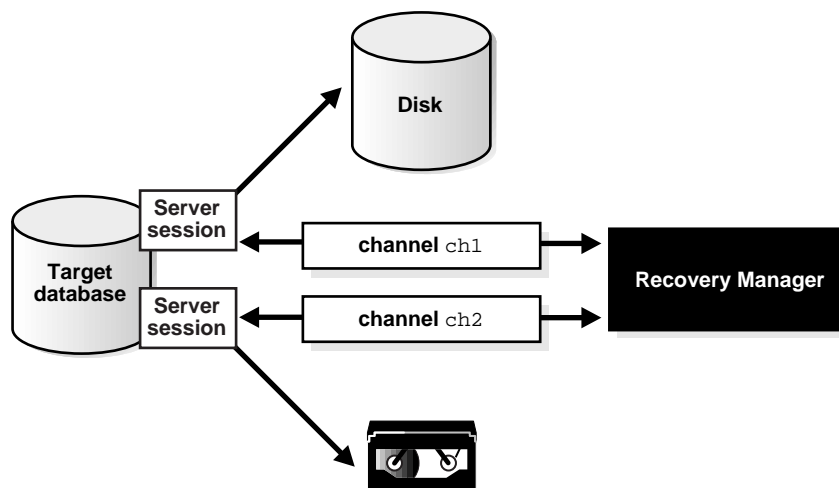
Current Incarnation	Orphaned Backups	Usable Backups (Non-Orphaned)
Incarnation A	All backups from incarnations B and C.	All backups from incarnation A.
Incarnation B	Backups from incarnation A after SCN 10. All backups from incarnation C.	Backups from incarnation A prior to SCN 10. All backups from incarnation B.
Incarnation C	All backups from incarnation A after SCN 10. All backups from incarnation B after SCN 10.	All backups from incarnation A prior to SCN 10. All backups from incarnation B prior to SCN 20. All backups from incarnation C.

See Also: To learn how to generate reports, see "[Generating Reports](#)" on page 7-5. For reference material on the **report** command, see "[report](#)" on page 11-102.

Channel Allocation

You must allocate a *channel* before you execute backup and recovery commands. Each allocated channel establishes a connection from RMAN to a target or auxiliary database (either a database created with the **duplicate** command or a temporary database used in TSPITR) instance by starting a server session on the instance. This server session performs the backup and recovery operations. Only one RMAN session communicates with the allocated server sessions.

Figure 4–8 Channel Allocation



The **allocate channel** command (executed within a **run** command) and **allocate channel for maintenance** command (executed at the RMAN prompt) specify the type of I/O device that the server session will use to perform the backup, restore, or maintenance operation. Each channel usually corresponds to one output device, unless your media management library is capable of hardware multiplexing.

WARNING: Oracle does not recommend hardware multiplexing of Oracle backups.

See Also: For reference material on the **allocate channel** command, see ["allocate"](#) on page 11-9. For reference material on the **allocate channel for maintenance** command, see ["allocateForMaint"](#) on page 11-13.

Channel Control Options

Use channel control commands to:

- Control the O/S resources RMAN uses when performing **backup**, **copy**, **restore**, and **recover** operations.
- Affect the degree of parallelism for a backup (in conjunction with the **filesperset** parameter of the **backup** command).

- Specify limits on I/O bandwidth consumption (**set limit channel ... readrate**).
- Specify limits on the size of backup pieces (**set limit channel ... kbytes**).
- Specify limits on the number of concurrently open files (**set limit channel ... maxopenfiles**).
- Send vendor-specific commands to the media manager (**send**).

On some platforms, these commands specify the name or type of an I/O device to use. On other platforms, they specify which O/S access method or I/O driver to use. Not all platforms support the selection of I/O devices through this interface; on some platforms, I/O device selection is controlled through platform-specific mechanisms.

Whether the **allocate channel** command actually causes your O/S to allocate resources is O/S-dependent. Some operating systems allocate resources when you issue the command; others do not allocate resources until you open a file for reading or writing.

Note: When you specify **type disk**, no O/S resources are allocated other than for the creation of the server session.

You must allocate a maintenance channel before issuing a **change ... delete** command, which calls the O/S to delete a file, or a **change ... crosscheck** command. A maintenance channel is useful only for a maintenance task; you cannot use it as an input or output channel for a backup or restore job. You can only allocate one maintenance channel at a time.

See Also: For reference material on the **set** command, see ["set"](#) on page 11-129. For reference material on the **allocate channel for maintenance** command, see ["allocateForMaint"](#) on page 11-13.

Channel Parallelization

You can allocate multiple channels, thus allowing a single RMAN command to read or write multiple backups or image copies in parallel. Thus, the number of channels that you allocate affects the degree of parallelism within a command. When backing up to tape you should allocate one channel for each physical device, but when backing up to disk you can allocate as many channels as necessary for maximum throughput.

Each **allocate channel** or **allocate auxiliary channel** command uses a separate connection to the target or auxiliary database. You can specify a different connect string for each channel to connect to different instances of the target database, which is useful in an Oracle Parallel Server (OPS) configuration for distributing the workload across different nodes.

Factors Affecting Degrees of Parallelization

RMAN internally handles parallelization of **backup**, **copy**, and **restore** commands. You only need to specify:

- More than one **allocate channel** command.
- The objects that you want to back up, copy, or restore.

RMAN executes commands serially; that is, it completes the current command before starting the next one. Parallelism is exploited only within the context of a single command. Consequently, if you want 5 datafile copies, issue a single **copy** command specifying all 5 copies rather than 5 separate **copy** commands.

The following RMAN script uses serialization to create the file copies: 5 separate **copy** commands are used to back up the files. Only one channel is active at any one time.

```
run {
  allocate channel c1 type disk;
  allocate channel c2 type disk;
  allocate channel c3 type disk;
  allocate channel c4 type disk;
  allocate channel c5 type disk;
  copy datafile 22 to '/dev/prod/backup1/prod_tab5_1.dbf';
  copy datafile 23 to '/dev/prod/backup1/prod_tab5_2.dbf';
  copy datafile 24 to '/dev/prod/backup1/prod_tab5_3.dbf';
  copy datafile 25 to '/dev/prod/backup1/prod_tab5_4.dbf';
  copy datafile 26 to '/dev/prod/backup1/prod_tab6_1.dbf';
}
```

The following statement uses *parallelization* on the same example; one RMAN **copy** command copies 5 files, with 5 channels available. All 5 channels are *concurrently active*—each channel copies one file.

```
run {
  allocate channel c1 type disk;
  allocate channel c2 type disk;
  allocate channel c3 type disk;
  allocate channel c4 type disk;
  allocate channel c5 type disk;
  copy datafile 5 to '/dev/prod/backup1/prod_tab5_1.dbf',
```

```

datafile 23 to '/dev/prod/backup1/prod_tab5_2.dbf',
datafile 24 to '/dev/prod/backup1/prod_tab5_3.dbf',
datafile 25 to '/dev/prod/backup1/prod_tab5_4.dbf',
datafile 26 to '/dev/prod/backup1/prod_tab6_1.dbf';
}

```

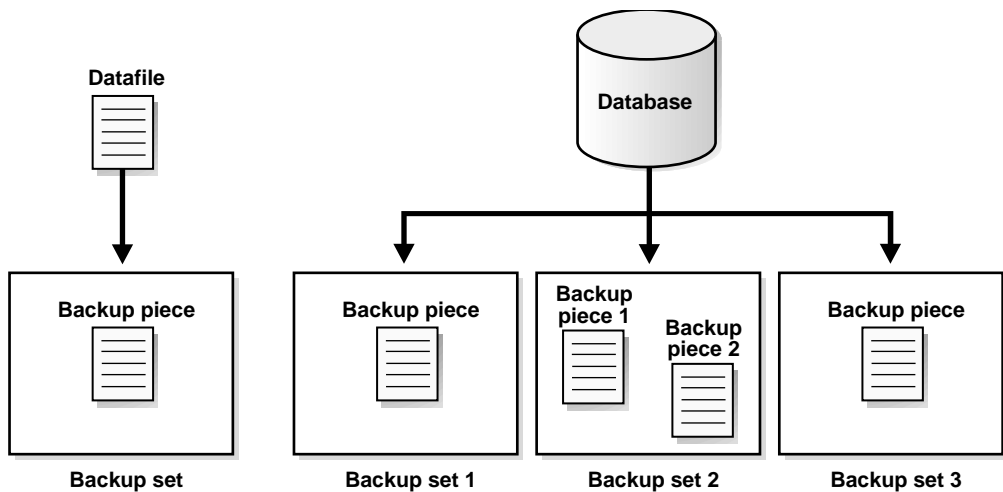
See Also: For information about parallelization in an OPS configuration, see *Oracle8i Parallel Server Concepts and Administration*.

Backup Sets

When you execute the **backup** command, you create one or more backup sets. A *backup set*, which is a logical construction, contains one or more physical *backup pieces*. Backup pieces are O/S files that contain the backed up datafiles, control files, or archived redo logs. You cannot split a file across different backup sets or mix archived redo logs and datafiles into a single backup set.

A backup set is a complete set of backup pieces that constitute a full or incremental backup of the objects specified in the **backup** command. Backup sets are in an RMAN-specific format; image copies, in contrast, are available for use without additional processing.

Figure 4–9 Backup Sets Contain One or More Backup Pieces



When backing up files, the target database must be mounted or open. If the database is mounted and was not shut down abnormally prior to mounting, then RMAN produces a consistent backup. The control file must be current.

If the database is in ARCHIVELOG mode, then the target database can be open or closed; you do not need to close the database cleanly (although Oracle recommends you do so that the backup is consistent). If the database is in NOARCHIVELOG mode, then you must close it cleanly prior to taking a backup.

Note: You cannot make a backup of a plugged-in tablespace until after it has been specified read-write.

This section contains the following topics:

- [Storage of Backup Sets](#)
- [Backup Set Compression](#)
- [Filenames for Backup Pieces](#)
- [Size of Backup Pieces](#)
- [Number of Backup Sets](#)
- [Multiplexed Backup Sets](#)
- [Duplexed Backup Sets](#)
- [Parallelization of Backups](#)
- [Backup Errors](#)

See Also: To learn how to make backups, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#). For information on the **backup** command, see "backup" on page 11-21.

Storage of Backup Sets

RMAN can create backup sets that are written to disk or tertiary storage. If you specify **type disk**, then you must back up to random-access disks. You can make a backup on any device that can store an Oracle datafile: in other words, if the statement `CREATE TABLESPACE tablespace_name DATAFILE 'filename'` works, then '*filename*' is a valid backup pathname.

Using a sequential output device or media management system that is available and supported by Oracle on your operating system, you can write backup sets to

sequential output media such as magnetic tape. If you specify **type** *'sbt_tape'*, then you can back up to any media supported by the media management software.

Note that you cannot archive directly to tape, but RMAN does allow you to back up archived redo logs from disk to tape. If you specify the **delete input** option, RMAN deletes the file after backing it up. RMAN automatically stages the required archived logs from tape to disk during recovery.

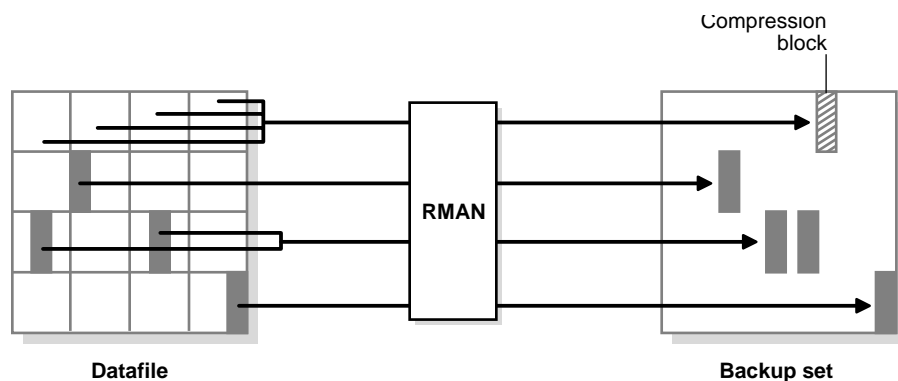
Backup Set Compression

RMAN performs *compression* on its backups, which means that the server session does not write datafile blocks that have never been used. Image copies of a datafile, however, always contain all datafile blocks.

Data blocks in the datafile are grouped into buffers. When RMAN encounters a used data block in a buffer, it writes only the used block to the backup set. When RMAN encounters four contiguous buffers of *unused* input blocks, it writes one *compression block* (of size `DB_BLOCK_SIZE`) to the backup set.

Use the `DB_FILE_DIRECT_IO_COUNT` initialization parameter to set the size of the buffer. For example, set the parameter to a value of 64K. In this case, RMAN writes one compression block for each 256K of contiguous unused input blocks in the input file.

Figure 4–10 Backup Set Compression



Filenames for Backup Pieces

You can either let RMAN determine a unique name for the backup piece or use the **format** parameter to specify a name. If you do not specify a filename, RMAN uses the %U substitution variable to guarantee a unique name. The **backup** command provides substitution variables that allow you to generate unique filenames.

See Also: For reference material on the **format** parameter and the substitution variables of the **backup** command, see "[backup](#)" on page 11-21.

Size of Backup Pieces

Each backup set contains at least one backup piece. If you are writing to disk, use the **setsize** operand of the **backup** command to restrict the size of a backup piece to the maximum file size supported by your media manager or O/S. If you do not restrict this size, Oracle will generate a backup set containing only one file.

RMAN writes the pieces of a backup set serially; striping a backup set across multiple output devices is not supported. If multiple output devices are available, you can partition your backups so that Oracle creates multiple backup sets in parallel. RMAN can perform this backup partitioning automatically.

See Also: For information on the **setsize** parameter, see "[backup](#)" on page 11-21.

Number of Backup Sets

Use the *backupSpec* clause to list what you want to back up as well as specify other useful options. The number of backup sets that RMAN produces depends on:

- The number of *backupSpec* clauses that you specify.
- The number of input files specified or implied in each *backupSpec* clause.
- The **filesperset** parameter, which limits the number of files for a backup set.

Each *backupSpec* clause produces at least one backup set. If the number of input files specified or implied in a *backupSpec* clause exceeds the **filesperset** limit, then RMAN produces multiple backup sets. If you do not specify a **filesperset** limit, then each *backupSpec* produces exactly one backup set.

For datafile or datafile copy backups, group multiple datafiles into a single backup set to the extent necessary to keep an output tape device streaming, or to prevent the backup from consuming too much bandwidth from a particular datafile.

The fewer the files that are contained in a backup set, the faster one of them can be restored, since there is less data belonging to other datafiles that must be skipped.

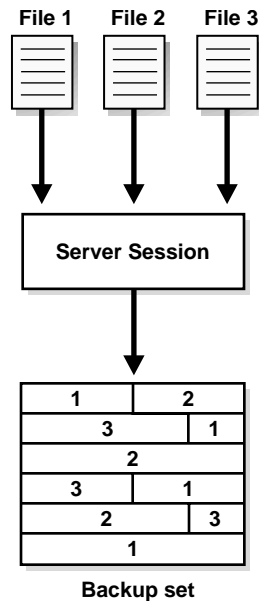
For backup sets containing archived logs, group logs from the same time period into a backup set because they will probably need to be restored at the same time.

See Also: To learn about the *backupSpec* clause, see "backup" on page 11-21.

Multiplexed Backup Sets

Oracle takes the datafile blocks included in the same backup set and *multiplexes* them, which means that the data blocks from all of the datafiles in the backup set are interspersed with one another. As Figure 4-11 illustrates, RMAN can back up three datafiles into a backup set that contains only one backup piece; this backup piece is constituted by the intermingled block components of the three input files.

Figure 4-11 Datafile Multiplexing



Use the **filesperset** parameter to control the number of datafiles that Oracle backs up concurrently to a backup set. Controlling concurrency is helpful when you want to keep a tape device streaming without saturating a single datafile with too many read requests, since these requests can subsequently degrade online performance. Limit the read rate by using the **readrate** option of the **set limit channel** command.

When multiplexing files, you can:

- Partition the datafiles into backup sets explicitly, or let RMAN automatically select a partitioning.
- Keep a high performance sequential output device streaming by including a sufficient number of datafiles in the backup. Keeping the device streaming is important for open database backups in which the backup operation must compete with the online system for I/O bandwidth.
- Include the control file in a datafile backup set. In this case, the control file is written first and its blocks are not multiplexed with datafile blocks.
- Create a backup set containing either datafiles or archived redo logs, but not both together. You cannot write datafiles and archived redo logs to the same backup set because the Oracle logical block size of the objects in a multiplexed backup must be the same.

See Also: To learn how to tune backup performance, see *Oracle8i Tuning*. To learn how to multiplex backups, see "[Multiplexing Datafiles in a Backup](#)" on page 8-18.

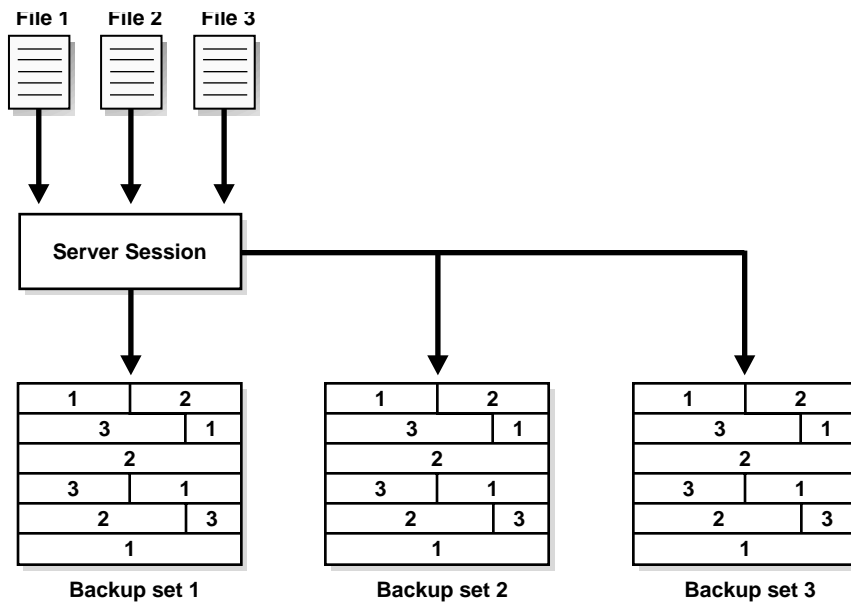
For reference material on the **filesperset** parameter of the **backup** command, see "[backup](#)" on page 11-21. For reference material on the **set** option of the **run** command, see "[set_run_option](#)" on page 11-133.

Duplexed Backup Sets

RMAN provides an efficient way to produce multiple copies of an archived redo log or datafile backup set. Create up to four identical copies of a backup set by issuing the **set duplex** command.

You should not confuse multiplexing, which is combining multiple input files into a single backup set, with duplexing, which is the output of two or more identical backup sets. RMAN allows you to multiplex your input files and duplex the output.

Figure 4–12 Duplexing Multiplexed Backup Sets



See Also: To learn how to duplex backups, see ["Duplexing Backup Sets"](#) on page 8-22. For reference material on the `set duplex` command, see ["set_run_option"](#) on page 11-133.

Parallelization of Backups

If you want to create multiple backup sets and allocate multiple channels, then RMAN automatically parallelizes its operation and writes multiple backup sets in parallel. The allocated server sessions divide up the work of backing up the specified files.

Note: You cannot stripe a single backup set across multiple channels.

RMAN automatically assigns a backup set to a device. You can specify that Oracle should write all backup sets for a *backupSpec* to a specific channel using the `channel` parameter, as in the following example:

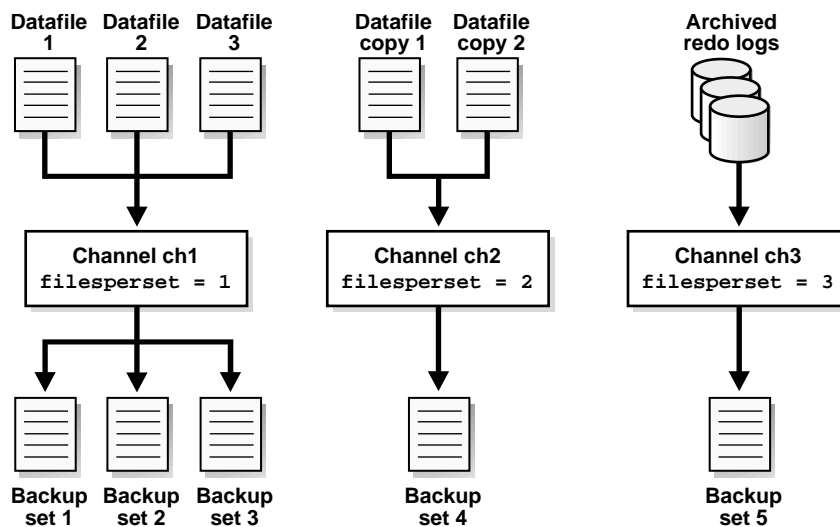

```

run {
  allocate channel ch1 type disk;
  allocate channel ch2 type disk;
  allocate channel ch3 type disk;
  backup
    (datafile 1,2,3 filesperset = 1
     channel ch1)
    (controlfilecopy '/oracle/copy/cf.f' filesperset = 1
     channel ch2)
    (archivelog all filesperset = 3
     channel ch3);
}

```

Table 4–13 show an example of parallelization in which channel 1 backs up datafiles, channel 2 backs up datafile copies, and channel 3 backs up archived redo logs.

Figure 4–13 *Parallelization of Backups*



See Also: For an overview of how allocated channels affect parallelization, see "[Channel Parallelization](#)" on page 4-26. To learn how to parallelize backups, see "[Parallelizing Backup Sets](#)" on page 8-23. For reference material on the **channel** parameter of the **backup** command, see "[backup](#)" on page 11-21.

Backup Errors

RMAN is equipped to handle the two primary types of backup errors: I/O errors and corrupt blocks. Any I/O errors that RMAN encounters when reading files or writing to the backup pieces cause the system to abort the jobs. RMAN will need to rewrite the backup sets that it was writing at the time of the error, but it retains any backup sets that it successfully wrote prior to the abort.

RMAN copies datafile blocks that are already identified as corrupt into the backup. If RMAN encounters datafile blocks that have not already been identified as corrupt, it writes them to the backup with a reformatted header indicating that the block has media corruption (assuming that **set maxcorrupt** is not equal to 0 for this datafile and the number of corruptions does not exceed the limit). In either case, Oracle records the address of the corrupt block and the type of corruption in the control file. Access these control file records through the V\$BACKUP_CORRUPTION view.

Use the **set maxcorrupt** command to limit the number of previously undetected block corruptions that Oracle allows in a specified datafile or list of datafiles. If a **backup** or **copy** command detects more than this number of corruptions, then the command aborts. The default limit is zero, meaning that RMAN will not tolerate any corrupt blocks.

See Also: For more information about fractured and corrupt blocks, see ["Integrity Checks"](#) on page 4-54. For more information on V\$BACKUP_CORRUPTION, see the *Oracle8i Reference*. For reference information on **set maxcorrupt**, see ["set_run_option"](#) on page 11-133.

Backup Types

Recovery Manager allows you to control the type of backups you produce. RMAN backups can be classified in these ways:

- Full or incremental
- Open or closed
- Consistent or inconsistent

Backup Type	Definition
Full	A backup that is non-incremental, i.e., it backs up all used data blocks in the datafiles. Note: A full backup is different from a whole database backup, which is a backup of all datafiles and the current control file.
Incremental	A backup of datafiles that includes only the blocks that have changed since a previous incremental backup. Incremental backups require an incremental level 0 backup to serve as a basis. Full backups cannot be included in an incremental strategy, although an RMAN copy made with the level parameter can be included.
Open	A backup of any part of the target database when it is open. Note: Do not put a tablespace in hot backup mode with the ALTER TABLESPACE BEGIN BACKUP statement. RMAN uses a different method to guarantee consistency in hot backups.
Closed	A backup of any part of the target database when it is mounted but not open. Closed backups can be consistent or inconsistent. Note: If you use a recovery catalog, the catalog database must be open.
Consistent	A backup taken when the database is mounted (but not open) and was <i>not</i> crashed or shut down with the ABORT option prior to mounting. The checkpoint SCNs in the datafile headers match the header information in the control file and none of the datafiles has changes beyond its checkpoint. Consistent backups can be restored without recovery.
Inconsistent	A backup of any part of the target database when: <ul style="list-style-type: none"> ■ It is open. ■ It crashed prior to mounting. ■ It was shut down with the ABORT option prior to mounting. An inconsistent backup requires recovery to become consistent.

Full Backups

A *full backup* reads the entire file and copies all blocks into the backup set, skipping only datafile blocks that have never been used. The server session does not skip blocks when backing up archived redo logs or control files.

Note: A full backup is not the same as a whole database backup; *full* is an indicator that the backup is not incremental.

A full backup has no effect on subsequent incremental backups, which is why it is not considered part of the incremental strategy. In other words, a full backup does not affect which blocks are included in subsequent incremental backups.

Oracle allows you to create and restore full backups of the following:

- Datafiles
- Datafile copies
- Tablespaces
- Control files (current or backup)
- Database (all datafiles and current control file)

Note that backup sets containing archived redo logs are always full backups.

Incremental Backups

An incremental backup reads the entire file and then backs up only those data blocks that have changed since a previous backup. Oracle allows you to create and restore incremental backups of datafiles, tablespaces, or the whole database. Note that RMAN can include a control file in an incremental backup set, but the control file is always included in its entirety—no blocks are skipped.

The primary reasons for making an incremental backup are:

- To save tape using a media manager.
- To save network bandwidth when backing up over a network.
- When the aggregate tape bandwidth available for tape write I/Os is much less than the aggregate disk bandwidth for disk read I/Os.

If none of these criteria apply, then full backups are usually preferable because the application of the incremental backup increases recovery time while the cost savings is negligible.

This section contains the following topics:

- [Multi-Level Incremental Backups](#)
- [How Incremental Backups Work](#)

- [Differential Incremental Backups](#)
- [Cumulative Incremental Backups](#)
- [Incremental Backup Strategy](#)

Multi-Level Incremental Backups

RMAN allows you to create *multi-level incremental backups*. Each incremental level is denoted by an integer, e.g., 0, 1, 2, etc. A level 0 incremental backup, which is the base for subsequent incremental backups, copies all blocks containing data. When you generate a level n incremental backup in which n is greater than 0, you back up:

- All blocks that have changed since the most recent backup at level n or lower (the default type of incremental backup, which is called a *differential backup*)
- All blocks used since the most recent backup at level $n-1$ or lower (called a *cumulative backup*)

The benefit of performing multi-level incremental backups is that you do not back up all of the blocks all of the time. Since RMAN needs to read all of the blocks of the datafile, full backups and incremental backups taking approximately the same amount of time (assuming that the output of the backup is not a bottleneck).

Incremental backups at levels greater than 0 only copy blocks that were modified. The size of the backup file depends solely upon the number of blocks modified and the incremental backup level.

Note: In most circumstances, cumulative backups are preferable to differential backups.

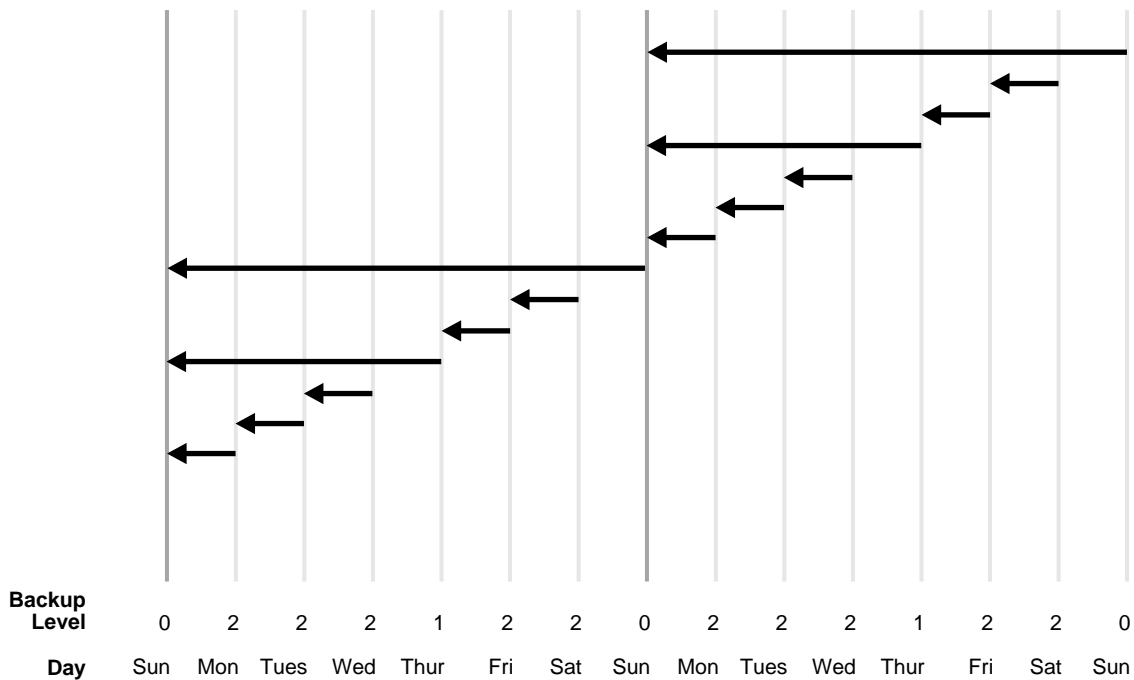
How Incremental Backups Work

Each data block in a datafile contains an SCN, which is the SCN at which the last change was made to the block. During an incremental backup, RMAN reads the SCN of each data block in the input file and compares it to the checkpoint SCN of the parent incremental backup. The parent backup is the backup that RMAN uses for comparing the SCNs, so the parent can be a level 0 backup or, depending on the incremental level, a level 1 or level 2 backup. If the SCN in the input data block is greater than the checkpoint SCN of the parent, then RMAN copies the block.

Differential Incremental Backups

In a differential level n incremental backup, RMAN backs up all blocks that have changed since the most recent backup at level n or lower. For example, in a differential level 2 backup, RMAN determines which level 1 or level 2 backup occurred most recently and backs up all blocks modified since that backup. If no level 1 is available, RMAN copies all blocks changed since the base level 0 backup. Incremental backups are differential by default.

Figure 4–14 Differential Incremental Backups (Default)



In the example above:

- Sunday

An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.

- Monday

A differential incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the most recent incremental backup at level 2 or less is the level 0 Sunday backup, so only the blocks changed since Sunday will be backed up.
- Tuesday

A differential incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the most recent incremental backup at level 2 or less is the level 2 Monday backup, so only the blocks changed since Monday will be backed up.
- Wednesday

An incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the most recent incremental backup at level 2 or less is the level 2 Tuesday backup, so only the blocks changed since Tuesday will be backed up.
- Thursday

An incremental level 1 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the most recent incremental backup at level 1 or less is the level 0 Sunday backup, so only the blocks changed since the Sunday level 0 backup will be backed up.
- Friday

An incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the *most recent* incremental backup at level 2 or less is the level 1 Thursday backup, so only the blocks changed since the Thursday level 1 backup will be backed up.
- Saturday

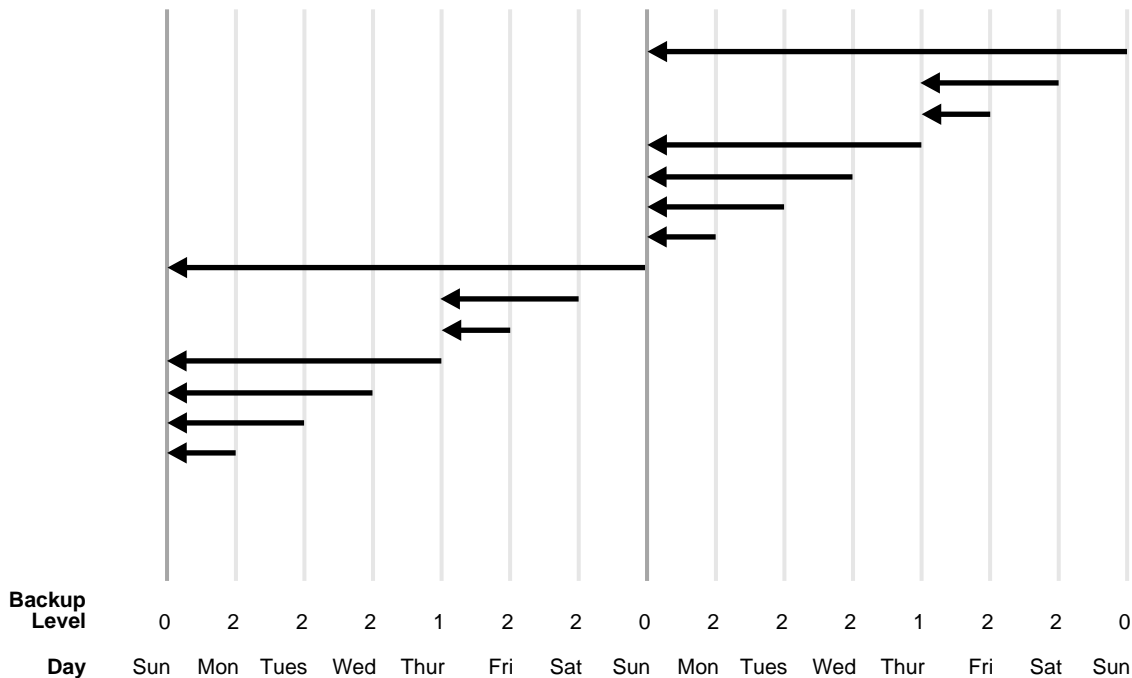
An incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level n or less; in this case, the *most recent* incremental backup at level 2 or less is the level 2 Friday backup, so only the blocks changed since the Friday level 2 backup will be backed up.
- The cycle is repeated.

Cumulative Incremental Backups

Oracle provides an option to make cumulative incremental backups at level 1 or greater. In a cumulative level n backup, RMAN backs up all the blocks used since the most recent backup at level $n-1$ or lower. For example, in a cumulative level 2 backup, RMAN determines which level 1 backup occurred most recently and copies all blocks changed since that backup. If no level 1 backup is available, RMAN copies all blocks changed since the base level 0 backup.

Cumulative incremental backups reduce the work needed for a restore by ensuring that you only need one incremental backup from any particular level. Cumulative backups require more space and time than differential backups, however, because they duplicate the work done by previous backups at the same level.

Figure 4–15 Cumulative Incremental Backups



In the example above:

- Sunday
An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.
- Monday
A cumulative incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 2-1 or less is the level 0 Sunday backup, so only the blocks changed since Sunday will be backed up.
- Tuesday
A cumulative incremental level 2 backup occurs. This backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 2-1 or less is the level 0 Sunday backup, so all the blocks changed since Sunday will be backed up. (This backup includes those blocks that were copied on Monday, since this backup is cumulative and includes the blocks copied at backups taken at the same incremental level as the current backup).
- Wednesday
A cumulative incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 2-1 or less is the level 0 Sunday backup, so all the blocks changed since Sunday will be backed up. (This backup includes those which were copied on Monday and Tuesday, since this backup is cumulative and includes the blocks copied at backups taken at the same incremental level as the current backup).
- Thursday
A cumulative incremental level 1 backup backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 1-1 or less is the level 0 Sunday backup, so all the blocks changed since Sunday will be backed up.
- Friday
A cumulative incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 2-1 or less is the level 1 Thursday backup, so all the blocks changed since Thursday will be backed up.

- Saturday

A cumulative incremental level 2 backup backs up all blocks that have changed since the most recent incremental backup at level $n-1$ or less; in this case, the most recent incremental backup at level 2-1 or less is the level 1 Thursday backup, so all the blocks changed since Thursday will be backed up.

- The cycle is repeated.

Incremental Backup Strategy

Choose your backup scheme according to an acceptable MTTR (mean time to recover). For example, you can implement a three-level backup scheme so that a level 0 backup is taken monthly, a cumulative level 1 backup is taken weekly, and a cumulative level 2 is taken daily. In this scheme, you will never have to apply more than a day's worth of redo for complete recovery.

When deciding how often to take level 0 backups, a good rule of thumb is to take a new level 0 whenever 50% or more of the data has changed. If the rate of change to your database is predictable, then you can observe the size of your incremental backups to determine when a new level 0 is appropriate. The following query displays the number of blocks written to a backup set for each datafile with at least 50% of its blocks backed up:

```
SELECT file#, incremental_level, completion_time, blocks, datafile_blocks
FROM v$backup_datafile
WHERE incremental_level > 0 AND blocks / datafile_blocks > .5
ORDER BY completion_time;
```

Compare the number of blocks in your differential or cumulative backups to your base level 0 backup. For example, if you only create level 1 cumulative backups, then when the most recent level 1 backup is about half of the size of the base level 0 backup, take a new level 0.

Backup Constraints

RMAN performs backup operations only when an instance has the database mounted or open. In an Oracle parallel server environment, if the instance where the backup operation is being performed does not have the database open, then the database must not be open by any instance.

RMAN supports tablespace, datafile, archived redo log, and control file backups. It does not back up:

- Parameter files

- Password files
- Operating system files
- Online redo logs
- Transported tablespaces (before they have been specified read-write)

See Also: For more information about backup constraints in a parallel server environment, see *Oracle8i Parallel Server Concepts and Administration*.

Image Copies

An *image copy* contains a single datafile, archived redo log file, or control file that you can use as-is to perform recovery. Use the RMAN **copy** command or an O/S command such as the UNIX **cp** command to create image copies.

An image copy produced with the RMAN **copy** command is similar to an O/S copy of a single file, except that an Oracle server session produces it. The server session performs additional actions like validating the blocks in the file and registering the copy in the control file. An image copy differs from a backup set because it is not multiplexed, nor is there any additional header or footer control information stored in the copy. RMAN only writes image copies to disk.

RMAN Image Copies

Use the RMAN **copy** command to create an image copy. If the original file needs to be replaced, and if the image copy is of a datafile, then you do not need to restore the copy. Instead, Oracle provides a **switch** command to point the control file at the copy and update the recovery catalog to indicate that the copy has been switched. Issuing the **switch** command in this case is equivalent to issuing the SQL statement ALTER DATABASE RENAME DATAFILE. You can then perform media recovery to make the copy current.

RMAN can catalog an image copy and read the metadata. This operation is important when the recovery catalog is lost and you must perform disaster recovery. Only image copies and archived logs can be cataloged.

O/S Image Copies

Oracle supports image copies created by mechanisms other than RMAN, also known as *O/S copies*. For example, a copy of a datafile that you make with the UNIX **cp** command is an O/S copy. You must catalog such O/S copies with RMAN before using them with the **restore** or **switch** commands.

You can create an O/S copy when the database is open or closed. If the database is open and the datafile is not offline normal, then you must place the tablespace in *hot backup mode*, i.e., issue the SQL statement ALTER TABLESPACE BEGIN BACKUP before creating the copy.

WARNING: If you do not put a tablespace in hot backup mode before making an online backup, Oracle can generate fractured blocks. See "[Detection of Logical Block Corruption](#)" on page 4-55.

Some sites store their datafiles on mirrored disk volumes, which permits the creation of image copies by breaking the mirrors. After you have broken the mirror, you can notify RMAN of the existence of a new O/S copy, thus making it a candidate for use in a restore operation. You must notify RMAN when the copy is no longer available for restore, however, by using the **change ... uncatalog** command. In this example, if the mirror is resilvered (not including other copies of the broken mirror), you must use a **change ... uncatalog** command to update the recovery catalog and indicate that this copy is no longer available.

See Also: To learn how to catalog copies, see "[How Do You Catalog an O/S Backup?](#)" on page 6-29. For reference material on the **change** command, see "[change](#)" on page 11-35.

Tags for Backups and Image Copies

You can assign a user-specified character string called a *tag* to backup sets and image copies (either RMAN-created copies or O/S-created copies). A tag is a symbolic name for a backup set or file copy such as *weekly_backup*; you can specify the tag rather than the filename when executing the **restore** or **change** command. The maximum length of a tag is 30 characters.

Tags do not need to be unique: multiple backup sets or image copies can have the same tag. When a tag is not unique, then with respect to a given datafile, the tag refers to the most current suitable file. By default, Recovery Manager selects the most recent backups to restore unless qualified by a tag or a **set until** command. The most current suitable backup containing the specified file may not be the most recent backup, as can occur in point-in-time recovery.

For example, if datafile copies are created each Monday evening and are always tagged *mondayPMcopy*, then the tag refers to the most recent copy. Thus, this command switches datafile 3 to the most recent Monday evening copy:

```
switch datafile 3 to datafilecopy tag mondayPMcopy;
```

Tags can indicate the intended purpose or usage of different classes of backups or file copies. For example, datafile copies that are suitable for use in a **switch** can be tagged differently from file copies that should be used only for **restore**.

Note: If you specify a tag when specifying input files to a **restore** or **switch** command, RMAN will consider *only* backup sets with a matching tag when choosing which particular backup set or image copy to use.

See Also: For reference information on the **switch** and **restore** commands, see [Chapter 11, "Recovery Manager Command Syntax"](#).

Restoring Files

Use the RMAN **restore** command to restore datafiles, control files, and archived redo logs from backup sets or image copies on disk. Because a backup set is in an Oracle proprietary format, you cannot simply import it; you must use the RMAN **restore** command to extract it. In contrast, Oracle can use image copies created using RMAN without additional processing.

You can restore:

- Datafiles
- Control files (current or copies)
- Archived redo logs

Note: You do not normally restore archived redo logs because RMAN performs this operation automatically as needed during recovery. You can improve the recovery performance, however, by pre-restoring backups of archived redo logs that you will need during the recovery.

RMAN completely automates the procedure for restoring files. You do not need to go into the O/S, locate the backup or copy that you want to use, and manually copy files into the appropriate directories. RMAN directs a server session to restore the correct backups and copies to either:

- The default location, which overwrites the files with the same name currently there.
- A new location, which you can specify using the **set newname** command. If you restore datafiles to a new location, then Oracle considers them datafile copies and records them as such in the control file and recovery catalog.

See Also: To learn how to restore backup sets and copies, see "[Restoring Datafiles, Control Files, and Archived Redo Logs](#)" on page 9-2. For reference material on the **restore** command, see "[restore](#)" on page 11-112.

File Selection in Restore Operations

RMAN uses the recovery catalog (or target database control file if no recovery catalog is available) to select the best available backup sets or image copies for use in the restore operation. It gives preference to image copies rather than backup sets. When multiple choices are available, RMAN uses the most current backup sets or copies, taking into account whether you specified the *untilClause*.

All specifications of the **restore** command must be satisfied before RMAN restores a backup set or file copy. The **restore** command also considers the device types of the allocated channels when performing automatic selection.

If no available backup or copy in the recovery catalog satisfies all the specified criteria, then RMAN returns an error during the compilation phase of the restore job. If the file cannot be restored because no backup sets or datafile copies reside on media compatible with the device types allocated in the job, then create a new job specifying channels for devices that are compatible with the existing backup sets or datafile copies.

Restore Constraints

Note the following constraints on the **restore** command:

- You must perform restore operations on a started instance; however, the database does not need to be mounted. This functionality allows you to perform restore operations when the control file has been lost.
- A tablespace or datafile that is to be restored must be offline, or the database must be closed.
- A restore operation either overwrites the existing datafiles or directs its output to a new file via the **set newname** command.

- You cannot restore a plugged-in tablespace until after it has been specified READ WRITE.
- You cannot use RMAN to restore image copies created on one host to a different host. You must transfer the files manually and use the **catalog** command to catalog them before restoring.

Media Recovery

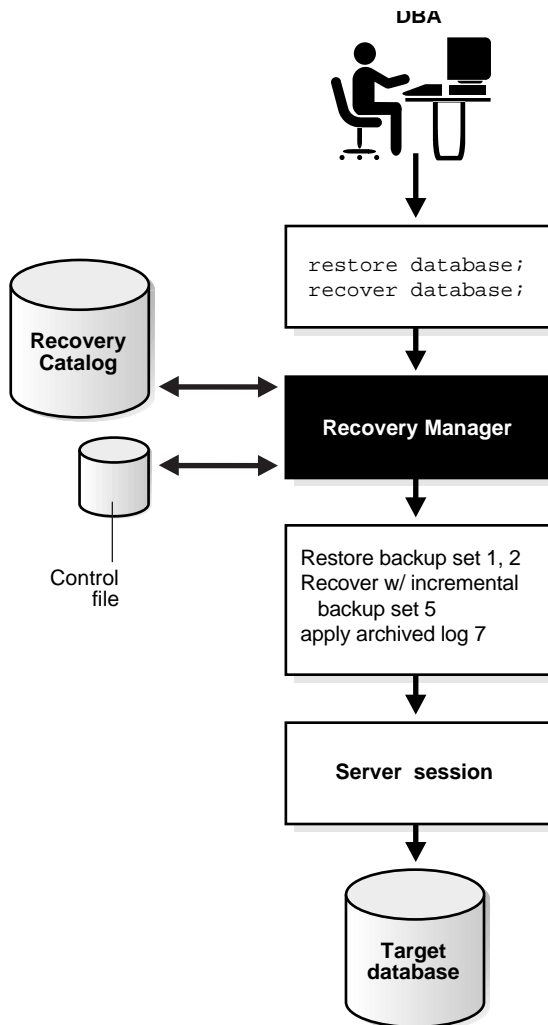
Media recovery is the application of online or archived redo logs or incremental backups to a restored datafile in order to update it to the current time or some other specified time. Use the RMAN **recover** command to perform media recovery and apply incremental backups automatically. You can only recover current datafiles.

If possible, make the recovery catalog available to perform the media recovery. If it is not available, then RMAN uses information from the target database control file. Note that if control file recovery is required, then you must make the recovery catalog available. RMAN cannot operate when neither the recovery catalog nor the target database control file is available.

The generic steps for media recovery using RMAN are:

- If you want to perform incomplete recovery, use the **set until** command to specify the time, SCN, or log sequence number at which recovery terminates.
- Restore the necessary files using the **restore** command.
- Recover the datafiles using the **recover** command.

See [Chapter 9, "Restoring and Recovering with Recovery Manager"](#) for detailed restore and recovery procedures.

Figure 4–16 Performing RMAN Media Recovery

See Also: To learn how to recover datafiles, see ["Recovering Datafiles"](#) on page 9-15. For reference material on the `recover` command, see ["recover"](#) on page 11-88.

Application of Incremental Backups and Redo Records

If RMAN has a choice between applying an incremental backup or applying redo to the restored datafiles, then it always chooses to use the incremental backup. If over-lapping levels of incremental backup are available, then RMAN automatically chooses the one covering the longest period of time.

If RMAN cannot find an incremental backup, it looks for an archived redo log. Whenever ARC*n* archives a redo log, Oracle immediately records it in the control file. Recovery Manager propagates that information into the recovery catalog during resynchronization, classifying archived redo logs as image copies. Use the **list** command to display them.

During recovery, RMAN looks for the appropriate archived redo logs in the default locations specified in the parameter file. If it cannot find them anywhere on disk, it looks in backup sets and restores archived redo logs as needed to perform the media recovery.

By default, RMAN restores the archived redo logs to the current log archive destination specified in the `init.ora` file. Use the **set archivelog destination** command to specify a different restore location.

See Also: For **set archivelog destination** syntax, see "[set_run_option](#)" on page 11-133.

Incomplete Recovery

RMAN can perform either complete or incomplete recovery. Using the **set until** command, you can specify a time, SCN, or log sequence number as a limit for incomplete recovery. Typically, you use this command before issuing the **restore** and **recover** commands. After performing incomplete recovery, always open the database with the RESETLOGS option and then immediately back up the database.

Tablespace Point-in-Time Recovery

Recovery Manager automated Tablespace Point-in-Time Recovery (TSPITR) enables you to recover one or more tablespaces to a point-in-time that is different from that of the rest of the database. RMAN TSPITR is most useful in these situations:

- To recover from an erroneous drop or truncate table operation.
- To recover a table that has become logically corrupted.
- To recover from an incorrect batch job or other DML statement that has affected only a subset of the database.

- In cases where there are multiple logical databases in separate tablespaces of one physical database, and where one logical database must be recovered to a point different from that of the rest of the physical database.
- For VLDBs (very large databases) even if a full database point-in-time recovery would suffice, you might choose to do tablespace point-in-time recovery rather than restore the whole database from a backup and perform a complete database roll-forward.

Similar to a table export, RMAN TSPITR enables you to recover a consistent data set; however, the data set is the entire tablespace rather than just one object.

See Also: To learn how to perform TSPITR using RMAN, see [Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager"](#).

Database Duplication

Use the RMAN **duplicate** command to create a test database on which to practice your backup and recovery procedures. The command takes backup sets located on disk of your primary database's files and uses them to create a new database. A test database is especially useful if your production database must be up and running 24 hours per day, 7 days a week.

As part of the duplicating operation, RMAN manages the following:

- Restores the target datafiles into the duplicate database and performs incomplete recovery using all available archived redo log and incremental backups.
- Opens the duplicate database with the **RESETLOGS** option after incomplete recovery to create the online redo logs.
- Generates a new, unique database identifier for the duplicate database.

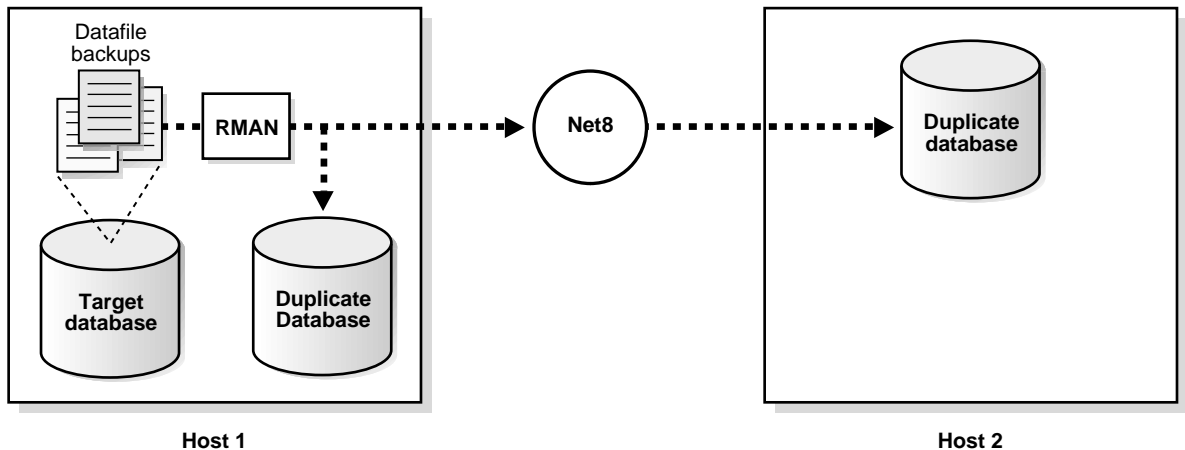
Note also the following features of RMAN duplication. You can:

- Skip read-only tablespaces with the **skip readonly** clause. Read-only tablespaces are included by default. If you omit them, you can add them later.
- Create your duplicate database in a new host. If the same directory structure is available, you can use the **nofilenamecheck** option and re-use the target datafile filenames for the duplicate datafiles.
- Create your duplicate database by using the **set until** option to recover it to a non-current time. By default, the **duplicate** command creates the database using the most recent backups of the target database and then performs

recovery to the most recent consistent point contained in the incremental and archived redo log backups.

- Use the duplicate database without a recovery catalog.
- Register the duplicate database in the same recovery catalog as the target database. This option is possible because the duplicate database receives a new database identifier during duplication. If you copy the target database using O/S utilities, the database identifier of the copied database remains the same so you cannot register it in the same recovery catalog.

Figure 4–17 *Creating a Duplicate Database from Backups*



The method you use to duplicate your database depends on whether you are creating your duplicate database on the same or a different host and whether the duplicate directory structure is the same as your target database filesystem. For example, in some cases you can keep the same directory structure and filenames in your duplicate database, while in other cases you must reset the filenames using **set newname** commands, the `DB_FILE_NAME_CONVERT` initialization parameter, or both.

See Also: To learn how to make a duplicate database, see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#). For **duplicate** command syntax, see "duplicate" on page 11-69.

Integrity Checks

Oracle prohibits any attempts to perform operations that will result in unusable backup files or corrupt restored datafiles. Oracle performs integrity checks to:

- Ensure that restore operations do not corrupt the database by applying backups from a previous incarnation of the database.
- Ensure that incremental backups are applied in the correct order.
- Prohibit accessing datafiles that are in the process of being restored or recovered.
- Allow only one restore operation per datafile at a time.
- Prohibit backups of unrecovered backup files.
- Control information stored in backups to ensure that corrupt backup files are detected.

Detection of Physical Block Corruption

Because an Oracle server session is performing backup and copy operations, the server session is able to detect many types of corrupt blocks. Each new corrupt block not previously encountered in a backup or copy operation is recorded in the control file and in the `alert.log`.

RMAN queries corruption information at the completion of a backup and stores it in the recovery catalog and control file. Access this data using the views `V$BACKUP_CORRUPTION` and `V$COPY_CORRUPTION`.

If RMAN encounters a datafile block during a backup that has already been identified as corrupt by the database, then the server session copies the corrupt block into the backup and Oracle logs the corruption in the control file as either a logical or media corruption.

If RMAN encounters a datafile block with a corrupt header that has not already been identified as corrupt by the database, then it writes the block to the backup with a reformatted header indicating that the block has media corruption.

Note: RMAN cannot detect all types of corruption.

Detection of Logical Block Corruption

RMAN can test data and index blocks that pass physical corruption checks for logical corruption, e.g., corruption of a row piece or index entry. If RMAN finds logical corruption, it logs the block in the `alert.log` and server session trace file.

Provided the sum of physical and logical corruptions detected for a file remain below its **maxcorrupt** setting, the RMAN command completes and Oracle populates `V$BACKUP_CORRUPTION` and `V$COPY_CORRUPTION` with corrupt block ranges. If **maxcorrupt** is exceeded, the command terminates without populating the views.

Note: For **copy** and **backup** commands the **maxcorrupt** setting represents the total number of physical and logical corruptions permitted on a file.

Detection of Fractured Blocks During Open Backups

When performing open backups *without* using Recovery Manager, you must put tablespaces in *hot backup mode* in case the operating system reads a block for a backup that is currently being written by `DBWn`, and is thus inconsistent. Thus, the block is a *fractured block*.

When performing a backup using RMAN, an Oracle server session reads the datafiles, not an operating system utility. The Oracle server session reads whole Oracle blocks and checks to see whether the block is fractured by comparing control information stored in the header and footer of each block. If the session detects a fractured block, then it re-reads the block. For this reason, do not put tablespaces into hot backup mode when using Recovery Manager to back up or copy database files.

See Also: For information about hot backup mode, see "[Backing Up Online Tablespaces and Datafiles](#)" on page 13-6.

Getting Started with Recovery Manager

This chapter describes how to get started using RMAN. It includes the following topics:

- [Setting Up Recovery Manager](#)
- [Consequences of Using the Control File for RMAN Metadata](#)
- [Connecting to RMAN](#)
- [Using Basic RMAN Commands](#)
- [Configuring a Media Manager](#)
- [Using Sample Scripts and Scenarios](#)

Setting Up Recovery Manager

Before using RMAN, you should decide:

- Whether you will use password files for authentication.
- How you will set the NLS environment variables.
- Where you will maintain the snapshot control file.

Using Password Files

Typically, you need to use a password file when connecting to the target database over a non-secure Net8 connection, especially when you:

- Run RMAN remotely on a different machine from the target database.
- Use RMAN with a net service name in the database connect string.
- Run the database in OPS mode and wish to back up this database from more than one node in the cluster concurrently while using only one RMAN session.

Note: Recovery Manager does not back up initialization or password files. When developing your backup and recovery strategy, plan how you will protect these files from media failure.

See Also: For an example of a backup distributed over two nodes in an OPS cluster, see *Oracle8i Parallel Server Concepts and Administration*.

Setting NLS Environment Variables

Before invoking RMAN, set the NLS_DATE_FORMAT and NLS_LANG environment variables. These variables determine the format used for the time parameters in RMAN commands such as **restore**, **recover**, and **report**.

The following example shows typical language and date format settings:

```
NLS_LANG=american
NLS_DATE_FORMAT='Mon DD YYYY HH24:MI:SS'
```

Specifying Dates in RMAN Commands

When specifying dates in RMAN commands, the date string can be either:

- A literal string whose format matches the NLS_DATE_FORMAT setting.

- A SQL expression of type DATE, e.g., 'SYSDATE-10' or "TO_DATE('01/30/1997', 'MM/DD/YYYY)". Note that the second example includes its own date format mask and so is independent of the current NLS_DATE_FORMAT setting.

Following are examples of typical date settings in RMAN commands:

```
backup archivelog from time 'SYSDATE-31' until time 'SYSDATE-14';
restore database until time "TO_DATE('12/20/98', 'MM/DD/YY')";
```

Specifying the Database Character Set

If you are going to use RMAN to connect to a non-mounted database and then mount the database later while RMAN is still connected, set the NLS_LANG variable so that it also specifies the character set used by the database.

A database that is not mounted assumes the default character set, which is US7ASCII. If your character set is different from the default, then RMAN will return errors after the database is mounted. To avoid this problem, set the NLS_LANG to specify the target database's character set. For example, if the character set is WE8DEC, set the NLS_LANG parameter as follows:

```
NLS_LANG=american_america.we8dec.
```

Note: You must set both NLS_LANG and NLS_DATE_FORMAT for NLS_DATE_FORMAT to be used.

See Also: For more information on the NLS_LANG and NLS_DATE_FORMAT parameters, see the *Oracle8i Reference*.

Determining the Snapshot Control File Location

When RMAN needs to resynchronize from a read-consistent version of the control file, it creates a temporary *snapshot control file*. The default name for the snapshot control file is port-specific. Use the **set snapshot controlfile name** command to change the name of the snapshot control file; subsequent snapshot control files that RMAN creates use the name specified in the command.

For example, start RMAN and then enter:

```
set snapshot controlfile name to '/oracle/dba/prod/snap_prod.ctl';
```

You can also set the snapshot control file name to a raw device. This operation is important for OPS databases in which more than one instance in the cluster use

RMAN because server sessions on each node must be able to create a snapshot control file with the same name and location. For example, enter:

```
set snapshot controlfile name to '/dev/vgd_1_0/rlvt5';
```

If one RMAN job is already backing up the control file while another needs to create a new snapshot control file, you may see the following message:

```
RMAN-08512: waiting for snapshot controlfile enqueue
```

Under normal circumstances, a job that must wait for the control file enqueue will wait for a brief interval and will then successfully retrieve the enqueue. Recovery Manager makes up to five attempts to get the enqueue and then fails the job. The conflict is usually caused when two jobs are both backing up the control file, and the job that starts backing up the control file first waits for service from the media manager.

See Also: For an overview of RMAN resynchronization using the snapshot control file, see ["Resynchronization of the Recovery Catalog"](#) on page 4-14. For **set** command syntax, see **"set"** on page 11-129.

Deciding Whether to Use a Recovery Catalog

Perhaps the most important decision you make when getting started with RMAN is whether to use a recovery catalog to store RMAN metadata. This section outlines some of the costs and benefits associated with using and not using a recovery catalog. If you decide to create a catalog, see ["Creating the Recovery Catalog"](#) on page 6-2.

See Also: For an overview of RMAN metadata, see ["Recovery Manager Metadata"](#) on page 4-13.

Consequences of Using the Recovery Catalog for RMAN Metadata

When you use a recovery catalog, RMAN can perform a wider variety of automated backup and recovery functions. For this reason, Oracle recommends that you use a recovery catalog with RMAN whenever possible.

When you use a recovery catalog, RMAN requires that you maintain a recovery catalog schema as well as any associated space used by that schema. The size of the recovery catalog schema:

- Depends on the number of databases monitored by the catalog.

- Depends on the number and size of Recovery Manager scripts stored in the catalog.
- Grows as the numbers of archived logs and backups for each database grow.

If you use a recovery catalog, decide which database you will use to install the recovery catalog schema, and also how you will back this database up. If you use RMAN to back up several databases, you may wish to create a separate recovery catalog database and create the RMAN user in that database. Also, decide whether to operate this database in ARCHIVELOG mode, which is recommended.

If you store the recovery catalog in a separate database, you will require a small amount of disk space for the following:

- System tablespace
- Temp tablespace
- Rollback segment tablespaces
- Online redo log files

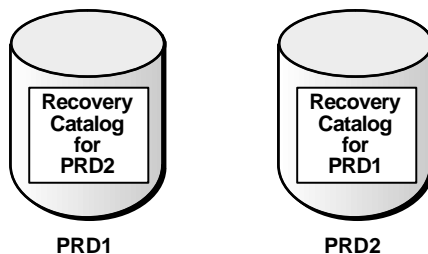
An additional benefit of maintaining a separate recovery catalog is that it is only unavailable at your discretion. Most of the space used in this database is devoted to supporting tablespaces, e.g., the system, temp, and rollback tablespaces.

Table 5–1 Typical Recovery Catalog Space Requirements for 1 Year

Type of Space	Space Requirement
System	50 megabytes
Temp	5 megabytes
Rollback	5 megabytes
Recovery catalog	10 megabytes
Online redo logs	1 megabyte each (3 groups, each with 2 members)

If you have more than one database to back up, you can create more than one recovery catalog and have each database serve as the other's recovery catalog. For example, assume you maintain two production databases, one called PRD1 and a second called PRD2. You can install the recovery catalog for PRD1 in the PRD2 database, and the recovery catalog for the PRD2 database in PRD1.

Figure 5–1 Using Production Databases as Recovery Catalog Databases



By allowing the production databases to serve as each other's recovery catalog, you avoid the extra space requirements and memory overhead of maintaining a separate recovery catalog database. This solution is not practical, however, if the recovery catalog databases for both reside in tablespaces on the same physical disk.

Note: You *must* install the recovery catalog schema in a different database from the target database. If you do not, the benefits of using a recovery catalog are lost if you lose the database and need to restore.

WARNING: Ensure that the recovery catalog and target databases do *not* reside on the same disk. If they are on the same disks and you lose database, then you will probably lose the other.

See Also: To learn how to manage the recovery catalog, see [Chapter 6, "Managing Recovery Manager Metadata"](#).

Consequences of Using the Control File for RMAN Metadata

If you choose not to use a recovery catalog, you can still use RMAN very effectively. RMAN obtains the information it needs from the control file of the target database. The following commands are only available when you use a recovery catalog:

- **change ... available, change ... unavailable, change ... uncatalog, change backupset ... crosscheck, change backuppiece ... crosscheck**
- **create catalog, upgrade catalog, drop catalog**
- **create script, delete script, replace script, print script**

- **crosscheck backup**
- **delete expired backup**
- **list incarnation**
- **register database**
- **report schema at time**
- **reset database**
- **restore** (when no control file is available)
- **resync catalog**
- **set auxname**

To restore and recover your database without using a recovery catalog, Oracle recommends that you:

- Use a minimum of three multiplexed or mirrored control files, with each control file on a separate disk.
- Keep excellent records of which files were backed up, the date they were backed up, and also the names of the backup pieces each file was written to. Use the **list** command to obtain information about backups and copies. Keep all RMAN backup logs.

WARNING: The only way to restore and recover when you have lost all control files and do not use a recovery catalog is to call Oracle WorldWide Customer Support (WWCS). WWCS will need to know the following:

- **Current schema of the database**
 - **Which files were backed up**
 - **What time the files were backed up**
 - **Names of the backup pieces containing the files**
-
-

See Also: For more information about maintaining the control file as the exclusive repository for RMAN metadata, see "[Managing RMAN Metadata Without a Recovery Catalog](#)" on page 6-35.

Connecting to RMAN

To use RMAN, you must first connect to it. This connection is necessary to:

- Authenticate you as a valid user.
- Identify your target database, which is the database that you are backing up or restoring.
- Identify the database containing the recovery catalog (if you use a recovery catalog).
- Identify an auxiliary database (if you use an auxiliary database).

Whenever you start RMAN, you must connect to a target database. Even so, you have several options for how you connect to RMAN. For example, you can start RMAN:

- With or without a recovery catalog.
- By connecting to databases at the O/S command line or the RMAN command line.
- In batch mode (using a command file that contains a series of RMAN commands) or interactive mode.
- With a log file that records RMAN output when you run in batch mode.
- By appending to or overwriting the log file.

This section includes the following sample Recovery Manager connection situations:

- [Connecting to RMAN Without a Recovery Catalog](#)
- [Connecting to RMAN with a Recovery Catalog](#)
- [Connecting to an Auxiliary Database](#)
- [Disconnecting from RMAN](#)

See Also: For an exhaustive list of command-line options, see "[cmdLine](#)" on page 11-39.

Connecting to RMAN Without a Recovery Catalog

In these examples, assume that:

<i>sys</i>	User with SYSDBA privileges
<i>target_pwd</i>	The password for connecting as SYSDBA specified in the target database's <code>orapwd</code> file
<i>target_str</i>	The net service name for the target database

Connecting to RMAN Using O/S Authentication

If the target database does not have a password file, then the user must be validated using O/S authentication. You can use O/S authentication only if you connect locally, i.e., RMAN and the target database reside on the same machine.

1. At the UNIX command line, enter:

```
% ORACLE_SID=PROD1; export ORACLE_SID
```

2. Issue the following statement:

```
% rman nocatalog
```

3. At the RMAN prompt enter:

```
RMAN> connect target /
```

Note: You do not need to specify the SYSDBA option because RMAN uses this option automatically.

Connecting to RMAN Using Password Files

If the target database uses password files, you can connect using a password. Use a password file for either local or remote access. You must use a password file if you are connecting remotely using a net service name.

Connecting from the O/S Command Line To connect from the O/S command line, enter the following, where *sys_pwd* is the password for SYS and *target_str* is the net service name for the target database:

```
% rman target sys/target_pwd@target_str nocatalog
```

Connecting from the RMAN Prompt Alternatively, start RMAN and connect to your target database from the RMAN prompt:

```
% rman nocatalog
RMAN> connect target sys/target_pwd@target_str
```

See Also: For information about command line options, see "[cmdLine](#)" on page 11-39. For information about the **connect** command, see "[connect](#)" on page 11-44. To learn about password files, see the *Oracle8i Administrator's Guide*.

Connecting to RMAN with a Recovery Catalog

In these examples, assume that you maintain a recovery catalog and:

<i>sys</i>	User with SYSDBA privileges
<i>rman</i>	Owner of the recovery catalog having RECOVERY_CATALOG_OWNER privilege
<i>target_pwd</i>	The password for connecting as SYSDBA specified in the target database's <code>orapwd</code> file
<i>target_str</i>	The net service name for the target database
<i>cat_pwd</i>	The password for user RMAN specified in the recovery catalog's <code>orapwd</code> file
<i>cat_str</i>	The net service name for the recovery catalog database

Connecting to RMAN Using O/S Authentication

If the target database does not have a password file, then the user must be validated using O/S authentication. You can use O/S authentication only if you connect locally, i.e., RMAN and the target database reside on the same machine.

1. If RMAN is running on the same machine as your target database, set the `ORACLE_SID` to the target database. For example, at the UNIX prompt type:

```
% ORACLE_SID=PROD1; export ORACLE_SID
```

2. Issue the following statement to connect to the recovery catalog as user RMAN:

```
% rman catalog rman/cat_pwd@cat_str
```

3. Once RMAN has started, issue a **connect target** command (which assumes that you have SYSDBA privileges):

```
RMAN> connect target
```


Connecting to RMAN Using Password Files

If the target and recovery catalog databases use password files, you can connect using a password. Use a password file for either local or remote access. You must use a password file if you are connecting remotely via a net service name.

Connecting from the O/S Command Line To connect to RMAN from the O/S command line, enter the following:

```
% rman target sys/target_pwd@target_str catalog rman/cat_pwd@cat_str
```

Connecting from the RMAN Prompt Alternatively, you can start RMAN and connect to the target database from the RMAN prompt:

```
% rman
RMAN> connect target sys/target_pwd@target_str
RMAN> connect catalog rman/cat_pwd@cat_str
```

Connecting to an Auxiliary Database

To use the **duplicate** command or to perform RMAN TSPITR, you need to connect to an auxiliary instance. In these examples, assume that:

<i>sys</i>	User with SYSDBA privileges
<i>rman</i>	Owner of the recovery catalog having RECOVERY_CATALOG_OWNER privilege
<i>target_pwd</i>	The password for connecting as SYSDBA specified in the target database's <code>orapwd</code> file
<i>target_str</i>	The net service name for the target database
<i>cat_pwd</i>	The password for user RMAN specified in the recovery catalog's <code>orapwd</code> file
<i>cat_str</i>	The net service name for the recovery catalog database
<i>aux_pwd</i>	The password for connecting as SYSDBA specified in the auxiliary database's <code>orapwd</code> file.
<i>aux_str</i>	The net service name for the auxiliary database.

If the auxiliary database uses password files, you can connect using a password. Use a password file for either local or remote access. You must use a password file if you are connecting remotely via a net service name.

Connecting from the O/S Command Line To connect to an auxiliary instance from the O/S command line, enter the following:

```
% rman auxiliary sys/aux_pwd@aux_str
```

To connect to the target, auxiliary, and recovery catalog databases, issue the following (all on one line):

```
% rman target sys/target_pwd@target_str catalog rman/cat_pwd@cat_str \  
> auxiliary sys/aux_pwd@aux_str
```

Connecting from the RMAN Prompt Alternatively, you can start RMAN and connect to the auxiliary database from the RMAN prompt:

```
% rman  
RMAN> connect auxiliary sys/aux_pwd@aux_str
```

To connect to the target, auxiliary, and recovery catalog databases, issue the following:

```
% rman  
RMAN> connect target sys/target_pwd@target_str  
RMAN> connect catalog rman/cat_pwd@cat_str  
RMAN> connect auxiliary sys/aux_pwd@aux_str
```

See Also: For **duplicate** command syntax, see "[duplicate](#)" on page 11-69. To learn how to perform RMAN TSPITR, see [Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager"](#).

Disconnecting from RMAN

To disconnect from RMAN, type **exit** at the RMAN prompt:

```
RMAN> exit
```

Using Basic RMAN Commands

Once you have learned how to connect to RMAN, you can immediately begin performing backup, recovery, and maintenance operations. Use the examples in this section to sample a few basic commands.

These examples assume the following:

- You are running in ARCHIVELOG mode.
- RMAN is running on the same machine as your target database.
- You are connecting from the command line using O/S authentication.

- You are not running an OPS configuration.

You will learn how to perform the following tasks:

- [Connecting to RMAN](#)
- [Mounting the Database](#)
- [Reporting the Current Schema](#)
- [Copying a Datafile](#)
- [Backing Up a Tablespace](#)
- [Listing Backups and Copies](#)
- [Validating a Restore](#)

Connecting to RMAN

Your first task is to connect to the target database. If you have created a recovery catalog, you can connect to it as well—although these examples assume you are connecting without a recovery catalog.

At the command line, enter the following:

```
% rman target / nocatalog
```

If your database is already mounted or open, you will see output similar to the following:

```
Recovery Manager: Release 8.1.5.0.0
```

```
RMAN-06005: connected to target database: RMAN (DBID=1237603294)
```

```
RMAN-06009: using target database controlfile instead of recovery catalog
```

The DBID displayed is the database identifier for your database.

If your database is not started, RMAN displays the following message when it connects:

```
RMAN-06193: connected to target database (not started)
```

See Also: For command line connection options, see "[cmdLine](#)" on page 11-39.

Mounting the Database

Because you are not using a recovery catalog, RMAN must obtain the information it needs from the target database control file. Consequently, the database must be mounted or open. For these examples, we will mount the database but not open it.

If the database is not started, issue the **startup** command, specifying a parameter file if the file is in a non-default location. This example uses the parameter file `initPRODL.ora`:

```
RMAN> startup mount pfile=/oracle/dbs/temp/initPRODL.ora
```

```
RMAN-06196: Oracle instance started  
RMAN-06199: database mounted
```

```
Total System Global Area      19799144 bytes  
  
Fixed Size                      64616 bytes  
Variable Size                  11001856 bytes  
Database Buffers               8192000 bytes  
Redo Buffers                   540672 bytes
```

```
RMAN>
```

RMAN displays the size of the SGA, including the size of the database and redo buffers, and returns you to the RMAN prompt.

If the database is open, issue the following to close it cleanly and then mount it:

```
RMAN> shutdown immediate
```

```
RMAN-06405: database closed  
RMAN-06404: database dismounted  
RMAN-06402: Oracle instance shut down
```

```
RMAN> startup mount pfile = initPRODL.ora # specify a parameter file if necessary
```

See Also: For **startup** syntax, see "[startup](#)" on page 11-142.

Reporting the Current Schema

In this example, ask RMAN to report which datafiles your target database contains. Use the **report schema** command as follows:

```
RMAN> report schema;
```

RMAN displays the datafiles currently in your database. Depending on the contents of your database, you will see output similar to the following:

```

RMAN-03022: compiling command: report
Report of database schema
File K-bytes    Tablespace      RB segs Name
-----
1          47104 SYSTEM          ***    /oracle/dbs/tbs_01.f
2           978 SYSTEM          ***    /oracle/dbs/tbs_02.f
3           978 TBS_1           ***    /oracle/dbs/tbs_11.f
4           978 TBS_1           ***    /oracle/dbs/tbs_12.f
5           978 TBS_2           ***    /oracle/dbs/tbs_21.f
6           978 TBS_2           ***    /oracle/dbs/tbs_22.f
7           500 TBS_1           ***    /oracle/dbs/tbs_13.f
8           500 TBS_2           ***    /oracle/dbs/tbs_23.f
9           500 TBS_2           ***    /oracle/dbs/tbs_24.f
10          5120 SYSTEM          ***    /oracle/dbs/tbs_03.f
11          2048 TBS_1           ***    /oracle/dbs/tbs_14.f
12          2048 TBS_2           ***    /oracle/dbs/tbs_25.f

```

See Also: For **report** syntax, see "**report**" on page 11-102. To learn how to make lists and reports, see [Chapter 7, "Generating Lists and Reports with Recovery Manager"](#).

Copying a Datafile

In this example, copy datafile 1 to a new location. Of course, if you do not want to copy this particular datafile, copy any datafile you choose. This example allocates the disk channel c1 and creates a datafile copy named df1 .bak.

Use the **copy** command as follows:

```

RMAN> run {
2> allocate channel c1 type disk;
3> copy datafile 1 to 'df1.bak';
4> }

```

You will see output similar to the following:

```

RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel c1: sid=12 devtype=DISK

RMAN-03022: compiling command: copy
RMAN-03023: executing command: copy
RMAN-08000: channel c1: copied datafile 1
RMAN-08501: output filename=/oracle/dbs/df1.bak recid=3 stamp=352381826
RMAN-08031: released channel: c1

```

The RMAN-08000 message informs you that the copy was successful. Note that RMAN displays the full filename of the output copy in message RMAN-08501.

See Also: For **copy** syntax, see "[copy](#)" on page 11-48. For **allocate** syntax, see "[allocate](#)" on page 11-9. To learn how to make image copies, see "[Making Image Copies](#)" on page 8-10.

Backing Up a Tablespace

In this example, back up your SYSTEM tablespace to disk. Of course, you can choose to back up a different object.

This example backs up the tablespace to its default backup location, which is port-specific: on UNIX systems the location is \$ORACLE_HOME/dbs. Because you do not specify the **format** parameter, RMAN automatically assigns the backup a unique filename.

Use the **backup** command as follows:

```
RMAN> run {
2> allocate channel c1 type disk;
3> backup tablespace system;
4> }
```

You will see output similar to the following:

```
RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel c1: sid=12 devtype=DISK

RMAN-03022: compiling command: backup
RMAN-03023: executing command: backup
RMAN-08008: channel c1: starting full datafile backupset
RMAN-08502: set_count=1 set_stamp=352382211 creation_time=18-DEC-98
RMAN-08010: channel c1: specifying datafile(s) in backupset
RMAN-08522: input datafile fno=00001 name=/vobs/oracle/dbs/tbs_01.f
RMAN-08011: including current controlfile in backupset
RMAN-08522: input datafile fno=00016 name=/oracle/dbs/tbs_03.f
RMAN-08522: input datafile fno=00002 name=/oracle/dbs/tbs_02.f
RMAN-08013: channel c1: piece 1 created
RMAN-08503: piece handle=/oracle/dbs/lhagl83_1_1 comment=NONE
RMAN-08525: backup set complete, elapsed time: 00:00:27
RMAN-08031: released channel: c1
```

The RMAN-08525 message informs you that RMAN created the backup set successfully. Note that RMAN displays the full filename for the backup piece in message RMAN-08503.

See Also: For **backup** syntax, see "[backup](#)" on page 11-21. To learn how to make image copies, see "[Making Backups](#)" on page 8-2.

Listing Backups and Copies

In this example, list your backup sets and image copies. Issue the **list** command as follows:

```
RMAN> list backup;
```

You will see output similar to the following:

```
List of Backup Sets
Key       Recid      Stamp      LV Set Stamp  Set Count  Completion Time
-----
3         3          352382231  0 352382211  49         18-DEC-98

List of Backup Pieces
Key       Pc# Cp# Status      Completion Time      Piece Name
-----
2         1  1  AVAILABLE  18-DEC-98            /oracle/dbs/lhagl83_1_1

List of Datafiles Included
File Name                                LV Type Ckp SCN      Ckp Time
-----
1       /oracle/dbs/tbs_01.f                    0 Full 114149   18-DEC-98
2       /oracle/dbs/tbs_02.f                    0 Full 114149   18-DEC-98
16      /oracle/dbs/tbs_03.f                    0 Full 114149   18-DEC-98
```

RMAN tells you which backup sets and pieces it created as well as which datafiles it included in those sets.

Now list your image copies as follows:

```
RMAN> list copy;
```

You will see output similar to the following:

```
List of Datafile Copies
Key       File S Completion time Ckp SCN      Ckp time      Name
-----
2         1  A 18-DEC-98      114148      18-DEC-98     /oracle/dbs/df1.bak
```

See Also: For an explanation of the column headings in the **list** output, see "[list](#)" on page 11-76. To learn how to make lists and reports, see "[Generating Lists](#)" on page 7-2.

Validating a Restore

Finally, check that you are able to restore your backup in case of a media failure. Use the output from the **list backup** command to determine the primary key for the backup set:

```
List of Backup Sets
-----
Key          Recid          Stamp          LV Set Stamp  Set Count  Completion Time
-----
3            3              352382231     0 352382211    49         18-DEC-98
```

In this example, the primary key is 3. Use the primary key value in your backup set in the **validate backupset** command as follows:

```
RMAN> run {
2> allocate channel c1 type disk;
3> validate backupset 3;
4> }
```

You should see output similar to the following:

```
RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel c1: sid=12 devtype=DISK

RMAN-03022: compiling command: validate
RMAN-03023: executing command: validate
RMAN-08096: channel c1: starting validation of datafile backupset
RMAN-08502: set_count=49 set_stamp=352382211 creation_time=18-DEC-98
RMAN-08023: channel c1: restored backup piece 1
RMAN-08511: piece handle=/oracle/dbs/lhagl1r83_1_1 params=NULL
RMAN-08098: channel c1: validation complete
RMAN-08031: released channel: c1
```

If there are no error messages, then RMAN confirms that it is able to restore the backup set. When there is an error, RMAN always displays an error banner and provides messages indicating the nature of the error.

For example, if you try to allocate a channel on the target database when not connected to it you see:

```
RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03002: failure during compilation of command
RMAN-03013: command type: allocate
RMAN-06171: not connected to target database
```


See Also: For **validate** syntax, see "[validate](#)" on page 11-150. To learn how to restore backups and copies, see "[Restoring Datafiles, Control Files, and Archived Redo Logs](#)" on page 9-2.

Configuring a Media Manager

To back up to and restore from sequential media such as tape you must integrate a media manager with Oracle. This section includes the following topics:

- [Linking with a Media Manager](#)
- [Generating Unique Filenames](#)
- [Limiting File Size](#)
- [Sending Device-Specific Strings to the Media Manager](#)

See Also: For an overview of media management software, see "[Media Management](#)" on page 4-16.

Linking with a Media Manager

To integrate Oracle with a media manager, you must:

- Install and configure media manager software and hardware.
- Obtain your vendor's media management library (MML) interface software, which you then link with the Oracle Server. This integration enables Oracle server sessions to call the media manager.

Note: For instructions on how to achieve this integration on your platform, see your operating system-specific Oracle documentation and the documentation supplied by your media manager.

Generating Unique Filenames

Use the substitution variables provided by RMAN to generate unique backup piece names when writing backups to a media manager (see "[backup](#)" on page 11-21 for the complete list of variables). A backup piece name is determined by the format string specified either in the **backup** command or in the **allocate channel** command.

If you do not specify the **format** parameter, RMAN automatically generates a unique filename using the %U substitution variable. The media manager considers

the backup piece name as the filename backed up, so this name must be unique in the media manager catalog.

Note: Some media managers only support a 14-character backup piece name. See your media management documentation to determine the limit for your media manager.

Limiting File Size

Some media managers have limits on the maximum size of files that they can back up or restore. File size is an issue in those situations in which RMAN multiplexes multiple datafiles into one output file, but the backup piece size is in excess of the size that the media manager or file system is able to store.

To avoid problems, see your media management documentation for operational limits on file sizes. Ensure that the files written out by RMAN do not exceed these limits. To limit backup piece file sizes, use the parameter **kbytes** in a **set limit channel** command. See the scripts in your \$ORACLE_HOME/rdbms/demo directory for an example.

See Also: To learn about various parameters affecting the **run** command, see "[set_run_option](#)" on page 11-133.

Sending Device-Specific Strings to the Media Manager

Use the **send** command to send a vendor-specific quoted string to the media management software. See your media management documentation to determine which commands it supports. You can use:

- The **send** command with no other operands to send the string to all allocated channels.
- The **send device type** command to send the string to all channels of the specified device type.
- The **send channel** command to send the string to channels specified in the command.

See Also: For **send** command syntax, see "[send](#)" on page 11-127.

Troubleshooting the Media Manager

To aid in troubleshooting media management, Oracle offers a client program, `sbttest`. This program, which is linked to RMAN to perform backups to tape, provides a stand-alone test of the media management software. Use it when Oracle is unable to create or restore backups using either the bundled Legato Storage Manager or another vendor's media management product. Only use the `sbttest` program at the direction of Oracle support.

Using Sample Scripts and Scenarios

The `$ORACLE_HOME/rdbms/demo` sub-directory (the location may differ depending on your operating system) contains a number of sample RMAN scripts. These files are executable Recovery Manager command files that are fully documented so that you can understand the features used. Edit them to customize them for your site.

The first file provides a number of scripts that back up, restore, and recover a database. These scripts are typical of how some DBAs back up their databases; use them as a starting point for developing backup, restore, and recovery scripts. The remaining files contain either a backup, recovery, or duplication scenario.

Run command files from either the O/S command line or the RMAN prompt. If you use a recovery catalog, you can also create scripts using the **create script** command and execute them within a **run** command.

See Also: To learn how to run command files from the RMAN prompt, see "[rmanCmd](#)" on page 11-121. To learn how to run command files from the command line, see "[cmdLine](#)" on page 11-39. To learn how to create and execute stored scripts, see "[Storing Scripts in the Recovery Catalog](#)" on page 6-20.

Managing Recovery Manager Metadata

This chapter describes how to manage the RMAN metadata. Depending on how you implement RMAN, you can store this data either in the recovery catalog or exclusively in the control file. The chapter includes the following topics:

- [Creating the Recovery Catalog](#)
- [Maintaining RMAN Metadata](#)
- [Storing Scripts in the Recovery Catalog](#)
- [Backing Up and Recovering the Recovery Catalog](#)
- [Upgrading the Recovery Catalog](#)
- [Dropping the Recovery Catalog](#)
- [Managing RMAN Metadata Without a Recovery Catalog](#)

Creating the Recovery Catalog

To use a recovery catalog, you need to set up the schema. Oracle suggests you put the recovery catalog schema in its own tablespace; however, you can put it in the SYSTEM tablespace if necessary.

Install the recovery catalog schema in a different database from the target database you will be backing up. If you do not, the benefits of using a recovery catalog are lost should you lose the database and need to restore.

WARNING: Ensure that the recovery catalog and target databases do *not* reside on the same disks; if they do and you lose one database, you will probably lose the other.

Assume the following for the examples below:

- User SYS with password CHANGE_ON_INSTALL has SYSDBA privileges on the recovery catalog database RCAT.
- There is a tablespace called RCVCAT on the recovery catalog database RCAT that stores the recovery catalog.
- There is a tablespace called TEMP in the recovery catalog database.
- The database is configured in the same way as all normal databases, e.g., `catalog.sql` and `catproc.sql` have successfully run.

See Also: To learn how to connect to RMAN, see "[Connecting to RMAN](#)" on page 5-8. To learn about the advantages and disadvantages of maintaining a recovery catalog, see "[Deciding Whether to Use a Recovery Catalog](#)" on page 5-4.

To set up the recovery catalog schema:

1. Start SQL*Plus and then connect with administrator privileges to the database containing the recovery catalog. For example, enter:

```
SQL> CONNECT sys/change_on_install@rcat
```

2. Create a log file that you can use to check for errors. For example, enter:

```
SQL> SPOOL create_rman.log
```

3. Create a user and schema for the recovery catalog. For example, enter:

```
SQL> CREATE USER rman IDENTIFIED BY rman  
2> TEMPORARY TABLESPACE temp
```

```
3> DEFAULT TABLESPACE rcvcat
4> QUOTA UNLIMITED ON rcvcat;
```

4. Grant the `RECOVERY_CATALOG_OWNER` role to the schema owner. This role provides the user with privileges to maintain and query the recovery catalog.

```
SQL> GRANT recovery_catalog_owner TO rman;
```

5. Grant other desired privileges to the RMAN user.

```
SQL> GRANT connect, resource TO rman;
```

6. Host out to the operating system to check the `create_rman.log` file for any errors before continuing. For example, a UNIX user can issue:

```
SQL> host
% vi create_rman.log
```

To create the recovery catalog:

1. Connect to the recovery catalog from the O/S command line. For example, enter:

```
% rman catalog rman/rman@rcat log = create_rman.log
```

You can also connect from the RMAN prompt:

```
% rman log = create_rman.log
RMAN> connect catalog rman/rman@rcat
```

2. Issue the **create catalog** command to create the catalog, specifying the `RCVCAT` tablespace:

```
RMAN> create catalog tablespace rcvcat;
```

3. Host out to the operating system to check the `create_rman.log` file for any errors before continuing. For example, a UNIX user might issue:

```
RMAN> host;
% vi create_rman.log
```

See Also: For **create catalog** command syntax, see "[createCatalog](#)" on page 11-52. For the SQL syntax for the `GRANT` and `CREATE USER` statements, see the *Oracle8i SQL Reference*.

Maintaining RMAN Metadata

This section describes how to manage the RMAN information repository. It assumes that you are using a recovery catalog. If you use a control file as the exclusive repository for RMAN metadata, most RMAN maintenance commands will work. See "[Consequences of Using the Control File for RMAN Metadata](#)" on page 5-6 for a list of catalog-only commands.

This section includes the following topics:

- [Registering a Database with the Recovery Catalog](#)
- [Unregistering a Database from the Recovery Catalog](#)
- [Resetting the Recovery Catalog](#)
- [Changing the Availability of a Backup or File Copy](#)
- [Crosschecking RMAN Metadata](#)
- [Deleting Backups and Copies and Updating Their Status in the RMAN Metadata](#)
- [Validating the Restore of Backups and Copies](#)
- [Resynchronizing the Recovery Catalog](#)
- [Managing Records in the Control File](#)
- [Cataloging O/S Backups](#)

Registering a Database with the Recovery Catalog

Before using RMAN with a target database, register the target database in the recovery catalog. If you do not, RMAN cannot use the recovery catalog to store information about the target database. RMAN obtains all information it needs to register the target database from the database itself. You can register more than one target database in the same recovery catalog.

Note: You can only register a database once.

To register the target database:

1. Connect to the target database and recovery catalog. For example, issue the following to connect to the catalog database RCAT as RMAN:

```
% rman target / catalog rman/rman@rcat
```


2. If the database is not mounted, then mount or open it. For example, issue:

```
startup mount;
```

3. If you are creating a new database or migrating an existing Oracle7 database, or if RMAN is installed for use with an existing version 8.0 or higher database, issue the following command:

```
register database;
```

4. If there are any existing user-created backups on disk that were created under version 8.0 or higher, add them to the recovery catalog by issuing the following command:

```
catalog datafilecopy 'filename';
```

For an Oracle7 backup to be usable for recovery in database of a later version, it must have been part of a tablespace that was offline normal or read-only when the database was migrated. See "[Cataloging O/S Backups](#)" on page 6-27.

Note: Backups made with the Enterprise Backup Manager (EBU) cannot be used or cataloged by RMAN.

5. RMAN automatically obtains information about the original archived redo logs from the target database control file. If you have made additional O/S backups of your archived logs, or if log records have aged out of the target control file, catalog them. For example, you can catalog logs as follows (where log1 etc. refers to the fully specified filename for an archived log):

```
catalog archivelog 'log1', 'log2', 'log3', ... 'logN';
```

Note: To determine whether log records have aged out of the control file, compare the number of logs on disk with the number of records in V\$ARCHIVED_LOG.

Oracle uses an internal, uniquely generated number called the *db identifier* to distinguish one database from another. Oracle generates this number when you create the database.

Typically, each database has a unique identifier; however, an exception occurs with databases that you create by copying files from an existing database (instead of

using a CREATE DATABASE statement). In such cases, RMAN detects the duplicate database identifiers and the **register database** command fails. Avoid this problem by using the **duplicate** command, which copies the database from backups and generates a new database identifier.

If a failure occurs because of duplicate database identifiers, you can create a second recovery catalog in another user's schema by re-executing the **create catalog** command using a different Oracle userid. Then, you can register the database with a duplicate database identifier into the newly created recovery catalog in the new schema.

Note: If you are using RMAN with different target databases that have the same database name and identifier, be extremely careful to always specify the correct recovery catalog schema when invoking Recovery Manager.

See Also: For **register** command syntax, see "[register](#)" on page 11-93. For **catalog** command syntax, see "[catalog](#)" on page 11-32. For **duplicate** command syntax, see "[duplicate](#)" on page 11-69. For issues relating to database migration, see *Oracle8i Migration*.

Unregistering a Database from the Recovery Catalog

RMAN allows you to unregister a database as well as register it. Make sure this procedure is really what you intend, because afterwards RMAN can longer recover all the backups for that database.

To unregister a database:

1. Start RMAN and connect to your target database. Note down the DBID value that is displayed when you use RMAN to connect to your target database. For example, enter:

```
% rman target sys/change_on_install@prodl nocatalog

RMAN-06005: connected to target database: RMAN (DBID=1231209694)
```
2. List the copies and backup sets recorded in the control file and then issue **change ... delete** statements to delete them from the O/S. See "[Generating Lists](#)" on page 7-2 and "[Deleting Backups and Copies and Updating Their Status in the RMAN Metadata](#)" on page 6-13.

- Use SQL*Plus to connect to your recovery catalog database and execute the following query in the recovery catalog to find the correct row of the DB table, setting DB_ID equal to the value you obtained from step 1. For example, enter:

```
SQL> SELECT db_key, db_id FROM db WHERE db_id = 1231209694;
```

This query should return exactly one row.

```
DB_KEY      DB_ID
-----
          1 1237603294
1 row selected.
```

- While still connected to the recovery catalog, enter the following, where DB_KEY and DB_ID are the corresponding columns from the row you got from the query in step 2:

```
SQL> EXECUTE dbms_rcvcat.unregisterdatabase(db_key, db_id)
```

For example, enter:

```
SQL> EXECUTE dbms_rcvcat.unregisterdatabase(1 , 1237603294)
```

Note: The DBMS_RCVCAT.UNREGISTERDATABASE package works on any Oracle version 8 database.

Resetting the Recovery Catalog

Before you can use RMAN again with a target database that you have opened with the RESETLOGS option, notify RMAN that you have reset the database incarnation. The **reset database** command directs RMAN to create a new database incarnation record in the recovery catalog. This new incarnation record indicates the current incarnation. RMAN associates all subsequent backups and log archiving done by the target database with the new database incarnation.

If you issue the ALTER DATABASE OPEN RESETLOGS command but do not reset the database, then RMAN will not access the recovery catalog because it cannot distinguish between a RESETLOGS command and an accidental restore of an old control file. By resetting the database, you inform RMAN that the database has been opened with the RESETLOGS option.

In the rare situation in which you wish to undo the effects of opening with the RESETLOGS option by restoring backups of some prior incarnation of the database, use the **reset database to incarnation key** command to change the current incarnation to an older incarnation.

To reset the recovery catalog to an older incarnation:

1. Specify the primary key of the desired database incarnation. Obtain the incarnation key value by issuing a **list** command:

```
list incarnation;
```

```
List of Database Incarnations
```

DB Key	Inc Key	DB Name	DB ID	CUR	Reset SCN	Reset Time
1	2	PROD1	1224038686	NO	1	02-JUL-98
1	582	PROD1	1224038686	YES	59727	10-JUL-98

2. Reset the database to the old incarnation. For example, enter:

```
reset database to incarnation 2;
```

3. After resetting the database, issue **restore** and **recover** commands to restore and recover the database files from the prior incarnation, then open the database with the **RESETLOGS** option. For example, enter:

```
run {
  allocate channel chl type disk;
  restore database;
  recover database;
  sql "ALTER DATABASE OPEN RESETLOGS";
}
```

See Also: For **reset database** command syntax, see "[reset](#)" on page 11-110. For **list** command syntax, see "[list](#)" on page 11-76.

Changing the Availability of a Backup or File Copy

The **unavailable** option provides for cases when a backup or copy cannot be found or has migrated offsite. A file that is marked unavailable will not be used in a **restore** or **recover** command. If the file is later found or returns to the main site, then you can mark it available again by using the **available** operand. Note that you do not need to allocate a channel of type **maintenance** for this operation.

Note: You must use a recovery catalog when executing **change ... unavailable** and **change ... available**.

To mark a backup piece or copy as available or unavailable:

1. Issue a **change ... unavailable** command to mark a backup or copy as **unavailable**. For example, enter:

```
change datafilecopy '/oracle/backup/cf_c.f' unavailable;
change backupset 12 unavailable;
```

2. If a previously unavailable file is reinstated, issue **change ... available** to change its status back to **available**.

```
change datafilecopy '/oracle/backup/cf_c.f' available;
change backupset 12 available;
```

See Also: For **change** command syntax, see "[change](#)" on page 11-35.

Crosschecking RMAN Metadata

Because backups and copies can disappear from disk or tape or become corrupted, the RMAN metadata repository can contain outdated information. To ensure that data about backup sets and image copies in the recovery catalog or control file is synchronized with corresponding data on disk or in the media management catalog, perform a [crosscheck](#).

Use either the **change ... crosscheck** or **crosscheck backup** command to check the specified files. Note that these commands do *not* delete O/S files or remove metadata records; you must use separate commands for those operations.

The following table explains the difference between the crosscheck commands:

Command	Catalog Needed?	Purpose
change ... crosscheck	Only for backupset and backuppiece options	To determine whether the backups or copies exist. If RMAN cannot find backup pieces, it marks them as expired . It marks all other types of absent files—image copies and archived redo logs—as deleted . If the files are on disk, RMAN queries the file headers. For other device types, RMAN queries the media manager to see if the file exists in the media catalog.

Command	Catalog Needed?	Purpose
crosscheck backup	Yes	<p>To determine whether backups stored on disk or tape exist. Backups are either backup sets or media-managed proxy copies.</p> <p>This command checks only backup sets marked available or expired, either by examining the backup pieces for type disk or by querying the media manager for type 'sbt_tape'. It only processes backups created on the specified channel.</p> <p>RMAN does not delete backup pieces that it cannot find, but marks them as expired.</p>

See Also: To learn how to delete files and update metadata records, see "[Deleting Backups and Copies and Updating Their Status in the RMAN Metadata](#)" on page 6-13. For **crosscheck** command syntax, see "[crosscheck](#)" on page 11-57.

Crosschecking Backups

Use the crosscheck feature to check the status of a backup on disk or tape. If the backup is on disk, the **change backupset ... crosscheck** and **crosscheck backup** commands determine whether the header of the backup piece is valid; if on tape, the commands simply check that the backups exist.

If a backup piece is unreadable or absent, then RMAN marks the backup piece **expired** in the output of the **list** command and the recovery catalog views. If it was marked **expired** but is now available, RMAN marks the backup piece as **available** in the output of the **list** command and the recovery catalog views.

Use **change backupset ... crosscheck** when you want to provide a list of backup sets or pieces to check; use **crosscheck backup** when you wish to restrict the crosscheck to a specified device type, object type, or date range and let RMAN generate the list of backup sets or pieces.

Note: The **change** command operates only on files that are recorded in the recovery catalog or the control file. The same is true for other commands except **catalog** and **resync from controlfilecopy**.

To provide a list of backups for a crosscheck:

1. Allocate a channel of type **maintenance**:

```
allocate channel for maintenance type 'sbt_tape';
```

2. Identify the desired backup piece, backup set, or proxy copy that you want to check by issuing a **list** command:

```
list backup;
```

3. Check whether the specified backup sets exist. This example checks whether backup sets with the primary keys of 1338, 1339, and 1340 still exist:

```
RMAN> change backupset 1338, 1339, 1340 crosscheck;
```

```
RMAN-03022: compiling command: change
RMAN-08074: crosschecked backup piece: found to be 'EXPIRED'
RMAN-08517: backup piece handle=/oracle/dbs/2eafnuj3_1_1 recid=77 stamp=352057957
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/oracle/dbs/2dafnuj2_1_1 recid=78 stamp=352057957
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/oracle/dbs/2fafnuj3_1_1 recid=79 stamp=352057960
```

If a backup set is no longer available, RMAN marks it as **expired**. If it was marked **expired** and is now available, RMAN marks it **available**.

4. Release the allocated maintenance channel:

```
release channel;
```

To let RMAN generate the list of backups for a crosscheck:

1. Allocate a channel of type **maintenance**:

```
allocate channel for maintenance type 'sbt_tape';
```

2. Check for the backups of the specified database, tablespace, datafile, control file, or archived redo log. Limit the crosscheck according to the time sequence.

For example, check all backups of datafile `tbs_8.f` over the last six months:

```
crosscheck backup of datafile "/oracle/dbs/tbs_8.f" completed after 'SYSDATE-180';
```

If a backup set is no longer available, RMAN marks it as **expired**. If it was marked **expired** and is now available, RMAN marks it **available**.

3. Release the allocated maintenance channel:

```
release channel;
```

Crosschecking Image Copies

Use the **change ... crosscheck** command to determine whether image copies of datafiles, control files, or archived redo logs on disk or tape are valid. If RMAN is unable to find the specified image copy or archived redo log, it updates its status to **deleted**. Query the relevant recovery catalog view to see copies and logs with **deleted** status.

Note that RMAN considers archived redo logs as image copies. If for some reason one or more archived redo logs becomes unavailable, issue a **change archivelog all crosscheck** command so that RMAN will mark the absent logs as **deleted**. You do not need to use a recovery catalog to execute this command.

To crosscheck image copies:

1. Connect to RMAN either with or without a recovery catalog. For example, enter one of the following:

```
% rman target / nocatalog
% rman target / catalog rman/rman@rcat
```

2. Identify the image copy that you want to check by issuing a **list** command. This example lists all recorded image copies and archived redo logs:

```
list copy of database archivelog all;
```

List of Datafile Copies

Key	File S	Completion time	Ckp SCN	Ckp time	Name
1262	1	A 18-AUG-98	219859	14-AUG-98	/oracle/dbs/copy/tbs_01.f

List of Archived Log Copies

Key	Thrd	Seq	S	Completion time	Name
789	1	1	A	14-JUL-98	/oracle/work/arc_dest/arcr_1_1.arc
790	1	2	A	11-AUG-98	/oracle/work/arc_dest/arcr_1_2.arc
791	1	3	A	12-AUG-98	/oracle/work/arc_dest/arcr_1_3.arc

3. Use **change ... crosscheck** to check whether the specified copy exists. If not, RMAN updates its status to **deleted**. This example checks whether the datafile copy with primary key 1262 exists:

```
change datafilecopy 1262 crosscheck;
```

```
RMAN-03022: compiling command: change
RMAN-06154: validation succeeded for datafile copy
RMAN-08513: datafile copy filename=/oracle/dbs/copy/tbs_01.f recid=1 stamp=351194732
```


If RMAN is unable to validate the copy, you will see:

```
RMAN-06153: validation failed for datafile copy
```

To update the RMAN metadata after archived redo logs have been removed:

1. Connect to RMAN with or without a catalog. For example, enter either:

```
% rman target / nocatalog
% rman target / catalog rman/rman@rcat
```

2. Issue a **change ... crosscheck** command to update the metadata records for the absent archived redo logs. This example crosschecks all archived redo logs:

```
change archivelog all crosscheck;
```

See Also: For **crosscheck** command syntax, see "[crosscheck](#)" on page 11-57. For **change** command syntax, see "[change](#)" on page 11-35. For **list** command syntax, see "[list](#)" on page 11-76.

Deleting Backups and Copies and Updating Their Status in the RMAN Metadata

You can use RMAN to delete backups and copies and update their status in the control file or recovery catalog to **deleted** status. Note that backup and copies with **deleted** status do not appear in the **list** command output: query the recovery catalog views instead.

Following are typical scenarios:

Command or Script	Need Catalog?	Purpose
change ... uncatalog	Yes	To remove the record of a specified backup or copy from the recovery catalog. This command does not delete physical backups or copies: it only removes their records. Note: If you resynchronize from a backup control file, the removed records can reappear in the catalog.
prgrmanc.sql	Yes	To remove all records of backups or copies with status deleted from the catalog at one time. The script does not delete physical backups or copies: it only removes their records. Note: If you resynchronize from a backup control file, the removed records can reappear in the catalog.

Command or Script	Need Catalog?	Purpose
delete expired backup	Yes	To update backup sets records from status expired to status deleted . RMAN also physically deletes any expired backups if they still exist. Typically, you issue this command after performing a crosscheck. A crosscheck marks inaccessible backups as expired .
change ... delete	No	To delete physical backup sets, image copies, or archived redo logs and update their metadata records to status deleted . Unlike the delete expired command, change ... delete operates on <i>any</i> backup or copy—not just those marked expired .

See Also: For **change** command syntax, see "[change](#)" on page 11-35. For **delete** command syntax, see "[deleteExpired](#)" on page 11-63. For descriptions of the recovery catalog views, see [Chapter 12, "Recovery Catalog Views"](#).

To update the recovery catalog record for a backup or copy deleted with an O/S utility:

1. Allocate a channel of type **maintenance**:

```
allocate channel for maintenance type disk;
```

2. Issue a **change ... uncatalog** command for the backups or copies that you deleted from the operating system using O/S commands. This example removes references to copies of the control file and datafile 1:

```
change controlfilecopy '/oracle/backup/cf_c.f' uncatalog;
change datafilecopy '/oracle/backup/df_1_c.f' uncatalog;
```

3. Release the allocated channel:

```
release channel;
```

To physically delete backups and copies and update their metadata records:

This procedure does not require the use of a recovery catalog.

1. Check for the expired backup sets and copies. Use the **list** output to obtain their primary keys.

```
list backup of database archivelog all; # lists backups of database files and logs
list copy;
```

2. Allocate a channel of type **delete**:

```
allocate channel for delete type 'sbt_tape';
```

3. Issue a **change ... delete** command to eliminate *both* the physical file and change the recovery catalog record to status **deleted**. This example deletes the backup piece with key 101 and the control file copy with key 63 from tape:

```
change backuppiece 101 delete;  
change controlfilecopy 63 delete;
```

4. Release the allocated maintenance channel:

```
release channel;
```

To update expired backup records in the catalog to status deleted (and delete any existing expired backup pieces):

This procedure requires the use of a recovery catalog.

1. Optionally, perform a crosscheck operation (see "[Crosschecking RMAN Metadata](#)" on page 6-9) to mark all inaccessible or absent backups as **expired**. The **delete expired backup** command only operates on expired backups.

2. Allocate a channel of type **delete**:

```
allocate channel for delete type disk;
```

3. Issue a **delete expired backup** command to check for backups marked **expired** and update them to status **deleted**. This example updates all backups registered in the recovery catalog that are expired:

```
delete expired backup;
```

4. Release the allocated channel:

```
release channel;
```

To remove individual copy or backup records from the recovery catalog (without physically deleting the backups or copies):

1. Check for the expired image copies or backups. Use the **list** output to obtain their primary keys. For example, enter:

```
list copy;
```

2. Issue a **change ... uncatalog** command to remove references to the specified backups or copies from the recovery catalog. This example removes records of copies of the control file and of the datafile copy with the primary key 4833:

```
change controlfilecopy '/oracle/backup/cf_c.f' uncatalog;
change datafilecopy 4833 uncatalog;
```

Note that **change ... uncatalog** does not remove files from the O/S; it only removes recovery catalog records.

3. View the relevant recovery catalog view, e.g., RC_DATAFILE_COPY or RC_CONTROLFILE_COPY, to confirm that a given record was removed. For example, this query confirms that the record of copy 4833 was removed:

```
SQL> SELECT cdf_key, status FROM rc_datafile_copy WHERE cdf_key = 4833;
CDF_KEY      S
-----
0 rows selected.
```

To remove all copy or backup records from the recovery catalog (without physically deleting the backups or copies):

You can remove all recovery catalog records of deleted backups and copies at once using the `$ORACLE_HOME/rdbms/prgrmanc.sql` script. Only perform this operation if you do not want a historical record of what you have backed up.

See Also: For **change** command syntax, see "[change](#)" on page 11-35.

1. Allocate a channel of type **maintenance**:

```
allocate channel for maintenance type 'sbt_tape';
```

2. Issue **change ... delete** commands to update the desired records to **deleted** status and remove the file from the O/S or media manager. Issue **list** commands or query the recovery catalog views to obtain primary keys for archived redo logs, backup sets, control file copies, or datafile copies.

```
change backupset 100, 101, 102, 103 delete;
```

3. Release the allocated maintenance channel:

```
release channel;
```

4. Start a SQL*Plus session and connect to the recovery catalog. This example connects to the database RCAT as user RMAN:

```
% sqlplus rman/rman@rcat
```

5. Run the script `prgrmanc.sql` script, which is stored in the `$ORACLE_HOME/rdbms/admin` directory:

```
SQL> @prgrmanc.sql
```

RMAN will remove all records with status **deleted** from the recovery catalog.

To remove incarnation records from the recovery catalog:

1. Allocate a channel of type **maintenance**:

```
allocate channel for maintenance type 'sbt_tape';
```

2. Issue **change ... delete** commands to update unwanted backup pieces, archived redo logs, and image copies to **deleted** status. Issue **list** commands or query the recovery catalog views to obtain primary keys for archived redo logs, backup sets, control file copies, or datafile copies.

```
change backupset 100, 101, 102, 103 delete;
```

3. Release the allocated maintenance channel:

```
release channel;
```

4. Start a SQL*Plus session and connect to the recovery catalog. This example connects to database RCAT as user RMAN:

```
% sqlplus rman/rman@rcat
```

5. Obtain the `DBINC_KEY` values for the incarnations whose records you want to delete by querying the `RC_DATABASE_INCARNATION` recovery catalog view:

```
SQL> SELECT * FROM rc_database_incarnation;
```

6. Execute the following DML statement, where `key_value` is the value of `DBINC_KEY`:

```
SQL> DELETE FROM dbinc WHERE dbinc_key=key_value;
```

RMAN will remove the specified incarnation records from the recovery catalog.

See Also: For more information about the `RC_DATABASE_INCARNATION` recovery catalog view, see "[RC_DATABASE_INCARNATION](#)" on page 12-15.

Validating the Restore of Backups and Copies

A restore *validation* executes a restore test run without actually restoring the files. Test the restore of the entire database or individual tablespaces, datafiles, or control files. The **restore ... validate** and **validate backupset** commands test whether you can restore backups or copies. You should use:

- **restore ... validate** when you want RMAN to choose which backups or copies should be tested.
- **validate backupset** when you want to specify which backup sets should be tested.

See Also: For **restore** command syntax, see "**restore**" on page 11-112. For **validate** command syntax, see "**validate**" on page 11-150.

To let RMAN choose which backup sets or copies to validate:

1. If you want to validate the whole database, close it. If you want to validate individual files, then either close the database or take the individual datafiles offline. This example checks the status of the database, aborts the instance, and then mounts it:

```
SQL> SELECT status FROM v$instance;
```

```
STATUS
-----
OPEN
1 row selected.
```

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP FORCE MOUNT;
```

2. Start RMAN and connect to the target database and optional recovery catalog database:

```
% rman target / catalog rman/rman@rcat
```

3. If you do not want to validate the whole database, identify the backup sets and copies that you want to validate by issuing **list** commands, noting primary keys:

```
list backupset;
list copy;
```

4. Validate the restore of the backup sets and copies. This example validates the restore of the backup control file, SYSTEM tablespace, and all archived redo logs from disk:

```
run {
  allocate channel ch1 type disk;
  allocate channel ch2 type tape;
  restore controlfile validate;
  restore tablespace 'system' validate;
  restore archivelog all validate;
}
```

5. Check the output. If you see an error message stack and then the following, you do not have a backup or copy of one of the files that you are validating:

```
RMAN-06026: some targets not found - aborting restore
```

If you see an error message stack and output similar to the following, for example, then there is a problem with the restore of the specified file:

```
RMAN-03002: failure during compilation of command
RMAN-03013: command type: restore
RMAN-03007: retryable error occurred during execution of command: IRESTORE
RMAN-07004: unhandled exception during command execution on channel c1
RMAN-10035: exception raised in RPC: ORA-19505: failed to identify file
           "oracle/dbs/lfafv9gl_1_1"
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
RMAN-10031: ORA-19624 occurred during call to DBMS_BACKUP_RESTORE.RESTOREBACKUPPIECE
```

If you do not see an error stack, then RMAN successfully validated the files.

To specify which backup sets to validate:

1. Start RMAN and connect to the target database and optional recovery catalog database:

```
% rman target / catalog rman/rman@rcat
```

2. Identify the backup sets that you want to validate by issuing **list** commands:

```
list backup of database archivelog all;
```

3. Validate the restore of the backup sets. This example validates the restore of the backup control file, SYSTEM tablespace, and all archived redo logs from disk:

```
run {
  allocate channel ch1 type disk;
  validate backupset 1121;
}
```

4. Check the output. If you see the RMAN-08024 message then RMAN successfully validated the restore of the specified backup set.

```
RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: ch1
RMAN-08500: channel ch1: sid=10 devtype=DISK

RMAN-03022: compiling command: validate
RMAN-03023: executing command: validate
RMAN-08016: channel ch1: starting datafile backupset restore
RMAN-08502: set_count=47 set_stamp=346169465 creation_time=08-OCT-98
RMAN-08023: channel ch1: restored backup piece 1
RMAN-08511: piece handle=/vobs/oracle/dbs/1faa483p_1_1 params=NULL
RMAN-08024: channel ch1: restore complete
RMAN-08031: released channel: ch1
```

Storing Scripts in the Recovery Catalog

A *stored script* is a sequence of RMAN commands stored within the recovery catalog. It provides a common repository for frequently executed collections of RMAN commands.

For example, you can collect the RMAN commands needed to perform nightly backups into a single script called `nightly_bkup`. Storing the script in the recovery catalog instead of in an O/S text file has the advantage that it is accessible to any DBA using RMAN, regardless of which machine RMAN is executed upon.

RMAN allows you to:

- Create a script and store it in the recovery catalog.
- Execute a stored script.
- Replace a stored script.
- Delete a script from the recovery catalog.
- Print a stored script to the message log file or the screen.
- Obtain a listing of all your stored scripts.

To create a stored script:

1. Start RMAN and connect to the recovery catalog database. For example, enter:

```
% rman catalog rman/rman@rcat
```


2. Write the desired script. For example, this script backs up the database and the archived redo logs:

```
create script b_whole{
  allocate channel ch1 type disk;
  allocate channel ch2 type disk;
  backup database;
  sql 'ALTER SYSTEM ARCHIVE LOG ALL';
  backup archivelog all;
}
```

3. Examine the output. If you see message RMAN-08085, the script was successfully created and stored in the recovery catalog:

```
RMAN-03022: compiling command: create script
RMAN-03023: executing command: create script
RMAN-08085: created script b_whole
```

To execute a stored script:

1. Start RMAN and connect to the recovery catalog database and target database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. Issue a **run** command to execute the script. RMAN inserts the contents of the script between the brackets of **run**. Note that you do not need to allocate channels if you already did so *within* the script:

```
run { execute script b_whole; }
```

See Also: For **execute script** command syntax, see "[run](#)" on page 11-124.

To replace a stored script:

1. Start RMAN and connect to the recovery catalog database and target database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. Issue a **replace script** command to replace a stored script with another. For example, this command replaces script `b_whole` with the following:

```
replace script b_whole {
  allocate channel ch1 type 'sbt_tape';
  backup database;
}
```

See Also: For **replace script** command syntax, see "[replaceScript](#)" on page 11-97.

To delete a stored script:

1. Start RMAN and connect to the recovery catalog database. For example, enter:

```
% rman catalog rman/rman@rcat
```

2. Issue a **delete script** command to delete the specified stored script:

```
delete script 'b_whole';
```

See Also: For **delete script** command syntax, see "[deleteScript](#)" on page 11-65.

To print a stored script to a message log:

1. Start RMAN and connect to the recovery catalog database and target database, specifying the **log** argument if you want to print to a message log. For example, enter the following to specify `rman_log`:

```
% rman target / catalog rman/rman@rcat log rman_log
```

2. Issue a **print script** command to write the script to the log:

```
print script b_whole;
```

3. Host out to the O/S and use an O/S utility to view the script. For example, enter:

```
RMAN> host;  
% vi rman_log
```

See Also: For **replace script** command syntax, see "[printScript](#)" on page 11-86.

To obtain a listing of all stored scripts:

1. Start SQL*Plus and connect to the recovery catalog database. For example, to connect to database RCAT enter:

```
% sqlplus rman/rman@rcat
```

2. Issue a **SELECT** statement on the `RC_STORED_SCRIPT` view:

```
SQL> SELECT script_name FROM rc_stored_script;  
SCRIPT_NAME  
-----  
backupdb  
binc  
bincl  
3 rows selected.
```

See Also: For information about the RC_STORED_SCRIPT view, see "[RC_STORED_SCRIPT](#)" on page 12-25.

Resynchronizing the Recovery Catalog

When RMAN performs a *resynchronization*, it compares the recovery catalog to either the current control file of the target database or a backup control file and updates it with information that is missing or changed.

Resynchronizations can be *full* or *partial*. In a partial resynchronization, RMAN reads the current control file to update changed information, but does not resynchronize metadata about the database *physical schema*: datafiles, tablespaces, redo threads, rollback segments (only if the database is open), and online redo logs. In a full resynchronization, RMAN updates all changed records, including those for the database schema.

Note: Although RMAN performs partial resynchronizations when the control file is a backup, it will not perform full resynchronizations.

When resynchronizing, RMAN creates a snapshot control file, compares the recovery catalog to the snapshot, and updates the catalog with information that is missing or changed. RMAN performs partial or full resynchronizations automatically as needed when you execute certain commands (see "[resync](#)" on page 11-118 for more information). To ensure a full resynchronization, issue a **resync catalog** command.

This section contains the following topics:

- [What Is Resynchronized?](#)
- [When Should You Resynchronize?](#)
- [How Do You Resynchronize?](#)

What Is Resynchronized?

[Table 6–1](#) describes the types of records that RMAN resynchronizes.

Table 6–1 Records Updated during a Resynchronization

Records	Description
log history	created when an online redo log switch occurs.

Table 6–1 Records Updated during a Resynchronization

Records	Description
archived redo logs	associated with archived logs that were created by archiving an online log, copying an existing archived redo log, or restoring an archived redo log backup set. RMAN tracks this information so that it knows which archived logs it should expect to find.
backup history	associated with backup sets, backup pieces, backup set members, and file copies. The resync catalog command updates these records when a backup or copy command is executed.
physical schema	associated with datafiles and tablespaces. If the target database is open, then rollback segment information is also updated. Physical schema information in the recovery catalog is updated only when the target has the current control file mounted. If the target database has mounted a backup control file, a freshly created control file, or a control file that is less current than a control file that was seen previously, then physical schema information in the recovery catalog is <i>not</i> updated. Physical schema information is also not updated when you use the resync catalog from controlfilecopy command.

When Should You Resynchronize?

RMAN automatically performs full or partial resynchronizations as needed in certain situations. Execution of the following commands performs a full or partial resynchronization (depending on whether the schema metadata has changed) automatically when the target database control file is mounted and the recovery catalog database is available when the command is executed:

- **backup**
- **copy**
- **crosscheck**
- **delete expired backupset**
- **duplicate**
- **list**
- **recover**
- **report**
- **restore**
- **switch**

Perform manual resynchronizations in the following scenarios.

Resynchronizing when the Recovery Catalog is Unavailable If the recovery catalog is unavailable when you issue **backup** or **copy** commands, then open the catalog database later and resynchronize it manually using the **resync catalog** command.

Resynchronizing when You Run in ARCHIVELOG Mode If you are running in ARCHIVELOG mode, resynchronize the recovery catalog regularly because the recovery catalog is *not* updated automatically when a redo log switch occurs or when a redo log is archived. Instead, Oracle stores information about log switches and archived redo logs in the control file. You must propagate this information periodically into the recovery catalog.

How frequently you resynchronize the recovery catalog depends on the rate at which Oracle archives redo logs. The cost of the operation is proportional to the number of records in the control file that have been inserted or changed since the previous resynchronization. If no records have been inserted or changed, then the cost of resynchronization is very low. Thus, you can perform this operation frequently—for example, hourly—without incurring undue costs.

Resynchronizing After Physical Database Changes Resynchronize the recovery catalog after making any change to the physical structure of the target database. As with redo log archive operations, the recovery catalog is *not* updated automatically when a physical schema change is made.

A physical schema change occurs when you:

- Add or drop a tablespace.
- Add a new datafile to an existing tablespace.
- Add or drop a rollback segment.

When resynchronizing from the current control file, RMAN automatically detects the records in the control file that have been updated and resynchronizes only those records. If the target database is open, then RMAN also updates information about rollback segments in the recovery catalog (this information is used for TSPITR).

When resynchronizing from a backup control file, RMAN *does not* verify that the backup pieces or file copies actually exist. Thus, you may need to use the **change ... crosscheck** and **crosscheck** commands to remove records for files that no longer exist.

How Do You Resynchronize?

Issue the **resync catalog** command to force a full resynchronization of the recovery catalog.

To perform a full resynchronization of the recovery catalog:

1. Open the recovery catalog database (if it is not already open). For example, enter:

```
SQL> STARTUP pfile=initRCAT.ora
```

2. Use RMAN to connect to the target and recovery catalog databases. For example, enter:

```
% rman target sys/change_on_install@prod1 catalog rman/rman@rcat
```

3. Mount the target database if it is not already mounted:

```
startup mount;
```

4. Issue the **resync catalog** command:

```
resync catalog;
```

```
RMAN-03022: compiling command: resync  
RMAN-03023: executing command: resync  
RMAN-08002: starting full resync of recovery catalog  
RMAN-08004: full resync complete
```

See Also: For **resync catalog** command syntax, see ["resync"](#) on page 11-118.

Managing Records in the Control File

The recovery catalog is dependent on the target database control file for information. Consequently, the currency of the information in the control file determines the effectiveness of the recovery catalog.

The size of the target database's control file grows depending on the number of:

- Backups that you perform.
- Archived redo logs that Oracle generates.
- Days that this information is stored in the control file.

You can use the `CONTROL_FILE_RECORD_KEEP_TIME` parameter to specify the minimum number of days that Oracle keeps this information in the control file. Entries older than the number of days specified are candidates for overwrites by

newer information. The larger the `CONTROL_FILE_RECORD_KEEP_TIME` setting, the larger the control file.

At a minimum, resynchronize your recovery catalog at intervals less than the `CONTROL_FILE_RECORD_KEEP_TIME` setting, because after the number of days specified in this parameter, Oracle will overwrite the information in the control file with the most recently created information. If you have not resynchronized the recovery catalog, and Oracle has overwritten the information, then this information cannot be propagated to the recovery catalog.

Note: The maximum size of the control file is port-specific. See your O/S-specific Oracle documentation.

See Also: For more information about the `CONTROL_FILE_RECORD_KEEP_TIME` parameter, see the *Oracle8i Reference*. For more information about managing control files for backup and recovery, see "[Managing the Control File](#)" on page 2-3. For more detailed information on all aspects of control file management, see the *Oracle8i Administrator's Guide*.

Cataloging O/S Backups

You can make RMAN aware of the existence of file copies that are created via means other than RMAN. This section contains the following topics:

- [What Is Cataloging?](#)
- [When Should You Catalog an O/S Backup?](#)
- [How Do You Catalog an O/S Backup?](#)

What Is Cataloging?

If you do not make backups or image copies by using RMAN commands, then the recovery catalog has no record of them. You must manually notify RMAN when you make backups with an O/S utility. Use the RMAN **catalog** command to:

- Add information about an O/S datafile copy, archived redo log copy, or control file copy to the recovery catalog and control file.
- Catalog a datafile copy as a level 0 backup, thus enabling you to perform an incremental backup later using the datafile copy as the base of an incremental backup strategy.

- Record the existence of backups of version 8 databases created before RMAN was installed.
- Record the existence of Oracle7 backups of read-only or offline normal files made before migrating to Oracle8i.

When Should You Catalog an O/S Backup?

Catalog backups whenever:

- You make a copy via an O/S utility.
- You want to catalog Oracle7 O/S backups that were made offline normal or read-only prior to migration.

Note: You cannot re-catalog backup sets or pieces.

Cataloging Hot and Cold O/S Backups Whenever you make a cold O/S backup, e.g., by using the UNIX `cp` command to copy a datafile, make sure to catalog it. Oracle8i supports the `ALTER TABLESPACE BEGIN/END BACKUP` command, which allows open database O/S backups. Although RMAN does not create such backups, you can add them to the recovery catalog so that RMAN is aware of them.

For a backup to be cataloged, it must be:

- Accessible on disk.
- A complete image copy of a single file.
- Either a consistent or inconsistent whole database, tablespace, datafile, control file, or archived redo log backup. RMAN treats all such backups as datafile copies.

For example, if you store datafiles on mirrored disk drives, you can create an O/S copy by breaking the mirror. In this scenario, use the **catalog** command to notify RMAN of the existence of the operating system copy after breaking the mirror. Before reforming the mirror, issue a **change ... uncatalog** command to notify RMAN that the file copy is being deleted.

Cataloging Oracle7 O/S Backups RMAN cannot catalog Oracle7 files, except in the following special circumstances. During the migration from Oracle7 to Oracle8i, you shut down your Oracle7 database cleanly prior to running the migration utility. At this time, you can take O/S backups of your Oracle7 datafiles and catalog the backups with RMAN: Oracle accepts them because no redo from the old database is

required to recover them. RMAN can then restore those backups in your Oracle8i database if no other backups exist.

Following is a scenario that generates an Oracle7 backup that you can catalog with RMAN:

1. Take a datafile offline clean or read-only.
2. Close the database. You must not have made the files read-write again before closing the database.
3. Make an O/S backup of the offline or read-only files.
4. Migrate the database to Oracle8i.

The pre-migration backups are identical to the backups that would be taken prior to migration, and may also be cataloged with RMAN.

How Do You Catalog an O/S Backup?

Use the **catalog** command to propagate information about O/S backups to the recovery catalog.

To catalog an O/S backup:

1. Make a backup via an O/S utility. This example backs up a datafile.

```
% cp $ORACLE_HOME/dbs/sales.f $ORACLE_HOME/backup/sales.bak';
```

2. Use RMAN to connect to the target and recovery catalog databases. For example, enter:

```
% rman target change_on_install@prod1 catalog rman/rman@rcat
```

3. Issue the **catalog** command. For example, enter:

```
catalog datafilecopy '$ORACLE_HOME/backup/sales.bak';
```

```
RMAN-03022: compiling command: catalog
RMAN-03023: executing command: catalog
RMAN-08050: cataloged datafile copy
RMAN-08513: datafile copy filename=/oracle/backup/sales.bak recid=121 stamp=342972501
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete
```

See Also: For **catalog** command syntax, see ["catalog"](#) on page 11-32.

Backing Up and Recovering the Recovery Catalog

Include the recovery catalog in your backup and recovery strategy. If you do not back up your recover catalog and a disk crash occurs, you may lose some or all of your data. Avoid this unpleasant scenario by thinking about how to back up and recover the recovery catalog.

This section contains the following topics:

- [Backing Up the Recovery Catalog](#)
- [Recovering the Recovery Catalog](#)
- [Re-Creating the Recovery Catalog](#)

Backing Up the Recovery Catalog

When developing a strategy for backing up the recovery catalog, follow these guidelines:

- [Make Regular Backups](#)
- [Use the Appropriate Method for Physical Backups](#)
- [Store the Recovery Catalog in an Appropriate Place](#)
- [Make Logical Backups](#)

Make Regular Backups

Back up the recovery catalog with the same frequency that you back up your target database. For example, if you make a weekly whole database backup of your target database, then back up your recovery catalog immediately after each target database backup. The backed up catalog will have a record of the target backup preceding it, so if you are forced to restore the catalog you will also be able to use it to restore the target database backup.

Use the Appropriate Method for Physical Backups

When backing up the recovery catalog, you have a choice of making O/S or RMAN backups. The advantage of making O/S backups is that you can restore them using O/S methods without relying on RMAN or a recovery catalog. For O/S backup procedures, see [Chapter 13, "Performing Operating System Backups"](#).

If you use RMAN to back up the recovery catalog database, consider how you will restore the recovery catalog in case of failure. For example, you can:

- Create a second recovery catalog in a separate database, e.g., the production database you are backing up. To learn how to create a catalog, see "[Creating the Recovery Catalog](#)" on page 6-2.
- Use the recovery catalog control file as the RMAN metadata repository for the recovery catalog, and make frequent backups of the recovery catalog control file using the `ALTER DATABASE BACKUP CONTROLFILE TO 'filename'` statement. To learn how to back up the control file using O/S methods, see "[Performing Control File Backups](#)" on page 13-11.

Store the Recovery Catalog in an Appropriate Place

Never store a recovery catalog containing the RMAN metadata for a database in the same database as your target database. For example, do not store the catalog for database PROD1 in PROD1. A recovery catalog for PROD1 is only effective if it is separated from the data that it is designed to protect. If PROD1 suffers a total media failure, and the recovery catalog data for PROD1 is also stored in PROD1, then you have no catalog to aid in recovery.

This rule is especially important when you want to back up a recovery catalog database. Take a case in which database RCAT contains the recovery catalog metadata for target database PROD1. You decide to use a recovery catalog to back up RCAT, but are not sure where to store this catalog. If you store the catalog containing metadata for RCAT in database RCAT itself, then if you lose RCAT because of a media failure you will have difficulty restoring RCAT *and* will leave PROD1 unprotected.

Make Logical Backups

Use the Export utility to make convenient backups of your recovery catalog data. An export of the catalog allows the most flexibility when the recovery catalog must be restored, because it can be restored to any existing Oracle8i database.

To make a logical export of the recovery catalog from the command line:

1. Execute the export at the command line, making sure to do the following:
 - a. Connect as the owner of the recovery catalog.
 - b. Specify the OWNER option.
 - c. Specify an output file.

For example, if the owner of the catalog in database PROD1 is RMAN, you can issue the following at the UNIX command line to export the catalog to file `cat.dmp`:

```
% exp rman/rman@prod1 file=cat.dmp owner=rman
```

2. Examine the output to make sure you were successful:

```
Export terminated successfully without warnings.
```

See Also: To learn how to use the Export utility, see *Oracle8i Utilities*. To learn how to make O/S backups, see [Chapter 13, "Performing Operating System Backups"](#).

Recovering the Recovery Catalog

The method you use to recover the catalog depends on the method you use to back it up. You have these options:

- [Recovering the Catalog Using O/S Methods](#)
- [Recovering the Catalog Using RMAN](#)
- [Importing a Logical Backup of the Catalog](#)

Recovering the Catalog Using O/S Methods

If you backed up the recovery catalog using operating system commands, then restore the backup of the catalog using O/S commands and issue SQL*Plus commands to recover it. The method you use depends on whether you are recovering the entire recovery catalog database or only the tablespace in which the recovery catalog is stored. For procedures, see [Chapter 14, "Performing Operating System Recovery"](#).

Recovering the Catalog Using RMAN

If you use RMAN to recover the catalog, then see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#) for the relevant procedures.

Importing a Logical Backup of the Catalog

If you used Export to make a logical backup of the recovery catalog, then use Import to recover it. If a media failure damages your recovery catalog database, do the following:

1. Create a new user in another database. For the recommended SQL syntax for creating a new user in a recovery catalog database, see "[Creating the Recovery Catalog](#)" on page 6-2.
2. Import the catalog data from the Export file. Execute the import at the command line, making sure to do the following:
 - a. Connect as the new owner of the recovery catalog.
 - b. Specify the old owner using the FROMUSER parameter.
 - c. Specify the new owner using the TOUSER parameter.
 - d. Specify the import file.

For example, assume the following:

- The old owner of the catalog in database PROD1 is RMAN.
- The user in the new recovery catalog database NEW_CAT is RCAT.
- The file containing the export of the catalog is `cat.dmp`.

```
imp userid=rcat/rcat_pwd@new_cat file=cat.dmp fromuser=rman touser=rcat
```

3. Use the imported catalog data for restore and recovery of your target database.

See Also: To learn how to use the Import utility, see *Oracle8i Utilities*.

Re-Creating the Recovery Catalog

If the recovery catalog database is lost or damaged, and recovery of the recovery catalog database through the normal Oracle recovery mechanisms is not possible, then you must re-create the catalog.

You have two options for partially re-creating the contents of the old catalog:

- Issue **catalog** commands to re-catalog archived redo logs, backup control files, and datafile copies.
- Use the **resync catalog from controlfilecopy** command to extract information from a backup control file and rebuild the recovery catalog from it.

You can re-create information about backup sets *only* by using the **resync catalog from controlfilecopy** command, because the **catalog** command does not support re-cataloging of backup pieces or backup sets. RMAN does not verify that the files being re-cataloged still exist, so the resynchronization may add records for files that no longer exist. Remove such records by issuing **change ... crosscheck** or **crosscheck backup** commands.

Upgrading the Recovery Catalog

If you use a version of the recovery catalog that is older than that required by the RMAN executable, then you must upgrade it. For example, you must upgrade the catalog if you use an 8.1 version of RMAN with an 8.0 recovery catalog.

You will receive an error on issuing **upgrade catalog** if the recovery catalog is already at a version greater than that required by the RMAN executable. RMAN permits the **upgrade catalog** command to be run if the recovery catalog is current and does not require upgrading, however, so that you can re-create packages at any time if necessary. Check the message log for error messages generated during the upgrade.

To upgrade the recovery catalog:

1. Use RMAN to connect to the target and recovery catalog databases. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

```
RMAN-06008: connected to recovery catalog database
RMAN-06186: PL/SQL package rcat.DBMS_RCVCAT version 08.00.04 in RCVCAT
           database is too old
```

2. Issue the **upgrade catalog** command:

```
RMAN> upgrade catalog

RMAN-06435: recovery catalog owner is rman
RMAN-06442: enter UPGRADE CATALOG command again to confirm catalog upgrade
```

3. Enter the **update catalog** command again to confirm:

```
RMAN> upgrade catalog

RMAN-06408: recovery catalog upgraded to version 08.01.05
RMAN-06452: DBMS_RCVMAN package upgraded to version 08.01.05
RMAN-06452: DBMS_RCVCAT package upgraded to version 08.01.03
```

See Also: For **upgrade catalog** command syntax, see "[upgradeCatalog](#)" on page 11-148. For complete compatibility and migration information, see *Oracle8i Migration*.

Dropping the Recovery Catalog

If you do not want to maintain a recovery catalog, then you can drop the recovery catalog schema from the tablespace. The **drop catalog** command deletes all information from the recovery catalog. Hence, if you have no backups of the recovery catalog schema, then all backups of all target databases managed by this catalog become unusable.

The **drop catalog** command is not appropriate for "unregistering" a single database from a catalog that registers multiple target databases. If you try to delete the information for one target database by dropping the catalog, you delete the information for all target databases.

To drop the recovery catalog schema:

1. Use RMAN to connect to the target and recovery catalog databases.

```
% rman catalog rman/rman@rcvcat
```

2. Issue the **drop catalog** command twice to confirm:

```
RMAN> drop catalog
```

```
RMAN-06435: recovery catalog owner is rman
```

```
RMAN-06436: enter DROP CATALOG command again to confirm catalog removal
```

```
RMAN> drop catalog
```

Note: Even after you drop the recovery catalog, the control file still contains records about your backups. To purge those records, re-create the control file.

See Also: For **drop catalog** command syntax, see "[dropCatalog](#)" on page 11-68.

Managing RMAN Metadata Without a Recovery Catalog

RMAN works perfectly well without a recovery catalog. In fact, the recovery catalog receives its information from the control file. If you choose not to use a recovery catalog, however, you must perform the following tasks:

- [Maintaining the Control File Metadata](#)
- [Backing Up the Control File](#)

See Also: For a conceptual overview of the control file, see *Oracle8i Concepts*. For a description of the importance of the control file for backup and recovery, see [Chapter 6, "Managing Recovery Manager Metadata"](#). For more detailed information on managing the control file, see *Oracle8i Administrator's Guide*.

Maintaining the Control File Metadata

RMAN provides several commands that allow you to check and delete records of backups as well as physically remove backups and copies. ["Maintaining RMAN Metadata"](#) on page 6-4 provides a complete description of these maintenance procedures. Most of these commands work whether or not you use a recovery catalog.

For a list of the commands that require a catalog, see ["Consequences of Using the Control File for RMAN Metadata"](#) on page 5-6. You do not lose much in the way of maintenance functionality when you do not use a recovery catalog, because you can:

- Use the **restore ... validate** or **validate backupset** commands to test whether backups and copies are available and can be restored. See ["Validating the Restore of Backups and Copies"](#) on page 6-18 for a description of this procedure.
- Use **change ... delete** to remove outdated records from the control file and physically delete unwanted or corrupted backups and copies from the O/S. See ["Deleting Backups and Copies and Updating Their Status in the RMAN Metadata"](#) on page 6-13 for a description of this procedure.

Backing Up the Control File

If you use the control file as the sole repository of the RMAN metadata, maintain alternate control files through multiplexing or O/S mirroring and back up the control file frequently. If you should lose your control file and do not have a backup, you will lose all information about RMAN backups and copies contained in the file.

To learn how to multiplex and mirror the control file, see ["Managing the Control File"](#) on page 2-3. To learn how to make O/S backups of the control file, see ["Backing Up the Control File to a Physical File"](#) on page 13-12 and ["Backing Up the Control File to a Trace File"](#) on page 13-12.

Generating Lists and Reports with Recovery Manager

This chapter describes how to use Recovery Manager to make useful lists and reports of your backups and image copies, and includes the following topics:

- [Using Lists and Reports in Your Backup and Recovery Strategy](#)
- [Generating Lists](#)
- [Generating Reports](#)
- [List and Report Scenarios](#)

Using Lists and Reports in Your Backup and Recovery Strategy

Use the **report** and **list** commands to determine what you have backed up or copied as well as what you need to back up or copy. The information, which is available to you whether or not you use a recovery catalog, is extremely helpful in developing an effective backup strategy. Refer to [Chapter 6, "Managing Recovery Manager Metadata"](#) to learn how to keep the RMAN metadata current.

The **list** command displays all RMAN backups (both backup sets and proxy copies) and image copies, while the **report** command performs more complex analysis. For example, generate a report on which datafiles need a backup and which backup pieces are obsolete. Oracle writes the output from the **report** and **list** commands to either the screen or a log file.

See Also: For **list** command syntax, see "[list](#)" on page 11-76. For **report** command syntax, see "[report](#)" on page 11-102.

Generating Lists

The **list** command queries the recovery catalog or control file and produces a listing of its contents. The primary purpose of the **list** command is to determine which backups or copies are available. For example, list:

- All backups (backup sets and proxy copies) or image copies recorded in the RMAN metadata.
- Backups or image copies of a specified database, tablespace, datafile, archived redo log, or control file.
- Backups and image copies restricted by time, pathname, device name, tag, or recoverability.
- Incarnations of a specified database or of all databases known to the recovery catalog.

Use the RMAN metadata to determine what you need to back up. In particular, ensure that:

- The STATUS columns of the output tables list all backups and image copies as available.
- All datafiles, archived redo logs, and control files that you want backed up are included in the output.
- The backups and copies are recent.

To generate a list of image copies and backups:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log '/oracle/log/mlog.f'
```

2. Execute **list copy** and **list backup** commands. If you do not specify the **of listObjList** parameter, **list** defaults to **of database**:

```
list copy of database archivelog all; # lists datafiles and archived redo logs
list backup; # lists backup sets, backup pieces, and proxy copies
```

3. Examine the output. See "list" on page 11-76 for an explanation of the various column headings in the **list** output. Following is sample output:

```
list copy of database archivelog all;
```

List of Datafile Copies

Key	File S	Completion time	Ckp SCN	Ckp time	Name
1262	1 A	18-AUG-98	219859	14-AUG-98	/vobs/oracle/dbs/copy/tbs_01.f

List of Archived Log Copies

Key	Thrd	Seq	S	Completion time	Name
789	1	1	A	14-JUL-98	/vobs/oracle/work/arc_dest/arcr_1_1.arc
790	1	2	A	11-AUG-98	/vobs/oracle/work/arc_dest/arcr_1_2.arc
791	1	3	A	12-AUG-98	/vobs/oracle/work/arc_dest/arcr_1_3.arc

```
list backup;
```

List of Backup Sets

Key	Recid	Stamp	LV	Set Stamp	Set Count	Completion Time
1174	12	341344528	0	341344502	16	14-AUG-98

List of Backup Pieces

Key	Pc#	Cp#	Status	Completion Time	Piece Name
1176	1	1	AVAILABLE	14-AUG-98	/vobs/oracle/dbs/0ga5h07m_1_1

Controlfile Included

Ckp SCN	Ckp time
-----	-----

```
219857      14-AUG-98
```

List of Datafiles Included

File Name	LV	Type	Ckp	SCN	Ckp Time
1 /vobs/oracle/dbs/tbs_01.f	0	Full	199843		14-AUG-98
2 /vobs/oracle/dbs/tbs_02.f	0	Full	199843		14-AUG-98

To generate a list of copies and backups restricted by object or other conditions:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcvcat
```

2. To restrict by object, use **list copy** or **list backup** with the **of listObjList** condition. For example, enter:

```
list backup of database;      # lists backups of all files in database
list copy of datafile '/oracle/dbs/tbs_1.f'; # lists copy of specified datafile
list backup of tablespace SYSTEM; # lists all backups of SYSTEM tablespace
list copy of archivelog all; # lists all archived redo logs and copies of logs
list backup of controlfile; # lists all control file backups
```

You can also restrict your search by specifying a combination of tag, device type, filename pattern, or time options. For example, enter:

```
list backup tag 'weekly_full_db_backup'; # by tag
list copy of datafile '/oracle/dbs/tbs_1.f' type 'sbt_tape'; # by type
list backup like '/oracle/backup/tbs_4%'; # by filename pattern
list backup of archivelog until time 'SYSDATE-30'; # by time
list copy of datafile 2 completed between '10-DEC-1998' and '17-DEC-1998'; # by time
```

3. Examine the output. For example, following is sample output for a list of copies of datafile 1:

```
RMAN> list copy of datafile 1;
```

```
RMAN-03022: compiling command: list
```

List of Datafile Copies

Key	File S	Completion time	Ckp SCN	Ckp time	Name
3	1	A 18-DEC-98	114148	18-DEC-98	/vobs/oracle/dbs/df1.bak

See Also: For *listObjList* syntax, see "[listObjList](#)" on page 11-84. See "[List Output](#)" on page 11-78 for an explanation of the various columns in the **list** output.

Generating Reports

To gain more detailed information from the RMAN metadata, generate a report. Use the **report** command to answer questions such as the following:

- Which files need a backup?
- Which files have not been backed up recently?
- Which files are listed as unrecoverable?
- Which backups or copies are obsolete and can be deleted?
- What was the physical schema of the database at some previous time?
- Which backups and copies are on disk and which are on tape?

Note: For the report to be accurate, the RMAN metadata must be current and you must have used the **change**, **uncatalog**, and **crosscheck** commands appropriately to update the status of all backups and copies is correct. To learn how to maintain the metadata, see "[Maintaining RMAN Metadata](#)" on page 6-4.

The information that you glean from reports can be extremely important for your backup and recovery strategy. In particular, use the **report need backup** and **report unrecoverable** commands regularly to ensure that:

- The necessary backups are available to perform recovery.
- Recovery can be performed within a reasonable length of time, i.e., that the mean time to recovery (MTTR) is minimized.

To report on objects that need a backup:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

To write the output to a log file, specify a file at startup:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. If necessary, issue **crosscheck** commands to update the status of backups and **change ... crosscheck** commands to update the status of image copies (if you

want to specify image copies by primary key, issue a **list** command to obtain the keys).

Following is a possible crosscheck session:

```
# must allocate maintenance channel for crosscheck
allocate channel for maintenance type disk;
crosscheck backup; # crosschecks all backups
change datafile copy 100,101,102,103,104,105,106,107 crosscheck; # specified by key
change archivelog copy 50,51,52,53,54 crosscheck; # specified by key
release channel;
```

3. Use the **need backup** option to identify which datafiles need a new backup, restricting the report by a threshold number of days or incremental backups. RMAN considers any backups older than the **days** parameter value as needing a new backup because backups require **days** worth of archived redo logs for recovery.

For example, enter:

```
report need backup days = 7 database; # needs at least 7 days of logs to recover
report need backup days = 30 tablespace system;
report need backup days = 14 datafile '/oracle/dbs/tbs_5.f';
```

You can also specify the **incremental** parameter. If complete recovery of a datafile requires more than the specified number of incremental backups, then RMAN considers it in need of a new backup. For example, enter:

```
report need backup incremental = 1 database;
report need backup incremental = 3 tablespace system;
report need backup incremental = 5 datafile '/oracle/dbs/tbs_5.f';
```

4. Examine the report and back up those datafiles requiring a new backup.

See Also: For an explanation of the various column headings in the **report** output, see "[Report Output](#)" on page 11-106.

To report on backups that are obsolete:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. If necessary, issue **crosscheck** commands to update the status of backups and **change ... crosscheck** commands to update the status of image copies (if you want to specify copies by primary key, issue a **list** command to obtain them):

```
allocate channel for maintenance type disk;
crosscheck backup;
change datafile copy 100,101,102,103,104,105,106,107 crosscheck;
change archivelog copy 50,51,52,53,54 crosscheck;
release channel;
```

3. Use the **obsolete** option to identify which backups are obsolete because they are no longer needed for recovery. The **redundancy** parameter specifies the minimum level of redundancy considered necessary for a backup or copy to be obsolete (if you do not specify the parameter, **redundancy** defaults to 1).

A datafile copy is obsolete if at least *integer* more recent backups of this file exist; a datafile backup set is obsolete if at least *integer* more recent backups or image copies of each file contained in the backup set exist. For example, enter:

```
# Lists backups or copies that have at least 2 more recent backups or copies
report obsolete redundancy = 2;
```

Use the *untilClause* to use make the redundancy check for backups sets or copies that are more recent, but not later than the specified time, SCN, or log sequence number:

```
# Obsolete if there are at least 2 copies/backups that are no more than 2 weeks old.
report obsolete redundancy = 2 until time 'SYSDATE-14';
report obsolete until scn 1000;
report obsolete redundancy = 3 until logseq = 121 thread = 1;
```

4. Use the **orphan** option to list which backups and copies are unusable because they belong to a incarnation that is not a direct predecessor of the current incarnation:

```
report obsolete orphan;
```

For an explanation of orphaned backups, see ["Reporting on Orphaned Backups"](#) on page 4-23.

5. Examine the report and delete those backups that are obsolete.

```
RMAN> report obsolete;
```

```
RMAN-03022: compiling command: report
Report of obsolete backups and copies
Type                Recid Stamp      Filename
-----
```

```

Backup Set          4          345390311
Backup Piece        4          345390310 /oracle/dbs/04a9cf76_1_1

RMAN> allocate channel for delete type disk;

RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: delete
RMAN-08500: channel delete: sid=11 devtype=DISK

RMAN> change backuppiece '/oracle/dbs/04a9cf76_1_1' delete;

RMAN-03022: compiling command: change
RMAN-03023: executing command: change
RMAN-08073: deleted backup piece
RMAN-08517: backup piece handle=/oracle/dbs/04a9cf76_1_1 recid=4 stamp=345390310
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete

```

To report on backups that are unrecoverable:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. Use the **unrecoverable** option of the **report** command to determine which datafiles have had an unrecoverable operation performed against an object residing in the datafile since its last backup.

```
report unrecoverable database; # Examines all datafiles.
```

To report the database schema at a specified point in time:

You must use a recovery catalog for reporting on database schema at a past time, SCN, or log sequence number.

1. Start RMAN and connect to the target database and the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```


2. Issue **report schema** for a list of all the datafiles and tablespaces in the target database at the current time:

```
report schema;
```

Use the *untilClause* to specify a past time, SCN, or log sequence number:

```
report schema at time 'SYSDATE-14';
report schema at scn 1000;
report schema at logseq 100;
```

3. Examine the report. For example, here is sample output:

```
RMAN> report schema at scn 1000;
RMAN-03022: compiling command: report
```

```
Report of database schema
File K-bytes    Tablespace          RB segs Name
-----
1          35840 SYSTEM             YES    /vobs/oracle/dbs/tbs_01.f
2           978 SYSTEM             YES    /vobs/oracle/dbs/tbs_02.f
3           978 TBS_1              NO     /vobs/oracle/dbs/tbs_11.f
4           978 TBS_1              NO     /vobs/oracle/dbs/tbs_12.f
5           978 TBS_2              NO     /vobs/oracle/dbs/tbs_21.f
6           978 TBS_2              NO     /vobs/oracle/dbs/tbs_22.f
```

This type of information is useful for incomplete recovery because you can determine the schema of the database for the time that you want to recover to.

See Also: For **report** command syntax, see "[report](#)" on page 11-102. For **list** command syntax, see "[list](#)" on page 11-76.

List and Report Scenarios

Following are some examples of list and report generation:

- [Makings Lists of Backups and Copies](#)
- [Using Lists to Determine Obsolete Backups and Copies](#)
- [Reporting Datafiles Needing Backups](#)
- [Reporting Unrecoverable Datafiles](#)
- [Reporting Obsolete Backups and Copies](#)
- [Deleting Obsolete Backups](#)
- [Generating Historical Reports of Database Schema](#)

Makings Lists of Backups and Copies

Use the **list** command to query the contents of the recovery catalog or the target database control file if no recovery catalog is used. You can use several different parameters to qualify your lists.

The following example lists all backups of datafiles in tablespace TBS_1 that were made since November 1, 1998:

```
list backup of tablespace tbs_1 completed before 'Nov 1 1998 00:00:00';
```

The following example lists all backup sets or proxy copies on media management devices:

```
list backup of database device type 'sbt_tape';
```

The following example lists all copies of datafile 2 using the tag weekly_df2_copy that are in the copy sub-directory:

```
list copy of datafile 2 tag weekly_df2_copy like '/copy/%';
```

This example lists all database incarnations registered in the recovery catalog:

```
list incarnation of database;
```

Using Lists to Determine Obsolete Backups and Copies

Use the **list** command to determine which copies and backups can be deleted. For example, if you created a full backup of the database on November 2, and you know you will not need to recover the database to an earlier date, then the backups and image copies listed in the following report can be deleted:

```
list backup of database completed before 'Nov 1 1998 00:00:00';  
list copy completed before 'Nov 1 1998 00:00:00';
```

Reporting Datafiles Needing Backups

The following command reports all datafiles in the database that would require the application of three or more incremental backups to be recovered to their current state:

```
report need backup incremental 3 database;
```

The following command reports all datafiles from tablespace SYSTEM that have not had a backup (full or incremental) in five or more days:

```
report need backup days 5 tablespace system;
```

The following command reports which of datafiles 1 - 5 need backups because they do not have two or more backups or copies stored on tape:

```
report need backup redundancy 2 datafile 1,2,3,4,5 device type 'sbt_tape';
```

Reporting Unrecoverable Datafiles

The following example reports on all datafiles on tape that need a new backup because they contain unlogged changes that were made after the last full or incremental backup.

```
report unrecoverable database device type 'sbt_tape';
```

Reporting Obsolete Backups and Copies

The following command reports all backups and copies on disk that are obsolete because three more recent backups or copies are already available:

```
report obsolete redundancy 3 device type disk;
```

The following command reports all backups on tape that are obsolete because at least two backups already exist that were made no more than one week ago:

```
report obsolete redundancy 2 until time 'SYSDATE-7' device type 'sbt_tape';
```

The following command reports which datafiles are obsolete because they belong to a database incarnation that is not a direct predecessor of the current incarnation:

```
report obsolete orphan;
```

For example, if a parent incarnation has two children, incarnation A and incarnation B, and B is the current incarnation, the children of A will be orphans

Deleting Obsolete Backups

In this scenario, assume that you want to delete the following:

- Datafile copies for which there are at least two more recent copies.
 - Datafile backups for which there are at least two more recent backups or image copies of each datafile contained in the backup set.
1. Generate a report with redundancy set to 2:

```
report obsolete redundancy 2;
```

```
RMAN-03022: compiling command: report
```

Report of obsolete backups and copies

Type	Recid	Stamp	Filename
Datafile Copy	23	345392880	/vobs/oracle/dbs/tbs_01.copy
Datafile Copy	22	345392456	/vobs/oracle/dbs/tbs_01_copy.f
Backup Set	31	345552065	
Backup Piece	31	345552061	/vobs/oracle/dbs/0va9hd5o_1_1
Backup Set	23	345399397	
Backup Piece	23	345399391	/vobs/oracle/dbs/0ma9co2p_1_1
Backup Set	20	345397468	
Backup Piece	20	345397464	/vobs/oracle/dbs/0ka9cm6l_1_1

- Issue **change ... delete** commands for the copies and backups. Use the filenames or issue a **list** command to obtain the primary keys:

```
allocate channel for delete type disk;
change backuppiece '/vobs/oracle/dbs/0va9hd5o_1_1', '/vobs/oracle/dbs/0ma9co2p_1_1',
'/vobs/oracle/dbs/0ka9cm6l_1_1' delete;
change datafilecopy '/vobs/oracle/dbs/tbs_01.copy', '/vobs/oracle/dbs/tbs_01_copy.f'
delete;
release channel;
```

- Optionally, start a SQL*Plus session and query the recovery catalog views to check that the status of the copies and backup pieces has been changed to **deleted** (the **list** output does not display files with **deleted** status):

```
SQL> SELECT handle FROM rc_backup_piece WHERE status = 'D';
SQL> SELECT name FROM rc_datafile_copy WHERE status = 'D';
```

Generating Historical Reports of Database Schema

The following commands reports the database schema in the present, a week ago, two weeks ago, and a month ago:

```
report schema;
report schema at time 'SYSDATE-7';
report schema at time "TO_DATE('12/20/98','MM/DD/YY')";
```

The following command reports on the database schema at SCN 953:

```
report schema at scn 953;
```

The following command reports on the database schema at log sequence number 12 of thread 2:

```
report schema at logseq 12 thread 2;
```

Making Backups and Copies with Recovery Manager

This chapter describes how to use Recovery Manager to manage backup and copy operations, and includes the following topics:

- [Making Backups](#)
- [Making Image Copies](#)
- [Monitoring Backup and Copy Operations](#)
- [Backup and Copy Scenarios](#)

Making Backups

Perform backups of any of the following objects using the RMAN **backup** command:

- Database (all datafiles and current control file)
- Tablespace
- Datafile (current or image copy)
- Archived redo log
- Control file (current or image copy)

RMAN backs up the files into one or more backup sets. You can set parameters for the **backup** command to specify the filenames for the backup pieces, the number of files to go in each set, and which channel should operate on each input file.

You can make RMAN backups when the database is open or closed. Closed backups can be *consistent* or *inconsistent*, depending on how the database was shut down; open backups are always inconsistent. Consistent backups can be restored without recovery. An inconsistent backup will require some media recovery when it is restored, but is otherwise just as valid as a consistent backup.

RMAN backup are further divided into *full* and *incremental* backups. Full backups are non-incremental, i.e., every used block is backed up.

This section contains the following procedures:

- [Making Consistent and Inconsistent Backups](#)
- [Making Whole Database Backups](#)
- [Backing Up Tablespaces and Datafiles](#)
- [Backing Up Control Files](#)
- [Backing Up Archived Redo Logs](#)
- [Making Incremental Backups](#)

See Also: For an overview of RMAN backups, see "[Backup Sets](#)" on page 4-28. For a complete description of **backup** command syntax, see "[backup](#)" on page 11-21. For a discussion of the various RMAN backup types, see "[Backup Types](#)" on page 4-36.

Making Consistent and Inconsistent Backups

The procedures in this chapter allow you to make backups when the database is open or closed. Closed backups are either consistent or inconsistent; open backups are always inconsistent. Note that Oracle does not permit inconsistent backups in NOARCHIVELOG mode.

To make a consistent backup, the database:

- Must be mounted, but not open.
- Must *not* have crashed or aborted the last time it was open.

If these conditions are not met, the backup will be inconsistent.

Making Whole Database Backups

If you can afford to close your database, Oracle recommends taking closed, consistent backups of your whole database. If you cannot shut down your database, then your only option is to make an open backup.

To make a whole database backup:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

To write the output to a log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log /oracle/log/mlog.f
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.
3. Allocate one or more channels of type **disk** or type *'sbt_tape'*. This example backs up all the datafiles as well as the control file. It does not specify a **format** parameter, so RMAN gives each backup piece a unique name automatically and stores it in the port-specific default location (\$ORACLE_HOME/dbs on UNIX):

```
run {
  allocate channel chl type disk;
  backup database;
  sql 'ALTER SYSTEM ARCHIVE LOG CURRENT'; # archives current redo log
  sql 'ALTER SYSTEM ARCHIVE LOG ALL'; # archives all un-archived redo logs
}
```

Optionally, use the **format** parameter to specify a filename for the backup piece. For example, enter:

```
run {
  allocate channel chl type 'sbt_tape';
  backup database
  format '/oracle/backup/%U'; # %U generates a unique filename
}
```

Optionally, use the **tag** parameter to specify a tag for the backup. For example, enter:

```
run {
  allocate channel chl type 'sbt_tape';
  backup database
  tag = 'weekly_backup'; # gives the backup a tag identifier
}
```

4. Issue a **list** command to see a listing of your backup sets and pieces.

Backing Up Tablespaces and Datafiles

Back up tablespaces and datafiles when the database is either open or closed. Note that all open database backups are always inconsistent. Do not issue ALTER DATABASE BEGIN BACKUP before making an online tablespace backup.

To back up a tablespace:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.
3. Allocate one or more channels before issuing the **backup** command. This example backs up three tablespaces, using the **filesperset** parameter to specify

that no more than three datafiles should go in each backup set, and also backs up the control file:

```
run {
  allocate channel ch1 type disk;
  allocate channel ch2 type disk;
  allocate channel ch3 type disk;
  backup filesperset = 3
    tablespace inventory, sales
    include current controlfile;
}
```

4. Issue a **list** command to see a listing of your backup sets and pieces.

To back up a datafile:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.
3. Allocate one or more channels before issuing the **backup** command. This example backs up datafiles 1-6 as well as an image copy of a datafile:

```
run {
  allocate channel ch1 type disk;
  backup
    (datafile 1,2,3,4,5,6
     filesperset 3)
    datafilecopy '/oracle/copy/tbs_1_c.f';
}
```

4. Issue a **list** command to see a listing of your backup sets and pieces.

To back up a datafile copy:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a message log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. Mount or open the database.
3. Allocate one or more channels before issuing the **backup** command. This example backs up datafile copy `df1.copy` to tape:

```
run {
    allocate channel ch1 type 'sbt_tape';
    backup datafilecopy '/oracle/copy/df1.copy';
}
```

4. Issue a **list** command to see a listing of your backup sets and pieces.

Backing Up Control Files

You can make backups of the control file when the database is open or closed. RMAN uses a snapshot control file to ensure a read-consistent version.

Whole database backups automatically include the current control file, but the current control file does not contain a record of the whole database backup. To obtain a control file backup with a record of the whole database backup, make a backup of the control file after executing the whole database backup.

Include a backup of the control file within any backup by specifying the **include current controlfile** option.

To back up the current control file:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. Mount or open the database.
3. Allocate a channel before issuing the **backup** command. This example backs up the current control file to tape and uses a tag:

```
run {
    allocate channel ch1 type 'sbt_tape';
    backup current controlfile
    tag = mondayPMbackup;
}
```

4. Optionally, issue a **list** command to see a listing of your backup sets and pieces.

To back up a control file copy:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. Mount or open the database.
3. Allocate a channel before issuing the **backup** command. This example backs up the control file copy `'/oracle/copy/cf.f'`:

```
run {
    allocate channel ch1 type disk;
    backup controlfilecopy '/oracle/copy/cf.f';
}
```

4. Optionally, issue a **list** command to see a listing of your backup sets and pieces.

To include the current control file in another backup:

Specify the **include current controlfile** option after specifying the backup object. For example, this command backs up tablespace FOO and includes the current control file in the backup:

```
run {
    allocate channel c1 type disk;
    backup tablespace foo
    include current controlfile;
}
```

Backing Up Archived Redo Logs

The archived redo logs are the key to successful recovery. Back them up regularly.

You can specify the **delete input** option in the **backup** command, which will delete the archived redo logs after you have backed them up. Thus, you can back up archived logs to tape and clear your disk space of the old logs in one step.

To back up a sequence of archived redo logs:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

If you want to write the output to a log file, specify the file at startup. For example, enter:

```
% rman target / catalog rman/rman@rcat log "/oracle/log/mlog.f"
```

2. Mount or open the database.
3. Allocate a channel before issuing the **backup** command. This example backs up all of the archived redo log files to tape and deletes the input logs from disk:

```
run {
  allocate channel chl type 'sbt_tape';
  backup archivelog all          # Backs up all archived redo logs.
  delete input;                 # Optionally, delete the input logs
}
```

You can also specify a range of archived redo logs by time, SCN, or log sequence number. This example backs up all archived logs created more than 7 and less than 30 days ago:

```
run {
  allocate channel chl type disk;
  backup archivelog
    from time 'SYSDATE-30' until time 'SYSDATE-7';
}
```

4. Issue a **list** command to see a listing of your backup sets and pieces.

Making Incremental Backups

You can make consistent or inconsistent incremental backups of the database or individual tablespaces or datafiles. This procedure makes an incremental backup of a database that was shut down cleanly.

To make a consistent, incremental backup:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.
3. Allocate a channel before issuing the **backup** command. This example makes a level 0 backup of the database:

```
run {
  allocate channel ch1 type disk;
  backup
    incremental level = 0
    database;
}
```

This example makes a differential level 1 backup of the SYSTEM tablespace and datafile `sales.f`. It will only back up those data blocks changed since the most recent level 1 or level 0 backup:

```
run {
  allocate channel ch1 type disk;
  backup
    incremental level = 1
    tablespace system
    datafile '/oracle/dbs/sales.f';
}
```

This example makes a cumulative level 2 backup of the tablespace TBS_2. It will only back up those data blocks changed since the most recent level 1 or level 0 backup:

```
run {
  allocate channel ch1 type disk;
  backup
    incremental level = 2 cumulative # specify cumulative option
    tablespace tbs_1;
}
```

4. Optionally, issue a **list** command to see a listing of your backup sets and pieces.

Making Image Copies

In many cases, making a copy is better than making a backup, because copies are not in an RMAN-specific format and hence are suitable for use without any additional processing. In contrast, you must process a backup set with a **restore** command before it is usable. So, you can perform media recovery on a datafile copy, but not directly on a backup set, even if it contains only one datafile and is composed of a single backup piece.

Note: You cannot make incremental copies, although you can use the **level** parameter to make a copy serve as a basis for subsequent incremental backup sets.

Use the **copy** command to create image copies. RMAN always writes the output file to disk. You can copy the following types of files:

- Datafiles (current or copies)
- Archived redo logs
- Control files (current or copies)

To make consistent copies of all database files:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.

3. Generate a report of the current database schema:

```
RMAN> report schema;
```

```
RMAN-03022: compiling command: report
```

```
Report of database schema
```

File	K-bytes	Tablespace	RB segs	Name
1	35840	SYSTEM	YES	/oracle/dbs/tbs_01.f
2	978	SYSTEM	YES	/oracle/dbs/tbs_02.f
3	978	TBS_1	NO	/oracle/dbs/tbs_11.f
4	978	TBS_1	NO	/oracle/dbs/tbs_12.f

4. Copy all of the datafiles and include the current control file. For example, enter:

```
run {
  allocate channel ch1 type disk;
  copy
    datafile 1 to '/oracle/copy/df_1.f',
    datafile 2 to '/oracle/copy/df_2.f',
    datafile 3 to '/oracle/copy/df_3.f',
    datafile 4 to '/oracle/copy/df_4.f',
    current controlfile to '/oracle/copy/cf.f';
}
```

5. Issue a **list copy** command to see a listing of your copies.

To copy datafiles, archived redo logs, and control files:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. For a consistent backup, mount but do not open the database and ensure that the database was closed cleanly prior to mounting. If the database is open, or is mounted but *not* closed cleanly when last opened, the backup will be inconsistent.
3. Copy the desired datafiles, archived redo logs, and control files. For example, enter:

```
run {
  allocate channel ch1 type disk;
  # allocate multiple channels for parallelization. For parallelization, issue one
  # copy command rather than multiple copy commands.
  allocate channel ch2 type disk;
  allocate channel ch3 type disk;
  copy
    # copy datafiles and datafile copies
    datafile '/oracle/dbs/tbs_8.f' to '/oracle/copy/df_8.f',
    datafilecopy '/oracle/copy/df_2.cp' to '/oracle/dontouch/df_2.f',
    datafilecopy tag = 'weekly_df1_copy' to '/oracle/copy/df_1.f',

    # copy archived redo logs
    archivelog '/oracle/arc_dest/arcr_1_1.arc' to '/oracle/copy/arcr_1_1.arc',
    archivelog '/oracle/arc_dest/arcr_1_2.arc' to '/oracle/copy/arcr_1_2.arc',

    # copy a control file copy
    controlfilecopy '/oracle/copy/cf.f' to '/oracle/dontouch/cf.f';
}
```

4. Issue a **list copy** command to see a listing of your copies.

Monitoring Backup and Copy Operations

You may need to identify what a server session performing a backup or copy operation is doing. RMAN allows you to perform the following checks:

- [Correlating Server Sessions with Channels](#)
- [Monitoring Progress](#)
- [Monitoring Performance](#)

Correlating Server Sessions with Channels

To identify which server sessions correspond to which RMAN channels, use the **set** command with the **command id** parameter. The **command id** parameter enters the specified string into the CLIENT_INFO column of the V\$SESSION dynamic performance view. Join V\$SESSION with V\$PROCESS to correlate the server session with the channel.

The CLIENT_INFO column of V\$SESSION will contain information for each Recovery Manager server session. The data will appear in one of the following formats:

- **id=string**
This form appears for the first connection to the target database established by RMAN.
- **id=string, ch=channel_id**
This form appears for all allocated channels.

The SPID column of V\$PROCESS identifies the O/S process number.

See Also: For more information on V\$SESSION and V\$PROCESS, see the *Oracle8i Reference*.

To correlate a process with a channel during a backup:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```


2. Set the **command id** parameter *after* allocating the channels and then back up the desired object. For example, enter:

```
run {
  allocate channel t1 type disk;
  allocate channel t2 type disk;
  set command id to 'rman';
  backup
    incremental level 0
    filesperset 5
    tablespace 'SYSTEM';
  # optionally, issue a host command to access the O/S prompt
  host;
  sql 'ALTER SYSTEM ARCHIVE LOG ALL';
}
```

3. Start a SQL*Plus session and then query the joined V\$SESSION and V\$PROCESS views *while the RMAN job is executing*. For example, enter:

```
SELECT sid, spid, client_info
FROM v$process p, v$session s
WHERE p.addr = s.paddr
AND client_info LIKE '%id=rman%';
```

SID	SPID	CLIENT_INFO
8	21973	id=rman
16	22057	id=rman
17	22068	id=rman,ch=t1
18	22070	id=rman,ch=t2

See Also: For **set command id** syntax, see "[set_run_option](#)" on page 11-133.

Monitoring Progress

Monitor the progress of backups, copies, and restores by querying the view V\$SESSION_LONGOPS.

Each server session performing a backup, restore, or copy reports its progress compared to the total amount of work required for that particular part of the restore. For example, if you perform a restore using two channels, and each channel has two backup sets to restore (a total of 4 sets), then each server session reports its progress through a single set. When that set is completely restored, RMAN starts reporting progress on the next set to restore.

Query this information using the following SQL statement:

```
SELECT sid, serial#, context, sofar, totalwork
       round(sofar/totalwork*100,2) "% Complete",
FROM v$session_longops
WHERE opname LIKE 'RMAN:%'
AND opname NOT LIKE 'RMAN: aggregate%';
```

See Also: For more information on V\$SESSION_LONGOPS, see the *Oracle8i Reference*. If you want more detailed progress and performance information, see V\$BACKUP_SYNC_IO and V\$BACKUP_ASYNC_IO.

Monitoring Performance

Monitor backup and restore performance by querying V\$BACKUP_SYNC and V\$BACKUP_ASYNC_IO. For a complete description of the contents of these views and how you can use them to tune backup performance, see *Oracle8i Tuning*.

See Also: For more information on these views, see the *Oracle8i Reference*.

Backup and Copy Scenarios

Following are some useful scenarios for performing backups and copies:

- [Reporting Datafiles Needing Backups](#)
- [Skipping Files when Backing Up a Database](#)
- [Spreading a Backup Across Multiple Disk Drives](#)
- [Backing Up a Large Database to Multiple Filesystems](#)
- [Specifying the Size of Backup Sets](#)
- [Multiplexing Datafiles in a Backup](#)
- [Backing Up Archived Redo Logs](#)
- [Backing Up and Deleting Multiple Copies of an Archived Redo Log](#)
- [Performing Differential Incremental Backups](#)
- [Performing Cumulative Incremental Backups](#)
- [Duplexing Backup Sets](#)
- [Parallelizing Backup Sets](#)
- [Backing Up in NOARCHIVELOG Mode](#)

- [Backing Up in a Parallel Server Environment](#)
- [Cataloging O/S Copies](#)
- [Maintaining Backups and Copies](#)
- [Handling Errors During Backups and Copies](#)

Reporting Datafiles Needing Backups

The following command reports all datafiles in the database that would require the application of 6 or more incremental backups to be recovered to their current state:

```
report need backup incremental 6 database;
```

The following command reports all datafiles from tablespace SYSTEM that have not had a backup (full or incremental) in five or more days:

```
report need backup days 5 tablespace system;
```

Skipping Files when Backing Up a Database

The following example, which assumes that the database is running in ARCHIVELOG mode, shows a common way to back up the database (skipping tablespaces that are offline):

```
run {
    allocate channel dev1 type 'sbt_tape';
    backup database
        skip readonly
        skip offline;
}
```

You need to back up a read-only tablespace only once after it has been made read-only. You can use the **skip readonly** option to skip read-only datafiles. If you use the **skip offline** option, then the **backup** will not attempt to access offline datafiles. Use this option if the offline datafiles are not available.

Spreading a Backup Across Multiple Disk Drives

Typically, you do not need to specify a format when backing up to tape because the default %U conversion variable generates a unique filename for all tape backups. When backing up to disk, however, you can specify a format if you want to spread your backup across several disk drives for improved performance. In this case, allocate one **disk** channel per disk drive and specify the format string on the

allocate channel command. Specify the format so that the filenames are on different disks.

For example, issue:

```
run {
  allocate channel disk1 type disk format '/disk1/%d_backups/%U';
  allocate channel disk2 type disk format '/disk2/%d_backups/%U';
  backup database;
}
```

Backing Up a Large Database to Multiple Filesystems

In this scenario, you have a 40 Gb database that you want to back up to disk. Because RMAN does not back up to raw disk, you must spread the backup across filesystems. You decide to back up to four filesystems and make each backup set roughly the same size: 10 Gb. You want to make each backup piece no more than 2 Gb so that each backup set contains five backup pieces.

You decide to use the **format** parameter of the **allocate channel** command so that each channel will write to a different filesystem. You use conversion variables to guarantee unique names for the backup pieces. For example, the following RMAN script spreads the backup across four filesystems (`/fs1`, `/fs2`, `/fs3`, `/fs4`), creating four backup sets in these directories and grouping the datafiles so that each backup set is about the same size.

```
run {
  allocate channel fs1 type disk format='/fs1/%u.%p';
  allocate channel fs2 type disk format='/fs2/%u.%p';
  allocate channel fs3 type disk format='/fs3/%u.%p';
  allocate channel fs4 type disk format='/fs4/%u.%p';

  setlimit channel fs1 kbytes=2000000; #limit file size to 2Gb
  setlimit channel fs2 kbytes=2000000;
  setlimit channel fs3 kbytes=2000000;
  setlimit channel fs4 kbytes=2000000;

  backup database;
}
```

Specifying the Size of Backup Sets

When making backups, RMAN divides the total number of files requiring backups by the number of allocated channels to calculate the number of files to place in each backup set. Use the **filesperset** and **setsize** parameters to override this calculation and specify how many files should go in each backup set.

Using filesperset

When you specify the **filesperset** parameter, RMAN compares the **filesperset** value to the automatically calculated value (number of files / allocated channels) and takes the lowest of the two, thereby ensuring that all channels are used. If the number of files specified or implied by the combined *backupSpec* clauses is greater than **filesperset**, e.g., eight total files need backing up when **filesperset** = 4, then RMAN creates multiple backup sets to maintain the correct ratio of files per backup set.

If you do not specify **filesperset**, RMAN compares the calculated value (number of files / allocated channels) to the default value of 64 and takes the lowest of the two, again ensuring that all channels are used. The default value of 64 is high for most applications: specify a lower value or use the **setsize** parameter to limit the size of a backup set.

RMAN always attempts to create enough backup sets so that all allocated channels have work to do. An exception to the rule occurs when there are more channels than files to back up. For example, if RMAN backs up one datafile when three channels are allocated and **filesperset** = 1, then two channels are necessarily idle.

This example parallelizes a backup, specifying that no more than three datafiles go in any one backup set, and no more than one archived redo log in any one backup set:

```
run {
  allocate channel ch1 type disk;
  allocate channel ch2 type disk;
  allocate channel ch3 type disk;
  allocate channel ch4 type disk;
  backup
    datafile 1,2,3,4,5,9,10,11,12,15
      filesperset = 3
    archivelog all
      filesperset = 1;
}
```

Using setsize

The **setsize** parameter specifies a maximum size for a backup set in units of 1K (1024 bytes). Thus, to limit a backup set to 2Mb, specify **setsize** = 2000. RMAN attempts to limit all backup sets to this size, which is useful in media manager configurations when you want each backup set to be no larger than one tape.

Configure your backup sets so that they fit on one tape volume rather than span multiple tape volumes. Otherwise, if one tape of a multi-volume backup set fails, then you lose the whole backup set.

The **setsize** parameter is easier to use than **filesperset** when you make backups of archived redo logs. This scenario backs up archived redo logs to tape, setting the size at 2Mb to ensure that each backup set fits on one tape volume:

```
run {
  allocate channel ch1 type 'sbt_tape';
  allocate channel ch2 type 'sbt_tape';
  backup setsize = 2000
    archivelog all;
}
```

The **setsize** parameter is also easier for datafile backups when your datafiles are striped or reside on separate disk spindles and you either:

- Use a high-bandwidth tape drive that requires several datafiles be multiplexed in order to keep the tape drive streaming.
- Make backups while the database is open and you want to spread the I/O load across several disk spindles in order to leave bandwidth for online operations.

For example, if your datafiles are striped four ways, then setting **filesperset** = 4 allows each backup set to read data from four disk spindles, thereby distributing the I/O load.

Multiplexing Datafiles in a Backup

Assume you want to back up a database called FOO. The following conditions obtain:

- You have three tape drives available for the backup.
- The database contains 26 datafiles.
- You want to multiplex datafiles into the backup, placing four datafiles into each backup set.

You set **filesperset** equal to 4 because it is sufficient to keep the tape drive streaming. In this example, you are not concerned about how datafiles are grouped into backup sets.

Issue the following commands:

```
create script foo_full {
  allocate channel t1 type 'SBT_TAPE';
  allocate channel t2 type 'SBT_TAPE';
  allocate channel t3 type 'SBT_TAPE';
  backup filesperset 4
    database format 'FOO.FULL.%n.%s.%p';
}
```

This script will back up the whole database, including all datafiles and the control file. Since there are 27 files to be backed up (26 datafiles and a control file) and a maximum of four files per backup set, Oracle creates seven backup sets. The backup piece filenames will have the following format, where *db_name* is the database name, *set_num* is the backup set number, and *piece_num* is the backup piece number:

```
FOO.FULL.db_name.set_num.piece_num
# for example, a file may have the following name:
FOO.FULL.prod1.3.1
```

Assuming no backup sets have been recorded in the recovery catalog prior to this job, then *set_num* will range from 1 through 7 and *piece_num* will be 1 or more.

Note that in the SBT API version 2.0, media vendors can specify the maximum size of a backup piece, causing RMAN to comply with this restriction automatically.

Backing Up Archived Redo Logs

You can also back up archived redo logs to tape. You can specify the range of archived redo logs by time, SCN, or log sequence number.

Note: If specifying by time range, set the NLS_LANG and NLS_DATE_FORMAT environment variables before invoking Recovery Manager. See "[Setting NLS Environment Variables](#)" on page 5-2.

Specifying an archived log range does not guarantee that RMAN backs up all redo in the range. For example, the last archived log may end before the end of the range, or a log in the range may be missing. RMAN backs up the logs it finds and does not issue a warning. Online logs cannot be backed up; you must first archive them.

This example backs up the logs archived between 8:57 p.m. and 9:06 p.m. on November 18, 1998.

```
NLS_LANG=american
NLS_DATE_FORMAT='Mon DD YYYY HH24:MI:SS'
run {
  allocate channel dev1 type 'sbt_tape';
  backup
  archivelog all
    from time 'Nov 13 1998 20:57:13'
    until time 'Nov 13 1998 21:06:05';
}
```

The following example backs up all archived logs from sequence# 288 to sequence# 301 and deletes the archived redo logs after the backup is complete. If the backup fails the logs are not deleted.

```
run {
  allocate channel dev1 type 'sbt_tape';
  backup
    archivelog from logseq 288 until logseq 301 thread 1
    delete input;
}
```

The following command backs up all archived logs generated during the last 24 hours. The example archives the current redo log first to ensure that all redo generated up to the present gets backed up.

```
run {
  allocate channel dev1 type 'sbt_tape';
  sql "ALTER SYSTEM ARCHIVE LOG CURRENT";
  backup archivelog from time 'SYSDATE-1';
}
```

See Also: For more information about your environment variables, see your operating system-specific documentation.

Backing Up and Deleting Multiple Copies of an Archived Redo Log

In this scenario, you set your initialization parameters so that you automatically archive your redo logs to two locations: `/oracle/arch/dest_1/*` and `/oracle/arch/dest_2/*`. Therefore, you have two identical copies of the archived redo log for each log sequence number. You decide to back up each copy of your archived redo logs and then delete the originals.

Because the **backup archivelog all** command has the meaning of "Back up exactly one copy of each distinct log sequence number," RMAN will not put two copies of the same log sequence number into the same backup set. Furthermore, if you specify the **delete input** option, RMAN only deletes the specific copy of the archived redo log that it backs up.

The easiest solution in this case is to back up both copies of each archived redo log and then delete both copies. Use the **like** parameter of the *archivelogRecordSpecifier* to indicate which destination to use. The **like** parameter allows you to match file filenames in both of your archive destinations. For example, execute the following:

```
run {
  allocate channel t1 type 'sbt_tape';
  allocate channel t2 type 'sbt_tape';
  backup
```



```

filesperset=20
format='al_%d/%t/%s/%p'
(archiveolog all like '/oracle/arch/dest1/%' channel t1 delete input)
(archiveolog all like '/oracle/arch/dest2/%' channel t2 delete input);
}

```

This example backs up the archived redo logs in the primary destination on one set of tapes and the logs from the second destination on another set of tapes. This scenario assumes that you have two tape drives available. Note that some tape subsystems will combine the two RMAN channels into a single data stream and write them to a single tape drive. You may need to configure your media management vendor to prevent this scenario from occurring.

Performing Differential Incremental Backups

A differential incremental backup contains only blocks that have been changed since the most recent backup at the same level or lower. The first incremental backup must be a level 0 backup that contains all used blocks.

```

run {
  allocate channel dev1 type 'sbt_tape';
  backup incremental level 0
    database;
}

```

An incremental backup at level 1 or higher will contain all blocks changed since the most recent level 1 backup. If no previous level 1 backup is available, RMAN copies all blocks changed since the base level 0 backup.

```

run {
  allocate channel dev1 type 'sbt_tape';
  backup incremental level 1
    database;
}

```

If you add a new datafile or tablespace to the database, then make a level 0 backup before making another incremental backup. Otherwise, the incremental backup of the tablespace or the database will fail because Recovery Manager does not find a parent backup for the new datafiles.

```

run {
  allocate channel dev1 type 'sbt_tape';
  backup incremental level 0
    tablespace new_tbs;
}

```

Note that you can perform incremental backups in NOARCHIVELOG mode.

Performing Cumulative Incremental Backups

A cumulative incremental backup at level n contains only blocks that have been changed since the most recent backup at level $n - 1$ or lower. Cumulative backups require more storage space than differential backups, but they are preferable during a restore operation because only one backup for a given level is needed. Note that the first incremental backup must be a level 0 backup that contains all used blocks.

A cumulative backup at level 2 will contain all blocks changed since the most recent level 1 backup, copying all blocks changed since the base level 0 backup only if a previous level 1 is unavailable. In contrast to a cumulative backup, a differential backup at level 2 will determine which level 1 or level 2 backup occurred most recently and copy all blocks changed since that backup.

```
run {
    allocate channel dev1 type 'sbt_tape';
    backup incremental level 2 cumulative # blocks changed since level 0 or level 1
    database;
}
```

Duplexing Backup Sets

Prudence suggests making multiple copies of your backups to protect against disaster, media damage, or human error. Oracle allows you to make up to four backup sets simultaneously, each an exact duplicate of the others.

The **set duplex** command, which affects only the **backup** command, specifies the number of copies of each backup piece that RMAN should create. The **set duplex** command affects all channels allocated after issuing the command and is in effect until explicitly disabled (**OFF**) or changed during the session.

Note: The **set duplex** command will generate an error if there are previously allocated channels.

For example, you can enter:

```
run {
    set duplex=3;
    allocate channel ch1 type 'sbt_tape';
    backup datafile 1;
}
```

This command will create three identical backups of `datafile 1`. Each backup piece will have a unique name since the `%U` default format guarantees it.

Note that you must set the `BACKUP_TAPE_IO_SLAVES` initialization parameter to `TRUE` in order to perform duplexed backups; otherwise, an error will be signaled. RMAN will configure as many slaves as needed for the number of backup copies you request.

See Also: For more information on the `BACKUP_TAPE_IO_SLAVES` initialization parameter, see the *Oracle8i Reference*.

Parallelizing Backup Sets

If you want to create multiple backup sets and allocate multiple channels, then RMAN automatically parallelizes its operation and writes multiple backup sets in parallel. The allocated server processes divide up the work of backing up the specified datafiles, control files, and archived redo logs. Note that you cannot stripe a single backup set across multiple channels.

RMAN automatically assigns a backup set to a device. You can specify that Oracle should write all backup sets for a *backupSpec* to a specific channel using the **channel** parameter.

For example, this RMAN parallelizes the backup operation by specifying which channels RMAN should allocate for which operations:

```
run {
  allocate channel ch1 type 'SBT_TAPE';
  allocate channel ch2 type disk;
  allocate channel ch3 type disk;
  backup
    # channel ch1 backs up datafiles to tape
    (datafile 1,2,3,4
     channel ch1)
    # channel ch2 backs up control file copy to disk
    (controlfilecopy '/oracle/copy/cf.f'
     channel ch2)
    # channel ch3 backs up archived redo logs to disk
    (archiveolog from time 'SYSDATE-14'
     channel ch3);
}
```

Backing Up in NOARCHIVELOG Mode

This script puts the database into the correct mode for a consistent, whole database backup and then backs up the database. Note that the script performs a shutdown, startup, shutdown, and then startup again before performing a duplexed backup:

```
# Shut down the database cleanly using immediate priority. This type of shutdown lets
# current calls to the database complete, but prevents further logons or calls.
# If the database is not up now, you will get a message saying so but RMAN will not
# treat this situation as an error.

shutdown immediate;

# Start up the database in case it crashed or was not shutdown cleanly prior to
# starting this script. This will perform a crash recovery if it is needed. Oracle
# uses the default INIT.ORA file. Alternatively, use this form: startup force dba
# pfile=<filename>. Use the DBA option because you are going to shut down again right
# away and do not want to let users in during the short interval. Use the FORCE
# option because it cannot hurt and might help in certain situations.

startup force dba;
shutdown immediate;

# Here, we know that the database is cleanly closed and is now ready for a cold
# backup. RMAN requires that the database be started and mounted to perform a backup,
# so do that now.

startup mount;
run {
    # duplex the backup
    set duplex = 2;

    # allocate channel t1 type 'SBT_TAPE';
    # allocate channel t2 type 'SBT_TAPE';
    allocate channel t1 type disk;
    allocate channel t2 type disk;

    set limit channel t1 kbytes 2097150;
    set limit channel t2 kbytes 2097150;

    backup
        incremental level 0
        filesperset 5
        database;
}

# now that the backup is complete, open the database.
alter database open;
```

Note that you can skip tablespaces, but any skipped tablespace that has not been offline or read-only since its last backup will be lost if the database has to be restored from a backup. When backing up to disk, make sure that the destination (file system or raw device) has enough free space.

Backing Up in a Parallel Server Environment

The following script distributes datafile and archived redo log backups across two nodes in a parallel server environment:

```
run {
  allocate channel node_1 type disk connect 'sys/sys_pwd@node_1';
  allocate channel node_2 type disk connect 'sys/sys_pwd@node_2';
  backup filesperset 1
    (tablespace system, rbs, data1, data2
     channel node_1)
    (tablespace temp, reccat, data3, data4
     channel node_2);
  backup filesperset 20
    (archivelog
     until time 'SYSDATE'
     thread 1
     delete input
     channel node_1);
    (archivelog
     until time 'SYSDATE'
     thread 2
     delete input
     channel node_2);
}
```

See Also: For more information about OPS backups, see *Oracle8i Parallel Server Concepts and Administration*.

Cataloging O/S Copies

You can use O/S utilities to make datafile copies and then catalog them in the recovery catalog. Note that you can only catalog copies made to disk. Because the format of backup pieces is proprietary, O/S utilities cannot write backups that Recovery Manager can read.

You must make the datafile copies using O/S techniques. If the database is open and the datafile is online, first issue ALTER TABLESPACE BEGIN BACKUP. The resulting image copy can be cataloged:

```
catalog datafilecopy '?/dbs/tbs_33.f';
```

Maintaining Backups and Copies

How long you must keep backups and copies depends on factors such as:

- How frequently you take backups.
- How far in the past point-in-time recovery is needed.

For example, if you back up all datafiles daily, do not require point-in-time recovery, and need only one backup per datafile, then you can delete previous backups as soon as the new one completes.

```
# delete a specific datafile copy
change datafilecopy '?/dbs/tbs_35.f' delete;

# delete archived redo logs older than 31 days
change archivelog until time 'SYSDATE-31' delete;
```

You must allocate a channel before deleting a backup piece. The specified backup piece must have been created on the same type of device. Note that the **allocate channel for maintenance** command is *not* issued inside of a **run** command.

```
# delete a backup piece
allocate channel for maintenance type 'sbt_tape';
change backuppiece 'testdb_87fa39e0' delete;
release channel;
```

Handling Errors During Backups and Copies

By default a checksum is calculated for every block read from a datafile and stored in the backup or image copy. If you use the **nochecksum** option, then checksums are not calculated. If the block already contains a checksum, however, then the checksum is validated and stored in the backup. If the validation fails, the block is marked corrupt in the backup.

The **set maxcorrupt for datafile** command determines how many corrupt blocks in a datafile that **backup** or **copy** will tolerate. If any datafile has more corrupt blocks than specified by the **maxcorrupt** parameter, the command aborts. Note that if you specify the **check logical** option, RMAN checks for logical and physical corruption.

By default, the **backup** command terminates when it cannot access a datafile. You can specify various parameters to prevent termination:

If you specify	then RMAN skips
the skip inaccessible option	inaccessible datafiles. Note that a datafile is only considered inaccessible if it cannot be read. Some offline datafiles can still be read because they still exist on disk. Others have been deleted or moved and so cannot be read, making them inaccessible.
the skip offline option	offline datafiles.
the skip readonly option	datafiles with read-only status.

```
run {  
  allocate channel dev1 type 'sbt_tape';  
  set maxcorrupt for datafile 1,2,3 to 5;  
  backup database  
    skip inaccessible  
    skip readonly  
    skip offline;  
}
```

Restoring and Recovering with Recovery Manager

This chapter describes how to use Recovery Manager to perform restore and recovery operations, and includes the following topics:

- [Restoring Datafiles, Control Files, and Archived Redo Logs](#)
- [Recovering Datafiles](#)
- [Restore and Recovery Scenarios](#)

Restoring Datafiles, Control Files, and Archived Redo Logs

Use the RMAN **restore** command to restore datafiles, control files, or archived redo logs from backup sets or image copies. RMAN restores backups from disk or tape, but image copies only from disk.

When restoring files, you should:

- Allocate at least one channel before restoring backups or copies. Specify multiple channels to parallelize the restore operation.
- Allocate a channel of type **disk** if you use the **from copy** option.
- Allocate the appropriate **disk** or *'sbt_tape'* channel when restoring files. If the appropriate type of device is not allocated, then you may not be able to find a candidate backup set or copy to restore, and the **restore** command fails.

Restore files to either:

- The default location, which overwrites the files with the same name.
- A new location specified by the **set newname** command. If you restore datafiles to a new location, then Oracle considers them datafile copies and records them as such in the control file and recovery catalog.

This section contains the following topics:

- [Restoring a Database](#)
- [Restoring Tablespace and Datafiles](#)
- [Restoring Control Files](#)
- [Restoring Archived Redo Logs](#)
- [Restoring in Preparation for Incomplete Recovery](#)

See Also: For **restore** syntax, see "[restore](#)" on page 11-112. For **set newname** syntax, see "[set_run_option](#)" on page 11-133.

Restoring a Database

When restoring a target database, you can:

- Restore the database to its default location because of a media failure.
- Move the database to a new host because of a media failure.
- Create a test database based on backups of your target database.

To restore the database to its default location, issue the **restore database** command. To move your target database to a new host, rename the datafiles as needed using **set newname**. To create a test database using backups of your target database, use the **duplicate** command (see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#) for complete instructions).

This chapter contains the following topics:

- [Restoring the Database to its Default Location](#)
- [Moving the Target Database to a New Host with the Same Filesystem](#)
- [Moving the Target Database to a New Host with a Different Filesystem](#)

Restoring the Database to its Default Location

If you do not specify **set newname** commands for the datafiles during a restore job, the database must be closed or the datafiles must be offline. Otherwise, you will see output similar to the following, which results from an attempt to restore datafile 3 while the file is online:

```

RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03002: failure during compilation of command
RMAN-03013: command type: restore
RMAN-03006: non-retryable error occurred during execution of command: IRESTORE
RMAN-07004: unhandled exception during command execution on channel chl
RMAN-10035: exception raised in RPC: ORA-19573: cannot obtain exclusive enqueue
for datafile 3
ORA-19600: input file is datafile-copy 102 (/vobs/oracle/dbs/df.3)
ORA-19601: output file is datafile 3 (/vobs/oracle/dbs/tbs_11.f)
RMAN-10031: ORA-19573 occurred during call to DBMS_BACKUP_RESTORE.COPYDATAFILECOPY

```

The database must be closed when you restore the whole database. If the target database is mounted, then its control file will be updated with any applicable datafile copy and archived log records to describe the restored files.

To restore the database to its default location:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. If the database is open, shut it down and then mount it:

```
shutdown immediate;
```

```
startup mount;
```

3. After allocating channels, restore the database:

```
run {  
    allocate channel ch1 type disk;  
    allocate channel ch2 type disk;  
    allocate channel ch3 type disk;  
    restore database;  
}
```

Moving the Target Database to a New Host with the Same Filesystem

A media failure may force you to move a database by restoring a backup from one host to another. Note that if you want to create a duplicate database for testing and still maintain your original database, use the **duplicate** command instead of following this procedure (see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#)).

Because your restored database will not have the online redo logs of your production database, you will need to perform incomplete recovery up to the lowest SCN of the most recently archived redo log in each thread and then open with the **RESETLOGS** option.

This scenario assumes that:

- You are restoring backups from **HOST_A** onto **HOST_B**.
- You are restoring from tape backups.
- The **HOST_A** database does not have the same **DB_NAME** as the **HOST_B** database.
- The **HOST_B** filenames and directory paths are the same as in **HOST_A**.

The restore procedure differs depending on whether the target database uses a recovery catalog.

Note: You cannot use **RMAN** to restore image copies created on one host to a different host. Transfer the files using an O/S utility and use the **catalog** command to update the **RMAN** metadata.

To restore the database from HOST_A to HOST_B with a recovery catalog:

1. Copy the `init.ora` file for HOST_A to HOST_B using an O/S utility.
2. Connect to the HOST_B target instance and HOST_A recovery catalog. For example, enter:

```
% rman target sys/change_on_install@host_b rman/rman@rcat
```

3. Start the instance without mounting it:

```
startup nomount;
```

4. Restore and mount the control file. Execute a **run** command with the following sub-commands:

- a. Allocate at least one channel.
- b. Restore the control file.
- c. Mount the control file.

```
run {
  allocate channel chl type disk;
  restore controlfile;
  alter database mount;
}
```

5. Because there may be multiple threads of redo, use change-based recovery. Obtain the SCN for recovery termination by finding the lowest SCN among the most recent archived redo logs for each thread.

Start SQL*Plus and use the following query to determine the necessary SCN:

```
SELECT min(scn)
FROM (SELECT max(next_change#) scn
      FROM v$sarchived_log
      GROUP BY thread#);
```

6. Execute a **run** command with the following sub-commands:
 - a. Set the SCN for recovery termination using the value obtained from the previous step.
 - b. Allocate at least one channel.
 - c. Restore the database.
 - d. Recover the database.
 - e. Open the database with the RESETLOGS option.

```
run {
  set until scn = 500; # use appropriate SCN for incomplete recovery
  allocate channel chl type 'sbt_tape';
  restore database;
  recover database;
  sql "ALTER DATABASE OPEN RESETLOGS";
}
```

To restore from HOST_A to HOST_B without a recovery catalog:

1. Copy the `init.ora` file for HOST_A to HOST_B using an O/S utility.
2. Use an O/S utility to make an image copy of the HOST_A control file and transfer it to HOST_B using an O/S utility.
3. Connect to the HOST_B target instance with the **nocatalog** option. For example, enter:

```
% rman target sys/change_on_install@host_b nocatalog
```

4. Mount the database:

```
startup mount;
```
5. Because there may be multiple threads of redo, use change-based recovery. Obtain the SCN for recovery termination by finding the lowest SCN among the most recent archived redo logs for each thread.

Start SQL*Plus and use the following query to determine the necessary SCN:

```
SELECT min(scn)
FROM (SELECT max(next_change#) scn
      FROM v$archived_log
      GROUP BY thread#);
```

6. Execute a **run** command with the following sub-commands:
 - a. Set the SCN for recovery termination using the value obtained from the previous step.
 - b. Allocate at least one channel.
 - c. Restore the database.
 - d. Recover the database.
 - e. Open the database with the RESETLOGS option.

```
run {
  set until scn 500; # use appropriate SCN for incomplete recovery
  allocate channel chl type 'sbt_tape';
  alter database mount;
  restore database;
  recover database;
  sql "ALTER DATABASE OPEN RESETLOGS";
}
```

Moving the Target Database to a New Host with a Different Filesystem

The procedure for moving the database to a machine with a different filesystem is basically the same as described in ["Recovering an Inaccessible Datafile in an Open Database"](#) on page 9-34; the difference is that you need to rename each datafile using **set newname**.

For example, assume that:

- The database on HOST_A has ten datafiles.
- You are restoring from tape backups.
- You restore some datafiles to /disk1 and others to /disk2 on HOST_B.

To restore to HOST_B with a recovery catalog:

1. Follow the procedure in ["Moving the Target Database to a New Host with the Same Filesystem"](#) on page 9-4 (with a recovery catalog), stopping before you execute the **run** command. Make sure to reset all *_DEST and *_PATH parameters in the `init.ora` file that specify a pathname.
2. Execute this **run** command instead:
 - a. Set the end SCN obtained from the SQL*Plus query.
 - b. Allocate at least one channel.
 - c. Specify a new filename for each datafile.
 - d. Mount the database.
 - e. Restore the database.
 - f. Switch the datafiles.
 - g. Recover the database.
 - h. Open the database with the RESETLOGS option.

```
run {
  set until scn 500; # use appropriate SCN for incomplete recovery
  allocate channel chl type disk;
  set newname for datafile 1 to '/disk1/%U'; # rename each datafile manually
  set newname for datafile 2 to '/disk1/%U';
  set newname for datafile 3 to '/disk1/%U';
  set newname for datafile 4 to '/disk1/%U';
  set newname for datafile 5 to '/disk1/%U';
  set newname for datafile 6 to '/disk2/%U';
  set newname for datafile 7 to '/disk2/%U';
  set newname for datafile 8 to '/disk2/%U';
  set newname for datafile 9 to '/disk2/%U';
  set newname for datafile 10 to '/disk2/%U';
  alter database mount;
  restore database;
  switch datafile all; # points the control file to the renamed datafiles
  recover database;
  sql "ALTER DATABASE OPEN RESETLOGS";
}
```

To restore to HOST_B without a recovery catalog:

1. Follow the procedure in ["Moving the Target Database to a New Host with the Same Filesystem"](#) on page 9-4 (without a recovery catalog), stopping before you execute the **run** command in step 6. Make sure to reset all *_DEST and *_PATH parameters in the `init.ora` file that specify a pathname.
2. Execute a **run** command with the following sub-commands:
 - a. Set the end SCN obtained from the SQL*Plus query.
 - b. Allocate at least one channel.
 - c. Specify a new filename for each datafile.
 - d. Restore the database.
 - e. Switch the datafiles.
 - f. Recover the database.
 - g. Open the database with the RESETLOGS option.

```
run {
  set until scn 500; # use appropriate SCN for incomplete recovery
  allocate channel chl type disk;
  set newname for datafile 1 to '/disk1/%U'; # rename each datafile manually
  set newname for datafile 2 to '/disk1/%U';
  set newname for datafile 3 to '/disk1/%U';
  set newname for datafile 4 to '/disk1/%U';
  set newname for datafile 5 to '/disk1/%U';
```



```

set newname for datafile 6 to '/disk2/%U';
set newname for datafile 7 to '/disk2/%U';
set newname for datafile 8 to '/disk2/%U';
set newname for datafile 9 to '/disk2/%U';
set newname for datafile 10 to '/disk2/%U';
restore database;
switch datafile all; # point control file to renamed datafiles
recover database;
sql "ALTER DATABASE OPEN RESETLOGS";
}

```

Restoring Tablespaces and Datafiles

If a datafile is lost or corrupted but the disk is accessible, then you can restore the datafile to its previous location. Simply take the tablespace offline and issue a restore tablespace command. If the old location is inaccessible, take the tablespace offline and restore the associated datafiles to a new location.

If you cannot restore datafiles to the default location, use the **set newname** command before restoring. In this case, Oracle considers the restored datafiles as datafile copies; perform a **switch** to make them the current datafiles. Oracle creates the filename or overwrites it if it already exists.

The RMAN **switch** command is equivalent to the ALTER DATABASE RENAME DATAFILE statement. Note that a switch effectively causes the location of the current datafile to change. Also note that switching "consumes" the copy, i.e., deletes the corresponding records in the recovery catalog and the control file.

If you do not specify the target of the switch, then the filename specified in a prior **set newname** for this file number is used as the switch target. If you specify **switch datafile all**, then all datafiles for which a **set newname** has been issued in this job are switched to their new name.

If you issue **set newname** commands to restore datafiles to a new location with the intention of performing a recovery afterwards, perform a switch after restoring but before recovering to make the restored datafiles the current datafiles.

See Also: For **switch** command syntax, see "[switch](#)" on page 11-144.

To restore a tablespace to its default location:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. After allocating channels, do the following:
 - Take the tablespace that you want to recover offline.
 - Restore the tablespace.

For example, to restore tablespace USER_DATA to disk you might issue:

```
run {
  sql 'ALTER TABLESPACE user_data OFFLINE TEMPORARY';
  allocate channel ch1 type disk;
  restore tablespace user_data;
}
```

3. You will need to perform media recovery on the restored tablespace. See ["Recovering an Inaccessible Datafile in an Open Database"](#) on page 9-34 for the required procedure.

To restore a tablespace to a new location:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. After allocating channels, do the following:
 - Take the tablespace offline.
 - Specify an accessible location to which you can restore the damaged datafile for the offline tablespace.
 - Restore the datafile to the new location.
 - Switch the restored datafile so that the control file considers it the current datafile.

To restore the datafiles for tablespace TBS_1 to a new location on disk, you might enter:

```
run {
  allocate channel ch1 type disk;
  sql 'ALTER TABLESPACE user_data OFFLINE TEMPORARY';
  # restore the datafile to a new location
  set newname for datafile '/disk1/oracle/tbs_1.f' to '/disk2/oracle/tbs_1.f';
  restore tablespace tbs_1;
  # make the control file recognize the restored file as current
  switch datafile all;
}
```

3. You will need to perform media recovery on the restored tablespace. See "[Recovering an Inaccessible Datafile in an Open Database](#)" on page 9-34 for the required procedure.

Restoring Control Files

If a media failure damages your control file and you do not have multiplexed copies, you must restore a backup. Issue **restore controlfile** to restore the control file to the first CONTROL_FILES location specified in the parameter file. RMAN automatically replicates the control file to all CONTROL_FILES locations specified in the parameter file.

Specify a destination name with **restore controlfile to 'filename'** when restoring a control file to a non-default location. If the filename already exists, then Oracle overwrites the file. When you restore the control file to a new location, use the **replicate controlfile from 'filename'** command to copy it to the CONTROL_FILES destinations: RMAN will not replicate the control file automatically.

Using the **replicate controlfile** command is equivalent to using multiple **copy controlfile** commands. After you specify the input control file by name, RMAN replicates the file to the locations specified in the CONTROL_FILES initialization parameter of the target database.

To restore the control file to its default location using a recovery catalog:

1. Start RMAN and connect to the target and recovery catalog databases. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. Start the instance without mounting the database:

```
startup nomount;
```

3. Do the following:
 - a. If for some reason you need to restore a control file created before a certain date, issue a **set until** command for that date. Otherwise, go to the next step.
 - b. Allocate one or more channels.
 - c. Restore the control file.
 - d. Mount the database.

```
run {
  # To restore a control file created before a certain date, issue the following
  # set command using a valid date for 'date_string'. You can also specify an SCN
  # or log sequence number.
  # set until time = 'date_string';
  allocate channel chl type 'sbt_tape';
  restore controlfile;
  alter database mount;
}
```

RMAN automatically replicates to the control file to the locations specified by the `CONTROL_FILES` initialization parameter.

4. If you need to perform media recovery on the datafiles after restoring the control file, see ["Performing Complete Recovery"](#) on page 9-18 or ["Performing Incomplete Recovery"](#) on page 9-23.

To restore the control file to a new location without a recovery catalog:

Note that the control file that contains information about a given backup is not the control file that is backed up along with the backup. For example, if you issue **backup database**, the backup control file in this whole database backup does not contain the record of the whole database backup. The next control file backup will contain information about the whole database backup.

1. Start RMAN and connect to the target database. For example, enter:

```
% rman target / nocatalog
```

2. Mount the database:

```
startup mount;
```

3. Do the following:

- a. If you need to restore a control file created before a certain date, issue a **set until** command for that date. Otherwise, go to the next step.
- b. Allocate one or more channels.
- c. Restore the backup control file to a temporary location to prevent accidental overwriting of the current control file.
- d. Shut down the database.
- e. Replicate the control file from the restored location to all locations specified in the `CONTROL_FILES` parameter of the parameter file.
- f. Mount the database.

```

run {
  # To restore a control file created before a certain date, issue the following
  # set command using a valid date for 'date_string'. You can also specify an SCN
  # or log sequence number.
  # set until time = 'date_string';
  allocate channel chl type 'sbt_tape';
  # restore control file to new location
  restore controlfile to '/oracle/dbs/cf1.ctl';
  shutdown immediate;
  # replicate the control file manually to locations in parameter file
  replicate controlfile from '/oracle/dbs/cf1.ctl';
  startup mount;
}

```

4. If you need to perform media recovery on the datafiles after restoring the control file, see ["Performing Complete Recovery"](#) on page 9-18 or ["Performing Incomplete Recovery"](#) on page 9-23.

See Also: For **replicate controlfile** command syntax, see ["replicate"](#) on page 11-100.

Restoring Archived Redo Logs

RMAN restores archived redo logs with names constructed using the LOG_ARCHIVE_FORMAT parameter and either the LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_1 parameters of the target database. These parameters combine in a port-specific fashion to derive the name of the restored archived log.

Override the destination parameter with the **set archivelog destination** command. By issuing this command, you can manually stage many archived logs to different locations while a database restore is occurring. During recovery, RMAN knows where to find the newly restored archived logs; it does not require them to be in the location specified in the parameter file.

For example, if you specify a different destination from the one in the `init.ora` file and restore backups, subsequent restore and recovery operations will detect this new location and will not look for the files in the `init.ora` parameter destination.

If desired, you can also specify multiple restore destinations for archived redo logs, although you cannot specify these destinations simultaneously. For example, you can issue:

```

run {
  allocate channel chl type disk;
  # Set a new location for logs 1 through 10.
  set archivelog destination to '/disk1/oracle/temp_restore';
  restore archivelog from logseq 1 until logseq 10;
  # Set a new location for logs 11 through 20.
}

```

```
set archivelog destination to '/disk1/oracle/arch';
restore archivelog from logseq 11 until logseq 20;
# Set a new location for logs 21 through 30.
set archivelog destination to '/disk2/oracle/temp_restore';
restore archivelog from logseq 21 until logseq 30;
. . .
recover database;
}
```

Note that if you restore archived redo logs to multiple locations, you only need to issue a single **recover** command. RMAN finds the restored archived logs automatically and applies them to the datafiles.

To restore archived redo logs:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

Optionally, specify a message log file at connect time:

```
% rman target / catalog rman/rman@rcat log = rman_log
```

2. If the database is open, shut it down and then mount it:

```
shutdown immediate; startup mount
```

3. Perform the following operations within your **run** command:

- a. If desired, specify the new location for the restored archived redo logs using **set archivelog destination**. Otherwise, go to next step.
- b. Allocate channels.
- c. Restore the archived redo logs.

For example, this job restores all backup archived redo logs:

```
run {
  # Optionally, set a new location for the restored logs.
  set archivelog destination to '/oracle/temp_restore';
  allocate channel ch1 type disk;
  restore archivelog all;
}
```

See Also: For **set archivelog destination** command syntax, see "[set_run_option](#)" on page 11-133.

Restoring in Preparation for Incomplete Recovery

Use the **set until** command to specify the termination point for recovery. This command affects any subsequent **restore**, **switch**, and **recover** commands that are in the same **run** command.

To restore the database in preparation for incomplete recovery:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

Optionally, specify a message log file at connect time:

```
% rman target / catalog rman/rman@rcat log = rman_log
```

2. If the database is open, shut it down and then mount it:

```
shutdown immediate;
startup mount;
```

3. Perform the following operations within your **run** command:
 - a. Determine whether you want to recover to a specified time, SCN, or log sequence number and issue the appropriate **set until** command.
 - b. Allocate channels.
 - c. Restore the database.

For example, this job restores the database in anticipation of an incomplete recovery until December 15, 1998 at 9 a.m.

```
run {
  set until time 'Dec 15 1998 09:00:00';
  allocate channel chl type 'sbt_tape';
  restore database;
}
```

Recovering Datafiles

Media recovery is the application of redo logs or incremental backups to a restored file in order to update it to the current time or some other specified time. You can only recover or apply incremental backups to current datafiles, not datafile copies.

Perform media recovery when:

- Media failure has damaged datafiles and you need to recover them to the current time.
- You need to recover the entire database to a previous time.
- Media failure has damaged the control file.
- You have executed the `CREATE CONTROLFILE` command.

RMAN restores backup sets of archived redo logs as needed to perform the media recovery. By default, RMAN restores the archived redo logs to the current log archive destination specified in the `init.ora` file. Use the **set archivelog destination** command to specify a different location.

If RMAN has a choice between applying an incremental backup or applying redo, then it always chooses the incremental backup. If overlapping levels of incremental backup are available, then RMAN automatically chooses the one covering the longest period of time.

If possible, make the recovery catalog available to perform the media recovery. If it is not available, RMAN uses information from the target database control file.

Note: If control file recovery is required, then you must make the recovery catalog available. RMAN cannot operate when neither the recovery catalog nor the target database control file are available.

This section contains the following topics:

- [Preparing for Media Recovery](#)
- [Performing Complete Recovery](#)
- [Performing Incomplete Recovery](#)

See Also: For an overview of incremental backups, see "[Incremental Backups](#)" on page 4-38.

Preparing for Media Recovery

When and how to recover depends on the state of the database and the location of its datafiles.

To determine if media recovery is necessary:

1. Start SQL*Plus and connect to your target database. For example, issue the following to connect to PROD1:

```
% sqlplus sys/change_on_install@prod1;
```

2. Determine the status of the database by executing the following SQL query at the command line:

```
SELECT parallel, status FROM v$instance;
```

```
PAR STATUS
---
NO OPEN
```

If the STATUS column reads OPEN, then the database is open, but it is still possible that you need to restore or recover some tablespaces and their datafiles.

3. Execute the following query to check the datafile headers and respond according to the table below:

```
SELECT file#, status, error, recover, tablespace_name, name
FROM v$datafile_header
WHERE error IS NOT NULL
OR recover = 'YES';
```

ERROR column	RECOVER column	Solution
NULL	NO	Unless the error is caused by a temporary hardware or O/S problem, restore the datafile or switch to a copy of that datafile.
NULL	YES	Recover the datafile. The recover command first applies any suitable incremental backups and then applies redo logs. RMAN restores incremental backups and archived redo logs as needed.
not NULL		Unless the error is caused by a temporary hardware or O/S problem, restore the datafile or switch to a copy of that datafile.

Note: Because V\$DATAFILE_HEADER only reads the header block of each datafile it does not detect all problems that require the datafile to be restored. For example, Oracle reports no error if the datafile contains unreadable data blocks but its header block is intact.

Performing Complete Recovery

When performing complete recovery, recover either the whole database or a subset of the database. For example, you can perform a complete recovery of a majority of your tablespaces, and then recover the remaining tablespaces later. It makes no difference if the datafiles are read-write or offline normal.

The method you use for complete recovery depends on whether the database is open or closed.

If the database is	Then
Closed	Do one of the following: <ul style="list-style-type: none"> ■ Recover the whole database in one operation. ■ Recover individual tablespaces in separate operations.
Open	Do one of the following: <ul style="list-style-type: none"> ■ Close it and recover. ■ Take individual tablespaces offline and recover them.

The **skip** option is useful for avoiding recovery of tablespaces containing only temporary data or for postponing recovery of some tablespaces. The **skip** clause takes the datafiles in the specified tablespaces offline before starting media recovery and keeps them offline until after media recovery completes.

Issue at least one **allocate channel** command before you issue the **recover** command unless you do not need to restore archived redo log or incremental backup sets. Allocate the appropriate type of device for the backup sets that you want to restore. If the appropriate type of storage device is not available, then the **recover** command will fail.

Recovering the Database

The procedure for performing complete recovery on the database differs depending on whether the control file is available.

To recover the database when the control file is intact:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. If the database is open, shut it down, then mount it:

```
shutdown immediate;
startup mount;
```

3. After allocating channels, restore the database and recover it. This example skips the read-only TEMP tablespace:

```
run {
  allocate channel ch1 type disk;
  restore database;
  recover database
    skip tablespace temp;
}
```

4. Examine the output to see if recovery was successful. After RMAN restores the necessary datafiles, look for RMAN-08055 in the output:

```
RMAN-08024: channel ch1: restore complete
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete

RMAN-03022: compiling command: recover

RMAN-03022: compiling command: recover(1)

RMAN-03022: compiling command: recover(2)

RMAN-03022: compiling command: recover(3)
RMAN-03023: executing command: recover(3)
RMAN-08054: starting media recovery
RMAN-08515: archivelog filename=/oracle/arc_dest/arcr_1_40.arc thread=1 sequence=40
RMAN-08515: archivelog filename=/oracle/arc_dest/arcr_1_41.arc thread=1 sequence=41
RMAN-08055: media recovery complete

RMAN-03022: compiling command: recover(4)
RMAN-08031: released channel: ch1
```

To recover the database using a backup control file:

When you perform a restore operation using a backup control file and you use a recovery catalog, RMAN automatically adjusts the control file to reflect the structure of the restored backup.

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. If you use a recovery catalog, RMAN updates the control file. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. Start the instance without mounting the database:

```
startup nomount;
```

3. After allocating one or more channels, do the following:

- a. Use the **restore controlfile** command to restore the control file to all locations specified in the CONTROL_FILES initialization parameter.
- b. Mount the database.
- c. Restore and recover the database.
- d. Open the database with the RESETLOGS option.

```
run {  
    allocate channel ch1 type 'sbt_tape';  
    restore controlfile;  
    alter database mount;  
    restore database;  
    recover database;  
    sql "ALTER DATABASE OPEN RESETLOGS";  
}
```

4. Reset the database:

```
reset database;
```

5. Immediately back up the database. Because the database is a new incarnation, the pre-RESETLOGS backups are not usable. For example, enter:

```
run {  
    allocate channel ch1 type 'sbt_tape';  
    backup database;  
}
```

Recovering Tablespaces

The procedure for recovery tablespaces depends on whether the database is open or closed and whether the default tablespace location is accessible.

To recover an accessible tablespace when the database is closed:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. After allocating channels, restore the tablespace and recover it. This example recovers tablespace TBS_3:

```
run {
    allocate channel ch1 type disk;
    restore tablespace tbs_3;
    recover tablespace tbs_3;
}
```

3. Examine the output to see if recovery was successful.

To recover an inaccessible tablespace when the database is closed:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. After allocating channels, do the following:
 - a. Rename the damaged datafile, specifying an accessible location.
 - b. Restore the backup datafile to the new location.
 - c. Switch the restored datafile so that the control file considers it the current datafile.
 - d. Recover the tablespace.

```
run {
    allocate channel ch1 type disk;
    set newname for datafile '/disk1/oracle/tbs_1.f' to '/disk2/oracle/tbs_1.f';
    restore tablespace tbs_1;
    switch datafile all;
    recover tablespace tbs_1;
}
```

To recover an accessible tablespace while the database is open:

If a datafile is lost or corrupted but the disk is accessible, restore the datafile to its default location.

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. Do the following:
 - a. Take the tablespace that you want to recover offline.
 - b. Allocate channels.
 - c. Optionally, set a restore destination for archived redo logs necessary for recovery. Because RMAN is restoring the logs to this location, it knows where to find them.
 - d. Restore and then recover the tablespace.
 - e. Bring the tablespace online.

```
run {  
  sql 'ALTER TABLESPACE user_data OFFLINE TEMPORARY';  
  allocate channel chl type disk;  
  set archivelog destination to '/oracle/temp/arcl_restore';  
  restore tablespace user_data;  
  recover tablespace user_data;  
  sql 'ALTER TABLESPACE user_data ONLINE';  
}
```

To recover an inaccessible tablespace while the database is open:

If a tablespace or datafile is inaccessible because of media failure, restore the datafile to a new location or switch to an existing datafile copy.

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

2. After allocating channels, do the following:
 - a. Take the tablespace that you want to recover offline.
 - b. Rename the damaged datafile, specifying an accessible location.
 - c. Restore the backup datafile to the new location.

- d. Switch the restored datafile so that the control file considers it the current datafile.
- e. Recover the tablespace.
- f. Bring the tablespace online.

```
run {
  sql 'ALTER TABLESPACE user_data OFFLINE IMMEDIATE';
  allocate channel ch1 type disk;
  set newname for datafile '/disk1/oracle/tbs_1.f' to '/disk2/oracle/tbs_1.f';
  restore tablespace tbs_1;
  switch datafile all;
  recover tablespace tbs_1;
  sql 'ALTER TABLESPACE tbs_1 ONLINE';
}
```

Performing Incomplete Recovery

RMAN allows you to perform recovery of the whole database to a specified non-current time, SCN, or log sequence number. This type of recovery is called *incomplete recovery*; if it is recovery of the whole database, it is sometimes called *database point-in-time recovery (DBPITR)*.

Incomplete recovery differs in several ways from complete recovery. The most important difference is that incomplete recovery requires you to open the database with the `RESETLOGS` option. Using this option gives the online redo logs a new timestamp and SCN, thereby eliminating the possibility of corrupting your datafiles by the application of obsolete archived redo logs.

Because you must open `RESETLOGS` after performing incomplete recovery, you have to recover all datafiles. You cannot recover some datafiles before the `RESETLOGS` and others after the `RESETLOGS`. In fact, Oracle will prevent you from resetting the logs if a datafile is offline. The only exception is if the datafile is offline normal or read-only. You can bring files in read-only or offline normal tablespaces online after the `RESETLOGS` because they do not need any redo applied to them.

The easiest way to perform *database point-in-time recovery (DBPITR)* is to use the `set until` command, which sets the desired time for any subsequent `restore`, `switch`, and `recover` commands in the same `run` job. Note that if you specify a `set until` command after a `restore` and before a `recover`, you may not be able to recover the database to the point in time required, since the restored files may already have timestamps more recent than the set time. Hence, it is usually best to specify the `set until` command *before* the `restore` or `switch` command.

See Also: For `set until` command syntax, see "[untilClause](#)" on page 11-146.

Performing Incomplete Recovery with a Recovery Catalog

The database must be closed to perform database point-in-time recovery. Note that if you are recovering to a time, you should set the time format environment variables before invoking RMAN (see ["Setting NLS Environment Variables"](#) on page 5-2). For example, enter:

```
NLS_LANG=american
NLS_DATE_FORMAT='Mon DD YYYY HH24:MI:SS'
```

To recover the database until a specified time:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

Optionally, specify a log file at connect time:

```
% rman target / catalog rman/rman@rcat log = rman_log
```

2. If the database is open, shut it down and then mount it:

```
shutdown immediate;
startup mount;
```

3. Determine which time you want to recover to. For example, if you discover at 9:15 a.m. that a user accidentally dropped a tablespace at 9:02 a.m., then you can recover to 9 a.m.—just before the drop occurred. You will lose all changes to the database made since that time.
4. Perform the following operations within your **run** command:
 - a. Set the end recovery time using the date format specified in your NLS_LANG and NLS_DATE_FORMAT environment variables.
 - b. Allocate channels.
 - c. Restore the database.
 - d. Recover the database.
 - e. Open the database with the RESETLOGS option.

For example, this job performs an incomplete recovery until Nov 15 at 9 a.m.

```
run {
  set until time 'Nov 15 1998 09:00:00';
  allocate channel ch1 type 'sbt_tape';
  restore database;
  recover database;
```



```

    sql 'ALTER DATABASE OPEN RESETLOGS';
}

```

5. Reset the database:

```
reset database;
```

6. Immediately back up the database. Because the database is a new incarnation, the pre-RESETLOGS backups are not usable. For example, enter:

```

run {
    allocate channel ch1 type 'sbt_tape';
    backup database;
}

```

To recover the database until a specified SCN:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

Optionally, specify a message log file at connect time:

```
% rman target / catalog rman/rman@rcat log = rman_log
```

2. If the database is open, shut it down and then mount it:

```

shutdown immediate;
startup mount;

```

3. Determine the SCN to which you want to recover. For example, if you made a backup of tablespace TBS_1 and then shortly afterwards a user accidentally overwrote a datafile in TBS_3, then you can issue a **list command to determine the SCN for the TBS_1 backup and then restore yesterday's whole database backup and recover to that SCN.**

4. Perform the following operations within your run command:

- a. Set the end recovery SCN.
- b. Allocate channels.
- c. Restore the database.
- d. Recover the database.
- e. Open the database with the RESETLOGS option.

For example, this job performs an incomplete recovery until SCN 1000.

```
run {
  set until scn 1000;
  allocate channel chl type 'sbt_tape';
  restore database;
  recover database;
  sql 'ALTER DATABASE OPEN RESETLOGS';
}
```

5. Reset the database:

```
reset database;
```

6. Immediately back up the database. Because the database is a new incarnation, the pre-RESETLOGS backups are not usable. For example, enter:

```
run {
  allocate channel chl type 'sbt_tape';
  backup database;
}
```

To recover the database until a specified log sequence number:

1. Start RMAN and connect to the target database and, optionally, the recovery catalog database. For example, enter:

```
% rman target / catalog rman/rman@rcat
```

Optionally, specify a message log file at connect time:

```
% rman target / catalog rman/rman@rcat log = rman_log
```

2. If the database is open, shut it down and then mount it:

```
shutdown immediate;
startup mount;
```

3. Determine the log sequence number to which you want to recover. For example, query V\$LOG_HISTORY to view the redo logs that you have archived.

RECID	STAMP	THREAD#	SEQUENCE#	FIRST_CHAN	FIRST_TIM	NEXT_CHANG
1	344890611	1	1	20037	24-SEP-98	20043
2	344890615	1	2	20043	24-SEP-98	20045
3	344890618	1	3	20045	24-SEP-98	20046
4	344890621	1	4	20046	24-SEP-98	20048
5	344890624	1	5	20048	24-SEP-98	20049
6	344890627	1	6	20049	24-SEP-98	20050
7	344890630	1	7	20050	24-SEP-98	20051
8	344890632	1	8	20051	24-SEP-98	20052
8 rows selected.						

4. Perform the following operations within your **run** command:
 - a. Set the log sequence number for recovery termination.
 - b. Allocate channels.
 - c. Restore the database.
 - d. Recover the database.
 - e. Open the database with the RESETLOGS option.

For example, this job performs an incomplete recovery until log sequence number 6 on thread 1:

```
run {
  set until logseq 6 thread 1;
  allocate channel chl type 'sbt_tape';
  restore database;
  recover database;
  sql 'ALTER DATABASE OPEN RESETLOGS';
}
```

5. Reset the database:

```
reset database;
```

6. Immediately back up the database. Because the database is a new incarnation, the pre-RESETLOGS backups are not usable. For example, enter:

```
run {
  allocate channel chl type 'sbt_tape';
  backup database;
}
```

Performing Incomplete Recovery Without a Recovery Catalog

Although you can perform DBPITR without a recovery catalog, be sure to follow these directions:

- Take a separate backup of the control file after executing all RMAN backup commands.
- Save the RMAN output for all backups.

Backing Up the Control File Separately Make a backup of the control file after your RMAN database backups because you need a backup control file that contains information about the database backup that you just made. Even if your database

backup included backing up the control file, as it does if you back up `datafile 1` or specify **include current controlfile**, the backup control file contained in the backup set is not self-referential. Consider this command:

```
backup database;
```

This command produces a backup set that contains a backup of the control file. The backup control file does not contain any record for the backup set in which it is itself contained. Consequently, if you restore this backup control file and then mount it, you will not be able to restore files out of the backup set since the control file has no record of them.

To back up the control file separately, issue the following sequence of commands within your **run** command:

```
backup database;
backup current controlfile; # obtain a useful control file backup.
```

These commands will create two backup sets, each of which contains a backup control file. The control file backup created by the second command will be the useful one, i.e., it will contain all the records related to the database backup.

To perform DBPITR without a recovery catalog:

1. Start RMAN and connect to the target database, specifying the **nocatalog** option:

```
% rman target / nocatalog
```

2. If the database is open, shut it down and then mount it:

```
startup force mount;
```

3. Restore a backup control file to a temporary location. If you created a separate control file backup as suggested, RMAN will usually restore this backup. RMAN will only choose the wrong backup control file if you specify a time that was in the interval between the **backup database** and the **backup current controlfile**.

If you saved all the RMAN output as suggested, then you can verify that the backup control file that RMAN picked was the correct one. Alternatively, you can use the **tag** option on the **backup current controlfile** command, and then specify this tag on the restore to force RMAN to pick the control file you want.

For example, issue the following command to restore the control file to a temporary location:

```
run {
  set until time 'Jun 18 1998 16:32:36';
  allocate channel chl type disk;
  # restore a backup controlfile to a temporary location.
  restore controlfile to '/tmp/cf.tmp';
}
```

Verify that the control file that RMAN restored was one created via the **backup current controlfile** command that followed all backups. Alternatively, if you specified a tag on the **backup current controlfile** command, specify the **from tag** option on the **restore controlfile** command.

4. Save a copy of the current control file, and then replace it with the backup control file that you restored in the previous step:

```
run {
  allocate channel chl type disk;
  # save a copy of the current controlfile just to be safe
  copy current controlfile to '/tmp/original.cf';
  shutdown immediate;
  startup nomount;
  replicate controlfile from '/tmp/cf.tmp';
  alter database mount;
}
```

5. Execute the following operations:
 - a. Set an end time, SCN, or log sequence number (if you run in ARCHIVELOG mode) for recovery.
 - b. Restore and recover the database. If the database is running in NOARCHIVELOG mode, specify the **noredo** option on the **recover** command. If the database runs in ARCHIVELOG mode, then omit the **noredo** option.
 - c. Open the database with the **RESETLOGS** option

```
run {
  set until time 'Jun 18 1998 16:32:36';
  restore database;
  recover database noredo;
  sql 'ALTER DATABASE OPEN RESETLOGS';
}
```

6. Reset the database:

```
reset database;
```

7. Immediately back up your database.

```
run {  
    allocate channel ch1 type disk;  
    backup database;  
}
```

Restore and Recovery Scenarios

Following are useful scenarios for performing restore and recovery operations:

- [Using Datafile Copies to Restore to a New Host](#)
- [Restoring when Multiple Databases Share the Same Name](#)
- [Performing an Irregular Restore of the Control File from a Backup Set](#)
- [Recovering an Inaccessible Datafile in an Open Database](#)
- [Recovering an Inaccessible Datafile Using Backups from Disk and Tape](#)
- [Performing Recovery After a Total Media Failure](#)
- [Recovering a Pre-RESETLOGS Backup](#)
- [Recovering a Database in NOARCHIVELOG Mode](#)

Using Datafile Copies to Restore to a New Host

If you wish to move the database to a new host using datafile copies, you must transfer the copies manually to the new machine. This example assumes that you are using a recovery catalog.

1. After connecting to your target database and recovery catalog, issue a **list** command to see a listing of your datafile copies and their associated primary keys:

```
list copy;
```

2. Copy the datafile copies to the new host using an O/S utility. For example, a UNIX user could enter:

```
% cp -r /oracle/copies /net/new_host/oracle/dbs
```

3. Uncatalog the datafile copies on the old host. For example, enter:

```
change datafile copy 1,2,3,4,5,6,7,9,10 uncatalog;
```

4. Catalog the transferred datafile copies, using their new filenames. For example, enter:

```
catalog datafilecopy '/oracle/dbs/tbs_1.f', '/oracle/dbs/tbs_2.f',
'/oracle/dbs/tbs_3.f', '/oracle/dbs/tbs_4.f', '/oracle/dbs/tbs_5.f',
'/oracle/dbs/tbs_6.f', '/oracle/dbs/tbs_7.f', '/oracle/dbs/tbs_8.f',
'/oracle/dbs/tbs_9.f', '/oracle/dbs/tbs_10.f';
```

5. Perform the restore and recovery operation described in ["Moving the Target Database to a New Host with the Same Filesystem"](#) on page 9-4 or ["Moving the Target Database to a New Host with a Different Filesystem"](#) on page 9-7. Specify a channel of type **disk** rather than type *'sbt_tape'*.

Restoring when Multiple Databases Share the Same Name

The database identifier is a 32-bit number that is computed when the database is created. If you want to restore a database that shares a name with another database, you must distinguish it. Use the RMAN **set dbid** command to specify a database according to its database identifier.

Obtaining the DBID of a Database You Want to Restore

If you have saved your RMAN output, refer to this information to determine the database identifier, since RMAN automatically provides it whenever you connect to the database:

```
% rman target /
```

```
Recovery Manager: Release 8.1.5.0.0
```

```
RMAN-06005: connected to target database: RMAN (DBID=1231209694)
```

If you have not saved your RMAN output and need the DBID value of a database for a restore operation, obtain it via the `RC_DATABASE` or `RC_DATABASE_INCARNATION` recovery catalog views.

Because the names of the databases that are registered in the recovery catalog are presumed non-unique in this scenario, some other unique piece of information must be used to determine the correct DBID. If you know the filename of a datafile or online redo log associated with the database you wish to restore, and this filename is unique across all databases registered in the recovery catalog, then

substitute this fully-specified filename for *filename_of_log_or_df* in the queries below. Determine the DBID by performing one of the following queries:

```
SELECT distinct db_id
FROM db, dbinc, dfatt
WHERE db.db_key = dbinc.db_key
      AND dbinc.dbinc_key = dfatt.dbinc_key
      AND dfatt.fname = 'filename_of_log_or_df';
```

```
SELECT distinct db_id
FROM db, dbinc, orl
WHERE db.db_key = dbinc.db_key
      AND dbinc.dbinc_key = orl.dbinc_key
      AND orl.fname = 'filename_of_log_or_df';
```

Restoring a Backup Control File Using the DBID

Only use the **set dbid** command to restore the control file when all of these conditions are met:

- The control file has been lost and must be restored from a backup.
- You are using a recovery catalog.
- Multiple databases registered in the recovery catalog share a database name.

If these conditions are not met, you will receive the `RMAN-20005: target database name is ambiguous` message when you attempt to restore the control file. RMAN will correctly identify the control file to restore, so you do not need to use the **set dbid** command.

RMAN accepts **set dbid** only if you have not yet connected to the target database, i.e., **set dbid** must precede the **connect target** command. If the target database is mounted, then RMAN verifies that the user-specified DBID matches the DBID from the database; if not, RMAN signals an error. If the target database is not mounted, RMAN uses the user-specified DBID to restore the control file. After restoring the control file, you can mount the database to restore the rest of the database.

To set the database id enter the following, where *target_dbid* is an integer value:

```
set dbid = target_dbid;
```

To restore the control file to its default location enter:

```
run {
  allocate channel dev1 type 'sbt_tape';
  restore controlfile;
  alter database mount;
}
```


Performing an Irregular Restore of the Control File from a Backup Set

You will be forced to use a non-standard procedure to restore a control file from an RMAN backup set in the following situations:

- You are using a pre-8.0.5 version of RMAN to restore a database when more than one database with the same name is registered in the recovery catalog (see ["Restoring when Multiple Databases Share the Same Name"](#) on page 9-31 for a discussion of this problem).
- You are not using a recovery catalog, and your only control file backup is in an RMAN backup set.

If you have no other backup of the control file except in a RMAN backup set, and you need the control file to perform a restore operation, use the following PL/SQL program to extract the control file from the backup set.

Run this program from SQL*Plus while connected as SYSDBA to the target database:

```

DECLARE
    devtype varchar2(256);
    done    boolean;
BEGIN
    devtype := dbms_backup_restore.deviceallocate('devtype', params=>'');
    # Replace 'devtype' with the device type you used when creating the backup: disk or
    # sbt_tape. If you used an sbt_tape device and specified a 'parms' option on the RMAN
    # allocate channel command, then put that parms data in the 'params' operand here.

    dbms_backup_restore.restoresetdatafile;

    dbms_backup_restore.restorecontrolfileto('/tmp/foo.cf');
    # This path specifies the location for the restored control file. If there are multiple
    # control files specified in the init.ora file, copy the control file to all specified
    # locations before mounting the database.

    dbms_backup_restore.restorebackuppiece('handle',done=>done);
    # Replace 'handle' with the your backup piece handle. This example assumes that the
    # backup set contains only one backup piece. If there is more than one backup piece in
    # the backup set (which only happens if the RMAN command set limit kbytes is used), then
    # repeat the restorebackuppiece statement for each backup piece in the backup set.

END;
/

```

Once you have successfully restored the control file, you can mount the database and perform restore and recovery operations.

Recovering an Inaccessible Datafile in an Open Database

In this scenario, the database is open but you cannot access a datafile. You execute the following SQL query to determine its status:

```
SELECT * FROM v$recover_file;

FILE# ONLINE  ERROR          TIME
-----
19 ONLINE  FILE NOT FOUND
```

You then decide to start RMAN and connect to the target and recovery catalog databases:

```
% rman target / catalog rman/rman@rcat
```

You issue a **report** command to determine the datafile's tablespace and filename:

```
RMAN> report schema;

RMAN-03022: compiling command: report
Report of database schema
File K-bytes  Tablespace          RB segs Name
-----
1          47104 SYSTEM             YES  /oracle/dbs/tbs_01.f
2           978 SYSTEM             YES  /oracle/dbs/tbs_02.f
3           978 TBS_1              NO   /oracle/dbs/tbs_11.f
4           978 TBS_1              NO   /oracle/dbs/tbs_12.f
5           978 TBS_2              NO   /oracle/dbs/tbs_21.f
6           978 TBS_2              NO   /oracle/dbs/tbs_22.df
7           500 TBS_1              NO   /oracle/dbs/tbs_13.f
8           500 TBS_2              NO   /oracle/dbs/tbs_23.f
9           500 TBS_2              NO   /oracle/dbs/tbs_24.f
10          500 TBS_3              NO   /oracle/dbs/tbs_31.f
11          500 TBS_3              NO   /oracle/dbs/tbs_32.f
12          500 TBS_4              NO   /oracle/dbs/tbs_41.f
13          500 TBS_4              NO   /oracle/dbs/tbs_42.f
14          500 TBS_5              YES  /oracle/dbs/tbs_51.f
15          500 TBS_5              YES  /oracle/dbs/tbs_52.f
16          5120 SYSTEM            YES  /oracle/dbs/tbs_03.f
17          2048 TBS_1              NO   /oracle/dbs/tbs_14.f
18          2048 TBS_2              NO   /oracle/dbs/tbs_25.f
19          2048 TBS_3              NO   /oracle/dbs/tbs_33.f
20          2048 TBS_4              NO   /oracle/dbs/tbs_43.f
21          2048 TBS_5              YES  /oracle/dbs/tbs_53.f
```

Because you need to take the datafile online immediately before you investigate the media failure, you decide to restore the datafile to a new location and switch to a copy of that datafile:

```

run {
  sql 'ALTER TABLESPACE tbs_3 OFFLINE IMMEDIATE';
  allocate channel chl type disk;
  set newname for datafile '/oracle/dbs/tbs_33.f' to '/oracle/temp/tbs_33.f';
  restore tablespace tbs_3;
  switch datafile all;
  recover tablespace tbs_3;
  sql 'ALTER TABLESPACE tbs_3 ONLINE';
}

```

Recovering an Inaccessible Datafile Using Backups from Disk and Tape

If you cannot access datafiles due to a disk failure, it is likely that you must restore it to a new location or switch to an existing datafile copy. The following restore example restores and recover tablespace TBS_1, which contains four datafiles. Since some copies of these files are on disk and some backups on tape, the example allocates one disk channel and one media management channel to allow **restore** to restore from both disk and tape:

```

run {
  allocate channel dev1 type disk;
  allocate channel dev2 type 'sbt_tape';
  sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE";
  set newname for datafile '/disk7/oracle/tbs11.f'
    to '/disk9/oracle/tbs11.f';
  set newname for datafile '/disk7/oracle/tbs12.f'
    to '/disk9/oracle/tbs12.f';
  set newname for datafile '/disk7/oracle/tbs13.f'
    to '/disk9/oracle/tbs13.f';
  set newname for datafile '/disk7/oracle/tbs14.f'
    to '/disk9/oracle/tbs14.f';
  restore tablespace tbs_1;
  switch datafile all;      # makes the renamed datafile the current datafile
  recover tablespace tbs_1;
  sql "ALTER TABLESPACE tbs_1 ONLINE";
}

```

Performing Recovery After a Total Media Failure

The following scenario assumes:

- You have lost the whole database, all control files, the online redo logs, and recovery catalog.
- You are restoring the database to its original location after fixing the media problem.

- There are four tape drives.
- You are using a recovery catalog.

Before restoring the database, you must:

- Restore your `init.ora` file, password file (if you use one), and your recovery catalog from your most recent backup using O/S commands or utilities.
- Catalog any archived redo logs, datafile copies, or backup sets that are on disk, but are not registered in the recovery catalog. The archived redo logs up to the logseq number being restored to must be cataloged in the recovery catalog, or Recovery Manager will not know where to find them. If you resync the recovery catalog frequently, and have an up-to-date copy from which you have restored, you should not have many archived redo logs that need cataloging.

The following scenario restores and recovers the database to the most recently available archived log, which is log 124 in thread 1. It:

- Starts the instance without mounting the database and restricts connections to DBA-only users
- Restores the control file to the locations specified by the `init.ora` parameter `CONTROL_FILES`.
- Mounts the control file.
- Catalogs any archived redo logs not in the recovery catalog.
- Restores the database files to their original locations. If volume names have changed, use the **set newname** command before the restore and perform a switch after the restore to update the control file with the new locations for the datafiles.
- Recovers the datafiles by either using a combination of incremental backups and redo, or just redo. RMAN will stop recovery when it reaches the log sequence number specified.
- Opens the database in RESETLOGS mode. Only complete this last step if you are certain that no other archived redo logs can be applied.
- Oracle recommends that you back up your database after the RESETLOGS (this is not shown in the example).

```

% rman target sys/sys_pwd@prod1 catalog rman/rman@rcat
startup nomount dba;

run {
  # If you need to restore the files to new locations, tell Recovery Manager
  # to do this using 'set newname'.
  # set newname for datafile 1 to '/dev/vgd_1_0/rlvt5_500M_1';
  # set newname for datafile 2 to '/dev/vgd_1_0/rlvt5_500M_2';
  # set newname for datafile 3 to '/dev/vgd_1_0/rlvt5_500M_3';
  # set newname for datafile 4 to '/dev/vgd_1_0/rlvt5_500M_4';
  # etc...

  # The set until command is used in case the database
  # structure has changed in the most recent backups, and you wish to
  # recover to that point-in-time. In this way Recovery Manager restores
  # the database to the same structure the database was at that time.
  set until logseq 124 thread 1;

  allocate channel t1 type 'SBT_TAPE';
  allocate channel t2 type 'SBT_TAPE';
  allocate channel t3 type 'SBT_TAPE';
  allocate channel t4 type 'SBT_TAPE';

  restore controlfile;
  alter database mount;

  # Catalog any archivelogs that are not in the recovery catalog
  # catalog archivelog '/oracle/db_files/prod1/arch/arch_1_123.rdo';
  # catalog archivelog '/oracle/db_files/prod1/arch/arch_1_124.rdo';
  # etc...
  restore database;

  # Update the control file by telling it the new location of the datafiles
  # only if you used 'set newname for datafile' above.
  # switch datafile all;
  recover database;

  # Complete this last step only if no more archived logs need to be applied.
  sql 'ALTER DATABASE OPEN RESETLOGS';
}

```

Recovering a Pre-RESETLOGS Backup

Assume the following:

- You use a recovery catalog.
- You made a backup of PROD1 on July 2, 1998.

- You performed incomplete recovery on this database and opened it with the RESETLOGS option on July 10, 1998. A new database incarnation was created.
- On July 25, you discover that you need crucial data that was dropped from the database at 8 a.m. on July 8, 1998.
- You decide to reset PROD1 to the prior incarnation, restore the July 2 backup, and recover to 7:55 a.m. on July 8.

```
# obtain primary key of old incarnation
list incarnation of database prod1;

List of Database Incarnations
DB Key  Inc Key  DB Name  DB ID      CUR   Reset SCN  Reset Time
-----  -
1        2        PROD1    1224038686 NO     1          02-JUL-98
1        582      PROD1    1224038686 YES    59727      10-JUL-98

# reset database to old incarnation
reset database to incarnation 2;
# recover it
run {
    set until time 'Jul 8 1998 07:55:00';
    allocate channel dev1 type disk;
    shutdown abort;
    startup nomount;
    restore controlfile;
    alter database mount;
    restore database;
    recover database;
    sql 'alter database open resetlogs';
}
# make this new incarnation the current incarnation
reset database;
```

Recovering a Database in NOARCHIVELOG Mode

You can recover a database running in NOARCHIVELOG mode using incremental backups. Assume the following scenario:

- You run database PROD1 in NOARCHIVELOG mode.
- You use a recovery catalog.
- You made a level 0 backup of database PROD1 to tape on Monday.
- You make a level 1 differential incremental backups to tape at 7 a.m. on Wednesday and Friday.

- The database suffers a media failure on Sunday, causing you to lose half of the datafiles.

In this case, you are forced to perform an incomplete media recovery until Friday, since that is the date of your most recent incremental backup. Note that RMAN always looks for incremental backups before looking for archived logs during recovery.

RMAN can perform the desired incomplete media recovery automatically if you specify the **noredo** option in the **recover** command. If you do not specify **noredo**, RMAN will search for archived redo logs after applying the Friday incremental backup, and issue an error message when it does not find them.

After connecting to PROD1 and the catalog database, recover the database using the following command:

```
run {
  allocate channel dev1 type 'sbt_tape';
  restore database;
  recover database noredo;
  sql 'ALTER DATABASE OPEN RESETLOGS';
}
```

Creating a Duplicate Database with Recovery Manager

This chapter describes how to use Recovery Manager to create a duplicate database for testing purposes, and includes the following topics:

- [Creating a Duplicate Database: Overview](#)
- [Creating a Duplicate Database on a Local or Remote Host](#)
- [Duplication Scenarios](#)

Creating a Duplicate Database: Overview

The RMAN **duplicate** command allows you to use your target database backups to create a test database while still retaining your original database. The command takes image copies or backup sets of your target database's files and generates a new database. A duplicate database is especially useful if your production database must be up and running 24 hours per day, 7 days a week.

As part of the duplicating operation, RMAN manages the following:

- Restores the target datafiles into the duplicate database and performs incomplete recovery using all available archived redo logs and incremental backups.
- Opens the duplicate database with the RESETLOGS option after incomplete recovery to create the online redo logs.
- Generates a new, unique database identifier for the duplicate database.

When duplicating a database you can:

- Skip read-only tablespaces with the **skip readonly** clause. Read-only tablespaces are included by default. If you omit them, you can add them later.
- Create your duplicate database in a new host. If the directory structure is the same on the new host, you can use the **nofilenamecheck** option and re-use the target datafile filenames for the duplicate datafiles.
- Use the **set until** option when creating the duplicate database to recover it to a non-current time. By default, the **duplicate** command creates the database using the most recent backups of the target database and then performs recovery to the most recent consistent point contained in the incremental and archived redo log backups.
- Use the **duplicate** command without a recovery catalog.
- Register the duplicate database in the same recovery catalog as the target database. This option is possible because the duplicate database receives a new database identifier during duplication. If you copy the target database using O/S utilities, the database identifier of the copied database remains the same so you cannot register it in the same recovery catalog.

See Also: For **duplicate** command syntax, see "[duplicate](#)" on page 11-69.

Obeying Restrictions

RMAN duplication has the following restrictions. You *cannot*:

- Duplicate offline tablespaces, although you can add these tablespaces manually after duplication.
- Duplicate a database when some backups of the target do not exist. RMAN attempts to duplicate the following:
 - All datafiles in online tablespaces, whether or not the datafiles are online.
 - All tablespaces taken offline with the IMMEDIATE option.

If no valid backups exist of any tablespace or datafile, the command fails.

- Use **nofilenamecheck** when the names of the duplicate datafiles are the same as the target datafiles and the databases are in the same host. If you do specify **nofilenamecheck**, RMAN may overwrite the datafiles or signal the following error:

```
RMAN-10035: exception raised in RPC: ORA-19504: failed to create file
           "/oracle/dbs/tbs_01.f"
ORA-27086: skgfglk: unable to lock file - already in use
SVR4 Error: 11: Resource temporarily unavailable
Additional information: 8
RMAN-10031: ORA-19624 occurred during call to
DBMS_BACKUP_RESTORE.RESTOREBACKUPPIECE
```

Note that if you want the filenames to be the same, and the databases are in *different* hosts, then you must use the **nofilenamecheck** option.

- Duplicate when the target database is not open.
- Use image copies located on one host to create a duplicate database on a new host. You must use tape backups instead.

Generating Files for the Duplicate Database

When duplicating a database, perform the following operations:

- [Creating the Control Files](#)
- [Creating the Online Redo Logs](#)
- [Renaming the Datafiles](#)

Creating the Control Files

The **duplicate** command creates the control files by using the names listed in the `init.ora` file of the duplicate database. When choosing names for the duplicate database control files, make sure that you do not overwrite the `init.ora` settings for the production files at the target database.

Creating the Online Redo Logs

You have these options for creating the names of the duplicate online redo logs, which are listed in the order of precedence:

Table 10–1 Order of Precedence for Redo Log Filename Creation

Order	Method	Result
1	Specify the logfile clause of duplicate command.	Creates redo logs as specified.
2	Set <code>LOG_FILE_NAME_CONVERT</code> initialization parameter.	Transforms target filenames, e.g., from <code>log_*</code> to <code>duplog_*</code> . Note: This parameter allows the redo log to exist as long as the size matches, since it uses the reuse parameter when creating the logs.
3	Do none of the above.	Reuses the target filenames. You must specify the nofilenamecheck option when using this method.

The order of precedence determines how RMAN renames the online redo logs. For example, if you specify both the **logfile** clause and the `LOG_FILE_NAME_CONVERT` parameter, RMAN uses the **logfile** clause. If you specify all options, then RMAN uses the **logfile** clause and ignores the others.

WARNING: If the target and duplicate databases are in the same host, do not use the name of an online redo log currently in use by the target database. Also, do not use the name of a redo log currently in use by the target database if the duplicate database is in a different host and the `nofilenamecheck` keyword is not used.

Renaming the Datafiles

If you want to have different filenames in your duplicate datafile, then you must use parameters or commands to specify them. You have these options for renaming datafiles, listed in the order of precedence:

Table 10–2 Order of Precedence for Datafile Filename Creation

Order	Method	Result
1	Issue set newname command.	Creates new datafile filenames. This command must be re-issued each time you want to rename files.
2	Issue set auxname command.	Creates new datafile filenames. This setting stays in effect until disabled with a set auxname ... to null command.
3	Set DB_FILE_NAME_CONVERT initialization parameter.	Transforms target filenames, e.g., from <code>tbs_*</code> to <code>dupdbs_*</code> . You can use this parameter for those files not renamed with either set newname and set auxname .
4	Do none of the above.	Re-uses the target filenames. You must specify the nofilenamecheck option when using this method.

The order of precedence determines how RMAN will name the datafiles. For example, if you specify all the commands and the initialization parameter, RMAN uses **set newname**. If you specify the **set auxname** command and **DB_FILE_NAME_CONVERT**, RMAN uses **set auxname**. If you do not specify any of the first three options, then RMAN uses the original target filenames for the duplicate file.

Skipping Read Only Tablespaces When you specify **skip readonly**, RMAN does not duplicate the datafiles of these tablespaces. You will see the following values in the specified views or tables:

Table/View	Column	Value
V\$DATAFILE	STATUS	OFFLINE
V\$DATAFILE	ENABLED	READ ONLY
V\$DATAFILE	NAME	MISSINGxxx
SYS.DBA_DATA_FILES	STATUS	AVAILABLE
SYS.DBA_TABLESPACES	STATUS	READ ONLY

Skipping Offline Clean Tablespaces When tablespaces are taken offline with the `OFFLINE NORMAL` option, RMAN does not duplicate the datafiles of these tablespaces. After duplication, you can manually add or drop these tablespaces.

You will see the following values in the specified views or tables:

Table/View	Column	Value
V\$DATAFILE	STATUS	OFFLINE
V\$DATAFILE	ENABLED	DISABLED
V\$DATAFILE	NAME	MISSINGxxx
SYS.DBA_DATA_FILES	STATUS	AVAILABLE
SYS.DBA_TABLESPACES	STATUS	OFFLINE

Note that when you take a tablespace offline with the `IMMEDIATE` option, RMAN duplicates rather than skips the tablespace. As with online tablespaces, RMAN requires a valid backup for duplication.

Preventing Filename Checking It is possible for a `set newname`, `set newname`, or `DB_FILE_NAME_CONVERT` to generate a name that is already in use in the target database. In this case, specify `nofilenamecheck` to avoid an error. For example, assume that the host A database has two files: datafile 1 is named `/oracle/data/file1.f` and datafile 2 is named `/oracle/data/file2.f`. When duplicating to host B, you issue:

```
run {
  set newname for datafile 1 to /oracle/data/file2.f; # rename datafile 1 as file2.f
  set newname for datafile 2 to /oracle/data/file1.f; # rename datafile 2 as file1.f
  allocate ...
  duplicate target database to newdb;
}
```

Even though you issued `set newname` commands for all your datafiles, the `duplicate` command will fail because the duplicate filenames are still in use in the target database. Although datafile 1 in the target is not using `/oracle/data/file2.f`, and datafile 2 in the target is not using `/oracle/data/file1.f`, the target filename is used by one of the duplicate datafiles and so you must specify `nofilenamecheck` to avoid an error.

Note: Only use `DB_FILE_NAME_CONVERT` without using either `set newname` or `set auxname` if all the datafiles will be converted by this parameter, i.e., all of the datafiles have the same suffix or prefix.

Preparing the Auxiliary Instance for Duplication

Satisfy the following requirements before performing RMAN duplication:

- [Create an Oracle Password File for the Auxiliary Instance](#)
- [Create a Parameter File for the Auxiliary Instance](#)
- [Start the Auxiliary Instance](#)
- [Ensure Net8 Connectivity to the Auxiliary Instance](#)
- [Open the Target Database](#)
- [Start the RMAN Command Line Interface](#)
- [Make Sure You Have the Necessary Backups and Archived Redo Logs](#)
- [Allocate Auxiliary Channels](#)

Create an Oracle Password File for the Auxiliary Instance

For information about creating and maintaining Oracle password files, see the *Oracle8i Administrator's Guide*.

Create a Parameter File for the Auxiliary Instance

Create an `init.ora` file for the auxiliary instance and set the following required parameters:

Parameter	Specify:
DB_NAME	The same name that you use in the duplicate command.
CONTROL_FILES	See " Creating the Control Files " on page 10-4.

Optionally, set the following parameters:

Parameter	Specify:
DB_FILE_NAME_CONVERT	See "Renaming the Datafiles" on page 10-5.
LOG_FILE_NAME_CONVERT	See "Creating the Online Redo Logs" on page 10-4.

Set other parameters, including the parameters that allow you to connect as SYSDBA through Net8, as needed. When duplicating to the same host or to a new host with a different filesystem, pay special attention to all parameters specifying pathnames.

Following are examples of the `init.ora` parameter settings for the duplicate database:

```
DB_NAME=newdb
CONTROL_FILES=(/oracle/dup_prod/cf/cf1.f,/oracle/dup_prod/cf/cf2.log)
DB_FILE_NAME_CONVERT=(/oracle/prod/db,/oracle/dup_prod/db)
LOG_FILE_NAME_CONVERT=( "/oracle/prod/log", "/oracle/dup_prod/log")
```

See Also: For more information about Net8, see the *Net8 Administrator's Guide*.

Start the Auxiliary Instance

Before beginning RMAN duplication, use SQL*Plus to connect to the auxiliary instance and start it in NOMOUNT mode (specifying a parameter file if necessary). In this example, `aux_pwd` is the password for the user with SYSDBA authority and `aux_str` is the net service name for the auxiliary instance:

```
SQL> connect sys/aux_pwd@aux_str
SQL> startup nomount pfile='/oracle/aux/dbs/initAUX.ora';
```

Because the auxiliary instance does not yet have a control file, you can only start the instance in NOMOUNT mode. Do not create a control file or try to mount or open the auxiliary instance.

Ensure Net8 Connectivity to the Auxiliary Instance

The auxiliary instance must be accessible via Net8. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.

Open the Target Database

Before beginning RMAN duplication, open the target database (specifying a parameter file if necessary) if it is not already open. For example, enter:

```
SQL> startup pfile='/oracle/dbs/initPRODL.ora';
```

Start the RMAN Command Line Interface

Use one of the following methods to start the RMAN command line interface:

- [Connect at the O/S Command Line](#)
- [Connect at the RMAN Prompt](#)

Connect at the O/S Command Line You must connect to the auxiliary instance with SYSDBA privileges, so you must use a password file. To connect to the auxiliary instance, target instance, and recovery catalog, supply the following information when starting up Recovery Manager:

```
% rman target sys/target_pwd@target_str catalog rman/cat_pwd@cat_str auxiliary \
> sys/aux_pwd@aux_str
```

Where:

<i>target_pwd</i>	The password for connecting as SYSDBA specified in the target database's <code>orapwd</code> file
<i>target_str</i>	The net service name for the target database
<i>cat_pwd</i>	The password for user RMAN specified in the recovery catalog's <code>orapwd</code> file
<i>cat_str</i>	The net service name for the recovery catalog database
<i>aux_pwd</i>	The password for connecting as SYSDBA specified in the auxiliary database's <code>orapwd</code> file.
<i>aux_str</i>	The net service name for the auxiliary database.

Connect at the RMAN Prompt You can start the RMAN command line interface without a connection to the auxiliary instance, and then use the **connect auxiliary** command at the RMAN prompt to make the auxiliary connection:

```
% rman
RMAN> connect auxiliary sys/aux_pwd@aux_str
RMAN> connect target sys/target_pwd@target_str
RMAN> connect catalog rman/cat_pwd@cat_str
```

Make Sure You Have the Necessary Backups and Archived Redo Logs

Make sure you have backups all the datafiles in your target database. If you do not have backups of everything, the duplicate operation will fail. The database backup does not have to be a whole database backup: you can use a mix of full and incremental backups of individual datafiles.

Make sure that you have enough backups of all the archived redo logs necessary to recover to the desired time, SCN, or log sequence number.

Allocate Auxiliary Channels

Before issuing the **duplicate** command, allocate at least one auxiliary channel within the same **run** command. The channel type (**disk** or *'sbt_tape'*) must match the media where the backups of the target database are located. If the backups reside on disk, then the more channels you allocate, the faster the duplication will be. For tape backups, limit the number of channels to the number of devices available for the operation.

```
run {
  # to allocate a channel of type 'sbt_tape' issue:
  allocate auxiliary channel ch1 type 'sbt_tape';

  # to allocate three auxiliary channels for disk issue (specifying whatever channel
  # id that you want):
  allocate auxiliary channel aux1 type disk;
  allocate auxiliary channel aux2 type disk;
  allocate auxiliary channel aux3 type disk;
  . . .
}
```

Creating a Duplicate Database on a Local or Remote Host

When you create your duplicate database, you have the following options:

- [Duplicating a Database on a Remote Host with the Same Directory Structure](#)
- [Duplicating a Database on a Remote Host with a Different Directory Structure](#)
- [Creating a Duplicate Database on the Local Host](#)

Duplicating a Database on a Remote Host with the Same Directory Structure

The simplest case is to duplicate your database to a different host and to use the exact same directory structure. In this case, you do not need to change the `init.ora` file or set new filenames for the duplicate database datafiles.

To create a duplicate database on a different host with the same filesystem:

1. Use an O/S utility to copy your parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure.
2. Use SQL*Plus to start the duplicate instance without mounting it. For example, enter:

```
SQL> startup nomount pfile=initDUPDB.ora;
```

3. Use SQL*Plus to open the target database if it is not already open. For example, enter:

```
SQL> startup pfile=initPRODL.ora;
```

4. The auxiliary instance must be accessible via Net8. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.
5. Use RMAN to connect to the target database, the duplicate database, and (if you use one) the recovery catalog database. In this example, connection is established without a recovery catalog using O/S authentication:

```
% rman target / auxiliary sys/sys_pwd@dupdb
```

In this example, user SCOTT has SYSDBA privileges and a net service name is used for the target:

```
% rman auxiliary scott/tiger@dupdb target sys/sys_pwd@prod
```

In this example, connection is established to three databases, all using net service names:

```
% rman catalog rman/rman@rcat target sys/sys_pwd@prodl auxiliary scott/tiger@dupdb
```

6. Perform the following operations:
 - Allocate at least one auxiliary channel.
 - Specify the **nofilenamecheck** parameter.

For example, enter the following:

```
run {
  allocate auxiliary channel ch1 type 'sbt_tape';
  duplicate target database to dupdb
  nofilenamecheck;
}
```

7. RMAN uses all available archived redo logs and incremental backups to perform incomplete recovery and then opens the database with the RESETLOGS option to create the online redo logs.

Duplicating a Database on a Remote Host with a Different Directory Structure

If you create your duplicate database on a host with a different filesystem, you need to change several `init.ora` file parameters and generate new filenames for the duplicate database datafiles.

Use `LOG_FILE_NAME_CONVERT` or the **logfile** clause to convert the online redo log filenames. Use `DB_FILE_NAME_CONVERT`, the **set newname** command, or the **set auxname** command for the datafile filenames.

See Also: For a table of the various datafile filename conversion options, see [Table 10-2](#) on page 10-5.

To duplicate a database with `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT`:

1. Use an O/S utility to copy your parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure. Make sure to set:
 - All `*_DEST` and `*_PATH` initialization parameters that specify a pathname.
 - `DB_FILE_NAME_CONVERT` so that it captures *all* the target datafiles and converts them appropriately, e.g., from `tbs_*` to `dup_tbs_*`.
 - `LOG_FILE_NAME_CONVERT` so that it captures *all* the online redo logs and converts them appropriately, e.g., `log_*` to `dup_log_*`.
2. Use SQL*Plus to start the duplicate instance without mounting it. For example, enter:

```
SQL> startup nomount pfile=initDUPDB.ora;
```
3. Use SQL*Plus to open the target database if it is not already open. For example, enter:

```
SQL> startup pfile=initPRODL.ora;
```
4. The auxiliary instance must be accessible via Net8. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.

5. Use RMAN to connect to the target database, the duplicate database, and (if you use one) the recovery catalog database. In this example, connection is established without a recovery catalog using O/S authentication:

```
% rman target / auxiliary sys/sys_pwd@dupdb
```

In this example, user SCOTT has SYSDBA privileges and a net service name is used for the target:

```
% rman auxiliary scott/tiger@dupdb target sys/sys_pwd@prod
```

In this example, connection is established to three databases, all using net service names:

```
% rman catalog rman/rman@rcat target sys/sys_pwd@prodl auxiliary scott/tiger@dupdb
```

6. Issue the **duplicate** command. For example, enter the following:

```
run {
  allocate auxiliary channel chl type 'sbt_tape';
  duplicate target database to dupdb;
}
```

7. RMAN uses all available archived redo logs and incremental backups to perform incomplete recovery and then opens the database with the RESETLOGS option to create the online redo logs.

To duplicate a database with DB_FILE_NAME_CONVERT and the logfile clause:

Follow the same procedure for creating a duplicate database using the parameter LOG_FILE_NAME_CONVERT, but make the following substitutions:

- In step 1, do not set the LOG_FILE_NAME_CONVERT parameter.
- In step 6, change the **run** command as follows, specifying the names for the duplicate database redo log members:

```
run {
  allocate auxiliary channel chl type 'sbt_tape';
  duplicate target database to dupdb
  logfile
    '/oracle/dbs/log1.f' size 200K,
    '/oracle/dbs/log2.f' size 200K;
}
```

To duplicate a database using the set newname command:

1. Use an O/S utility to copy your parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure. Set all *_DEST and *_PATH initialization parameters that specify a pathname.

2. Use SQL*Plus to start the duplicate instance without mounting it:

```
SQL> startup nomount pfile=initDUPDB.ora;
```

3. Use SQL*Plus to open the target database if it is not already open:

```
SQL> startup pfile=initPRODL.ora;
```

4. The auxiliary instance must be accessible via Net8. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.

5. Use RMAN to connect to the target database, the duplicate database, and (if you use one) the recovery catalog database. In this example, connection is established without a recovery catalog using O/S authentication:

```
% rman target / auxiliary sys/sys_pwd@dupdb
```

In this example, user SCOTT has SYSDBA privileges and a net service name is used for the target:

```
% rman auxiliary scott/tiger@dupdb target sys/sys_pwd@prod
```

In this example, connection is established to three databases, all using net service names:

```
% rman catalog rman/rman@rcat target sys/sys_pwd@prodl auxiliary scott/tiger@dupdb
```

6. Perform the following operations:
 - Allocate at least one auxiliary channel.
 - Specify the same number of redo log members and groups that are used in your target database.
 - Specify new filenames for the duplicate database datafiles.

For example, enter the following:

```
run {
  # allocate at least one auxiliary channel of type disk or tape
  allocate auxiliary channel dupdbl type 'sbt_tape';
  . . .
  # set new filenames for the datafiles
  set newname for datafile 1 TO '$ORACLE_HOME/dbs/dupdb_data_01.f';
  set newname for datafile 2 TO '$ORACLE_HOME/dbs/dupdb_data_02.f';
  . . .
  # issue the duplicate command
  duplicate target database to dupdb
  # create at least two online redo log groups
  logfile
    group 1 ('$ORACLE_HOME/dbs/dupdb_log_1_1.f',
            '$ORACLE_HOME/dbs/dupdb_log_1_2.f') size 200K,
    group 2 ('$ORACLE_HOME/dbs/dupdb_log_2_1.f',
            '$ORACLE_HOME/dbs/dupdb_log_2_2.f') size 200K;
}
```

7. RMAN uses all available archived redo logs and incremental backups to perform incomplete recovery and then opens the database with the RESETLOGS option to create the online redo logs.

To duplicate a database using the set auxname command:

1. Use an O/S utility to copy your parameter file from its location in the target host directory structure to the same location in the duplicate host directory structure. Set all *_DEST and *_PATH initialization parameters that specify a pathname.
2. Use SQL*Plus to start the duplicate instance without mounting it:


```
SQL> startup nomount pfile=initDUPDB.ora;
```
3. Use SQL*Plus to open the target database if it is not already open:


```
SQL> startup pfile=initPRODL.ora;
```
4. The auxiliary instance must be accessible via Net8. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance. Note that you must connect to the auxiliary instance with SYSDBA privileges, so a password file must exist.
5. Use RMAN to connect to the target database, the duplicate database, and (if you use one) the recovery catalog database. In this example, connection is established without a recovery catalog using O/S authentication:

```
% rman target / auxiliary sys/sys_pwd@dupdb
```

In this example, user SCOTT has SYSDBA privileges and a net service name is used for the target:

```
% rman auxiliary scott/tiger@dupdb target sys/sys_pwd@prod
```

In this example, connection is established to three databases, all using net service names:

```
% rman catalog rman/rman@rcat target sys/sys_pwd@prodl auxiliary scott/tiger@dupdb
```

6. Set the auxiliary names for your datafiles. For example, enter the following:

```
# set auxiliary names for the datafiles
set auxname for datafile 1 to '/oracle/auxfiles/aux_1.f';
set auxname for datafile 2 to '/oracle/auxfiles/aux_2.f';
...
set auxname for datafile n to '/oracle/auxfiles/aux_n.f';
```

7. Perform the following operations:

- Allocate at least one auxiliary channel.
- Specify the same number of redo log members and groups that are used in your target database.

```
run {
  # allocate at least one auxiliary channel of type disk or tape
  allocate auxiliary channel dupdbl type 'sbt_tape';
  . . .
  # issue the duplicate command
  duplicate target database to dupdb
  . . .
  # create at least two online redo log groups
  logfile
    group 1 ('$ORACLE_HOME/dbs/dupdb_log_1_1.f',
            '$ORACLE_HOME/dbs/dupdb_log_1_2.f') size 200K,
    group 2 ('$ORACLE_HOME/dbs/dupdb_log_2_1.f',
            '$ORACLE_HOME/dbs/dupdb_log_2_2.f') size 200K;
}
```

8. RMAN uses all available archived redo logs and incremental backups to perform incomplete recovery and then opens the database with the RESETLOGS option to create the online redo logs.

9. Un-specify the auxiliary names for your datafiles so that they will not be overwritten by mistake. For example, enter the following:

```
# un-specify auxiliary names for the datafiles
set auxname for datafile 1 to null;
set auxname for datafile 2 to null;
...
set auxname for datafile n to null;
```

Creating a Duplicate Database on the Local Host

When creating a duplicate database on the same host as your target database, follow the same procedure as for duplicating to a remote host with a different directory structure ("[Duplicating a Database on a Remote Host with a Different Directory Structure](#)" on page 10-12).

Note that you can duplicate your database to the same `$ORACLE_HOME` as your target, but you must convert the filenames using the same methods used for conversion on a separate host.

WARNING: Do not use the `nofilenamecheck` option when duplicating to the same `$ORACLE_HOME` as your primary database. If you do, you may overwrite your target database files or cause the duplicate command to fail with an error.

Duplication Scenarios

Following are some useful scenarios for creating a duplicate database:

- [Setting New Filenames Manually](#)
- [Resynchronizing the Duplicate Database with the Target Database](#)
- [Creating a Non-Current Duplicate Database](#)

Setting New Filenames Manually

This example assumes the following:

- You are using recovery catalog database RCAT.
- Your target database is on HOST1 and contains nine datafiles.

- You want to duplicate your target database to database DUPDB on remote host HOST2.
- HOST1 and HOST2 use different filesystems.
- You want to store all the datafiles in HOST2 in the `/oracle/dbs` sub-directory and use the `tbs_*` prefix for each datafile.
- You have used an O/S utility to copy your parameter file from HOST1 to an appropriate location in HOST2.
- You have reset all `*_DEST` and `*_PATH` initialization parameters that specify a pathname.
- You have disk copies or backup sets stored on disk for all the datafiles in the target database, and have manually moved them to HOST2 using an O/S utility.
- You want to have two online redo logs groups, each with two members of size 200K.

```

connect target;
connect catalog rman/rman@rcat;
connect auxiliary sys/change_on_install@dupdb;
run {
  allocate auxiliary channel dupdb1 type disk;
  allocate auxiliary channel dupdb2 type disk;
  allocate auxiliary channel dupdb3 type disk;
  allocate auxiliary channel dupdb4 type disk;
  set newname for datafile 1 TO '$ORACLE_HOME/dbs/tbs_01.f';
  set newname for datafile 2 TO '$ORACLE_HOME/dbs/tbs_02.f';
  set newname for datafile 3 TO '$ORACLE_HOME/dbs/tbs_03.f';
  set newname for datafile 4 TO '$ORACLE_HOME/dbs/tbs_04.f';
  set newname for datafile 5 TO '$ORACLE_HOME/dbs/tbs_05.f';
  set newname for datafile 6 TO '$ORACLE_HOME/dbs/tbs_06.f';
  set newname for datafile 7 TO '$ORACLE_HOME/dbs/tbs_07.f';
  set newname for datafile 8 TO '$ORACLE_HOME/dbs/tbs_08.f';
  set newname for datafile 9 TO '$ORACLE_HOME/dbs/tbs_09.f';
  duplicate target database to dupdb logfile
    group 1 ('$ORACLE_HOME/dbs/log_1_1.f',
             '$ORACLE_HOME/dbs/log_1_2.f') size 200K reuse,
    group 2 ('$ORACLE_HOME/dbs/log_2_1.f',
             '$ORACLE_HOME/dbs/log_2_2.f') size 200K reuse;
}

```

Resynchronizing the Duplicate Database with the Target Database

This example makes the same assumptions as in ["Setting New Filenames Manually"](#) on page 10-17. Additionally, it assumes that you want to update your duplicate database daily so that it stays current with the target database.

```
# start RMAN and then connect to the databases
connect target /
connect catalog rman/rman@rcat
connect auxiliary sys/change_on_install@dupdb

# set auxiliary names for the datafiles only once
set auxname for datafile 1 TO '$ORACLE_HOME/dbs/tbs_01.f';
set auxname for datafile 2 TO '$ORACLE_HOME/dbs/tbs_02.f';
set auxname for datafile 3 TO '$ORACLE_HOME/dbs/tbs_03.f';
set auxname for datafile 4 TO '$ORACLE_HOME/dbs/tbs_04.f';
set auxname for datafile 5 TO '$ORACLE_HOME/dbs/tbs_05.f';
set auxname for datafile 6 TO '$ORACLE_HOME/dbs/tbs_06.f';
set auxname for datafile 7 TO '$ORACLE_HOME/dbs/tbs_07.f';
set auxname for datafile 8 TO '$ORACLE_HOME/dbs/tbs_08.f';
set auxname for datafile 9 TO '$ORACLE_HOME/dbs/tbs_09.f';

# Create the duplicate database. Issue the same command daily
# to re-create the database, thereby keeping the duplicate
# in sync with the target.
run {
  # allocate auxiliary channels
  allocate auxiliary channel dupdb1 type disk;
  allocate auxiliary channel dupdb2 type disk;
  allocate auxiliary channel dupdb3 type disk;
  allocate auxiliary channel dupdb4 type disk;
  duplicate target database to dupdb
  logfile
    group 1 ('$ORACLE_HOME/dbs/log_1_1.f',
            '$ORACLE_HOME/dbs/log_1_2.f') size 200K reuse,
    group 2 ('$ORACLE_HOME/dbs/log_2_1.f',
            '$ORACLE_HOME/dbs/log_2_2.f') size 200K reuse;
}
```

Creating a Non-Current Duplicate Database

This example assumes the following:

- Your target database PROD1 and duplicate database DUPDB are on different hosts but have the exact same file structure.
- You wish to name the duplicate database files exactly like the target database files.

- You are not using a recovery catalog.
- You want to recover the duplicate database to one week ago.

```
connect target sys/change_on_install@prod1
connect auxiliary sys/sysdba@dupdb
run {
  set until time 'sysdate-7';
  allocate auxiliary channel dupdb1 type 'sbt_tape';
  allocate auxiliary channel dupdb2 type 'sbt_tape';
  duplicate target database to dupdb
  nofilenamecheck;
}
```

Part III

Recovery Manager Reference

Recovery Manager Command Syntax

This chapter describes, in alphabetical order, Recovery Manager commands and sub-clauses.

Conventions Used in this Reference

This section explains the conventions used in this book including:

- [Text](#)
- [Syntax Diagrams and Notation](#)
- [Code Examples](#)

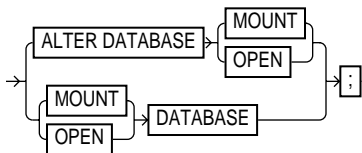
Text

The text in this reference adheres to the following conventions:

UPPERCASE	Uppercase text calls attention to SQL commands and keywords, filenames, column headings in tables and views, and initialization parameters.
bold	Bold text calls attention to Recovery Manager keywords.
<i>italics</i>	Italicized text calls attention to definitions of terms, the names for Recovery Manager parameters and options that are not keywords (e.g., <i>integer</i>), and sample values for Recovery Manager parameters (e.g., datafile <i>tbs_01.f</i>).

Syntax Diagrams and Notation

Syntax Diagrams This reference uses syntax diagrams to show Recovery Manager commands. These syntax diagrams use lines and arrows to show syntactic structure, as shown here:



This section describes the components of syntax diagrams and gives examples of how to write Recovery Manager commands. Syntax diagrams are made up of these items:

Keywords Keywords have special meanings in Recovery Manager syntax. In the syntax diagrams, keywords appear in square boxes and an uppercase font. When described in text, RMAN keywords appear in lowercase bold, e.g., **backup database**. You must use keywords in your RMAN statements exactly as they appear in the syntax diagram, except that they can be either uppercase or lowercase.

The RMAN language is free-form. Keywords must be separated by at least one white-space character, but otherwise there are no restrictions. A command may span multiple lines.

Parameters Parameters act as placeholders in syntax diagrams. In the syntax diagrams, they appear in ovals. When described in text, RMAN parameters appear in lowercase italics, e.g., *'filename'*. Parameters are usually names of database objects (*tablespace_name*), Oracle data type names (*date_string*), or sub-clauses (*datafileSpec*). When you see a parameter in a syntax diagram, substitute an object or expression of the appropriate type in your RMAN statement. For example, to write a **duplicate target database to** command, use the name of the duplicate database you want to create, such as *dupdb*, in place of the *database_name* parameter in the syntax diagram.

Some parameter values are enclosed in required or optional quotes. The syntax diagrams show single quotes, though in all cases double quotes are also legal. For example, you specify either *'filename'* or *"filename"*. For the **sql** command, it is recommended that you use double quotes.

This lists shows parameters that appear in the syntax diagrams and provides examples of the values you might substitute for them in your statements:

Parameter	Description	Examples
quoted strings such as <i>'filename'</i> , <i>'tablespace_name'</i> , <i>'channel_name'</i> , <i>'channel_parms'</i>	A string of characters contained in either single or double quotes, e.g., <i>'filename'</i> or <i>"filename"</i> . A quoted string may contain whitespace, punctuation, and RMAN and SQL keywords.	<i>"?/dbs/cf.f"</i> <i>'dev1'</i>
non-quoted strings such as <i>channel_id</i> , <i>tag_name</i> , <i>date_string</i>	A sequence of characters containing no white-space and no punctuation characters and starting with an alphabetic character.	<i>ch1</i>
<i>integer</i>	Any sequence of characters containing only number characters.	<i>67843</i>

Code Examples

This reference contains many examples of RMAN commands. These examples show you how to use elements of RMAN. The following example shows a **backup** command:

```
run {  
    allocate channel ch1 type disk;  
    backup database;  
}
```

Note that examples appear in a different font from the text.

Command Entries

The description of each command or sub-clause contains the following sections:

Syntax	shows the keywords and parameters that make up the statement. Note: Not all keywords and parameters are valid in all circumstances. Be sure to refer to the "Keywords and Parameters" section of each statement to learn about any restrictions on the syntax.
Purpose	describes the basic uses of the statement.
Requirements	lists any requirements and restrictions for proper use of the command.
Keywords and Parameters	describes the purpose of each keyword and parameter. Restrictions and usage notes also appear in this section.
Examples	shows how to use various clauses and options of the statement. Usage notes: Optional sections following the examples provide more information on how and when to use the statement.

Summary of RMAN Commands

The following table provides a functional summary of RMAN commands. Note that all release 8.0 commands still work with the release 8.1 RMAN executable.

Table 11–1 Recovery Manager Commands

Command	Purpose
"allocate" on page 11-9	Establish a <i>channel</i> , which is a connection between RMAN and a database instance.
"allocateForMaint" on page 11-9	Allocate a channel in preparation for issuing maintenance commands such as change .
"alterDatabase" on page 11-9	Mount or open a database.
"archivelogRecordSpecifier" on page 11-17	Specify a range of archived redo logs files for use in backup, restore, and maintenance operations as well as queries to the recovery catalog.
"backup" on page 11-21	Back up a database, tablespace, datafile, or archived redo log file.
"catalog" on page 11-32	Add information about a datafile copy, archived redo log, or control file copy to the recovery catalog and control file. Catalog a datafile copy as a level 0 backup, which enables you to use it as part of an incremental backup strategy. Record the existence of file copies created before RMAN was installed or generated via means other than RMAN.
"change" on page 11-35	Mark a backup piece, image copy, or archived redo log as having the status unavailable or available . Delete a backup piece, image copy, or archived redo log from the operating system and update its recovery catalog record to status deleted . Check whether backup pieces, datafile copies, or archived redo logs are available and, if they are not, mark them as expired .
"cmdLine" on page 11-39	Connect to the target, recovery catalog, or auxiliary database. Specify that you are using RMAN without a recovery catalog. Specify a command file, which is a user-defined file containing RMAN commands. Specify the file in which RMAN records the results of processed commands. Add to rather than overwrite the contents of the command file. Generate debugging output and specify its location.

Table 11–1 Recovery Manager Commands

Command	Purpose
" completedTimeSpec " on page 11-42	A sub-clause that specifies a time range during which the backup or copy completed.
" connect " on page 11-44	Establish a connection between RMAN and a target, auxiliary (duplicated or auxiliary instance used for TSPITR), or recovery catalog database.
" connectStringSpec " on page 11-46	Specify the username, password, and net service name for connecting to a target, recovery catalog, or auxiliary database. The connection is necessary to authenticate the user and identify the database.
" copy " on page 11-48	Create an image copy of a file.
" createCatalog " on page 11-52	Create a schema for the recovery catalog.
" createScript " on page 11-54	Create a stored script and store it in the recovery catalog for future reference.
" crosscheck " on page 11-57	Determine whether backup sets stored on disk or tape still exist.
" datafileSpec " on page 11-60	Specify a datafile by filename or absolute file number.
" debug " on page 11-62	Turn RMAN's debugging feature off and on.
" deleteExpired " on page 11-63	Delete backup sets marked EXPIRED by the crosscheck command and remove references to them from the recovery catalog and control file.
" deleteScript " on page 11-63	Delete a stored script from the recovery catalog.
" deviceSpecifier " on page 11-66	Specify the type of storage for a backup or copy.
" dropCatalog " on page 11-68	Remove the schema from the recovery catalog.
" duplicate " on page 11-69	Use backups of the target database to create a duplicate database that you can use for testing purposes.
" host " on page 11-74	Invoke an O/S command-line sub-shell from within RMAN.
" list " on page 11-76	Produce a detailed report about a specified group of backup sets or copies recorded in the recovery catalog or target control file.
" listObjList " on page 11-84	Specify a database or one or more tablespaces, control files, datafiles, or archived redo logs.
" printScript " on page 11-86	Print a stored script to the RMAN message log file. Specify the log filename with the log argument at the command line (see " connect " on page 11-44).
" recover " on page 11-88	Apply redo logs or incremental backups to a restored backup set or copy in order to update it to a specified time.

Table 11–1 Recovery Manager Commands

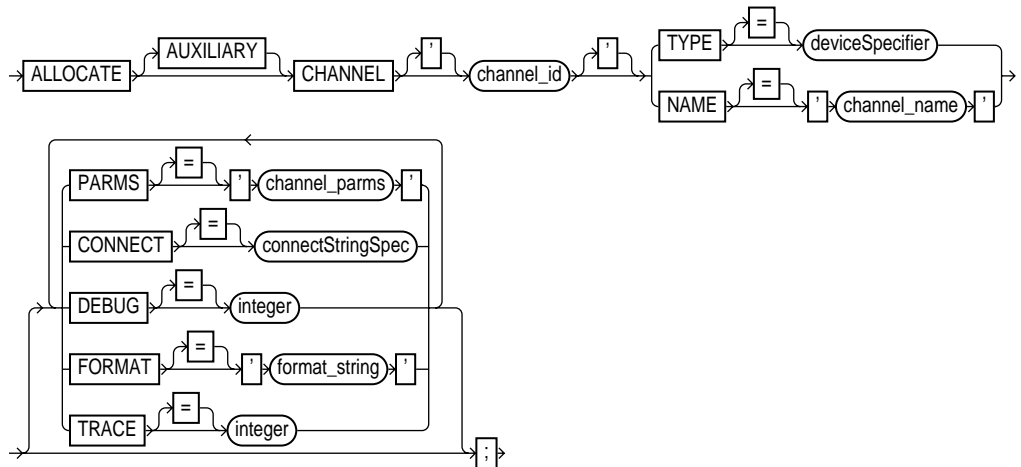
Command	Purpose
"register" on page 11-93	Register the target database in the recovery catalog so that RMAN can access it.
"release" on page 11-95	Release a sequential I/O device while maintaining the connection to the target database instance.
"releaseForMaint" on page 11-96	Release a sequential I/O device specified in an allocate channel command with the for delete or for maintenance option.
"replaceScript" on page 11-97	Replace an existing script stored in the recovery catalog. If the script does not exist, replace script creates it.
"replicate" on page 11-100	Copy the control file to multiple destinations.
"report" on page 11-102	Perform detailed analyses of the content of the recovery catalog.
"reset" on page 11-110	Create a new database incarnation record in the recovery catalog.
"restore" on page 11-112	Restore files from backup sets or from copies on disk to the current location, overwriting the files with the same name.
"resync" on page 11-118	Perform a full <i>resynchronization</i> , which creates a snapshot control file and then compares the recovery catalog to either the current control file of the target database or the snapshot control file and updates it with information that is missing or changed.
"rmanCmd" on page 11-121	Execute <i>stand-alone</i> commands, which are commands you run from the command-line interpreter (CLI), i.e., the RMAN prompt.
"run" on page 11-124	Compile and execute <i>job commands</i> , which are one or more statements executed within the braces of run .
"send" on page 11-127	Send a vendor-specific quoted string to one or more specific channels.
"set" on page 11-129	Specify the auxiliary filenames for target datafiles. This operation is useful when performing TSPITR. Display executed RMAN commands in the message log. Specify a database's db identifier. Set the filename of the snapshot control file.

Table 11–1 Recovery Manager Commands

Command	Purpose
"set_run_option" on page 11-133	Specify new filenames for datafiles. Specify a limit for the number of permissible block corruptions. Override default archived redo log destinations. Specify that backups should be duplexed. Determine which server process corresponds to which channel. Limit the number of buffers that will be read from each input datafile on a specified channel. Limit the number of input files that a backup operation can have open at any given time for a specified channel. Limit the size of the backup pieces for a specified channel.
"shutdown" on page 11-137	Shut down the target database without exiting RMAN. This command is equivalent to the SQL*Plus SHUTDOWN command.
"sql" on page 11-140	Execute a SQL statement from within Recovery Manager.
"startup" on page 11-142	Start up the database from within the RMAN environment. This command is equivalent to the SQL*Plus STARTUP command.
"switch" on page 11-144	Specify that a datafile copy is now the <i>current datafile</i> , i.e., the datafile pointed to by the control file.
"upgradeCatalog" on page 11-148	Upgrade the recovery catalog schema from an older version to the version required by the RMAN executable.
"validate" on page 11-150	Examine a backup set and report whether its data is intact. RMAN scans all of the backup pieces in the specified backup sets and looks at the checksums to verify that the contents can be successfully restored if necessary.

allocate

Syntax



Purpose

To establish a *channel*, which is a connection between RMAN and a database instance. Each connection initiates an Oracle server session on the target instance: this server session performs the work of backing up, restoring, and recovering backup sets and copies.

Each channel operates on one backup set at a time (for **backup**, **restore**, or **recover**) or one image copy at a time (for **copy**). RMAN automatically releases the channel at the end of the job.

Control the degree of parallelism within a job by the number of allocated channels. You can allocate multiple channels simultaneously, thus allowing a single job to read or write multiple backup sets or copies in parallel. If you establish multiple connections, each connection operates on a separate backup set or file copy.

Whether **allocate channel** causes operating system resources to be allocated depends on your operating system. On some platforms, operating system resources are allocated at the time the command is issued. On other platforms, operating system resources are not allocated until you open a file for reading or writing.

Note: When you specify **type disk**, no O/S resources are allocated other than for the creation of the server session.

Requirements

- Execute **allocate** only within the braces of a **run** command.
- Allocate a channel before executing a **backup**, **duplicate** (see "[duplicate](#)" on page 11-69), **copy**, **restore**, **recover**, or **validate** command.
- When duplexing backups, execute the **set duplex** command (see "[set_run_option](#)" on page 11-133) before allocating a channel for a **backup** command.

Keywords and Parameters

auxiliary	<p>specifies a connection between RMAN and an auxiliary database instance. An auxiliary instance is used when executing the duplicate command or performing TSPITR. An auxiliary database can reside in the same host as its parent or in a different host. When specifying this option, the auxiliary database must be mounted but not open.</p> <p>See Also: To learn how to duplicate a database, see "duplicate" on page 11-69. To learn how to connect to a duplicate database, see "connect" on page 11-44.</p>
channel <i>channel_id</i>	<p>specifies a connection between RMAN and the target database instance. Each connection initiates an Oracle server session on the database instance: this server session performs the work of backing up, restoring, and recovering backups and copies.</p> <p>Specify a channel id, which is the name of the channel, after the channel keyword. Oracle uses the identifier with the release channel command and to report I/O errors.</p>
type <i>deviceSpecifier</i>	<p>specifies the type of storage device (see "deviceSpecifier" on page 11-66).</p> <p>Note: If you do not specify the type parameter, then you must specify the name parameter to identify a particular sequential I/O device. Query the V\$BACKUP_DEVICE view for information about available device types and names.</p>
name <i>'channel_name'</i>	<p>specifies the name of a sequential I/O device. If you do not specify a device name, then the system uses any available device of the specified type. Do not use this parameter in conjunction with the type parameter.</p> <p>Currently, no platform supports the name parameter.</p>

parms <i>'channel_parms'</i>	<p>specifies parameters for the device to allocate. Do not use this port-specific string if you have specified type disk.</p> <p>If you use parms in conjunction with type 'sbt_tape', you can specify environment variables. Following are models for acceptable syntax:</p> <pre>PARMS="(ENV=(var1=value1,var2=value2,var3=value3 . . .))" PARMS="BLKSIZE=integer"</pre> <p>For example, you can specify:</p> <pre>PARMS="BLKSIZE=16384,ENV=(NSR_SERVER=tape_server,NSR_CLIENT=oracleclnt, NSR_GROUP=oracle_tapes)"</pre> <p>The maximum length of the quoted string is 1000 bytes.</p>
connect <i>connectStringSpec</i>	<p>specifies a connect string to the database instance where RMAN should conduct the backup or restore operations (see "connectStringSpec" on page 11-46). Use this parameter when you want to spread the work of backup or restore operations across different instances in an OPS configuration.</p> <p>If you do not specify this parameter, and you did not specify the auxiliary option, then RMAN conducts all operations on the target database instance specified by the command-line parameter (see "cmdLine" on page 11-39) or the instance connected to when you issued the connect command. Typically, you should not use the connect parameter in conjunction with the auxiliary option.</p>
debug <i>integer</i>	<p>specifies that Oracle should log debugging information about copy, backup, and restore operations performed on this channel by the target or auxiliary database to a trace file. Use this parameter only at the direction of Oracle Support: they will tell you which integer to use.</p>
format <i>'format_string'</i>	<p>specifies the format to use for the names of backup pieces that are created on this channel. If you do not specify a format, RMAN uses %U by default, which guarantees a unique identifier. For available format parameters, see the backup command.</p> <p>This parameter is useful if you allocate multiple disk channels and want each channel to write to a different filesystem. If you specify the format parameter in the backup command, it overrides the format parameter specified in allocate channel.</p>
trace <i>integer</i>	<p>specifies an integer whose meaning is determined by the media management software. Typically, this parameter controls how much diagnostic trace data is produced by the media manager.</p>

Examples

Allocating a Single Channel for a Backup This command allocates a tape channel for a whole database backup:

```
run {
    allocate channel dev1 type 'sbt_tape';
    backup database;
}
```

Spreading a Backup Set Across Multiple Channels When backing up to disk, you can spread your backup across several disk drives. Allocate one `type disk` channel per disk drive and specify the format string so that the filenames are on different disks:

```
run{
    allocate channel disk1 type disk format '/disk1/%d_backups/%U';
    allocate channel disk2 type disk format '/disk2/%d_backups/%U';
    allocate channel disk3 type disk format '/disk3/%d_backups/%U';
    backup database;
}
```

Duplexing a Backup Set When duplexing backup sets, specify the `set duplex` command (see ["set_run_option"](#) on page 11-133) before allocating a channel. The following example generates four identical backups of datafile 1:

```
run {
    set duplex = 4;
    allocate channel dev1 type 'sbt_tape';
    backup datafile 1;
}
```

Allocating an Auxiliary Channel When creating a duplicate database (see ["duplicate"](#) on page 11-69), allocate a channel using the `auxiliary` option:

```
run {
    allocate auxiliary channel c1 type disk;
    allocate auxiliary channel c2 type disk;
    duplicate target database to ndbnewh
        logfile
            '/oracle/dbs/log_1.f' size 200K,
            '/oracle/dbs/log_2.f' size 200K
    skip readonly
    nofilenamecheck;
}
```

Related Topics

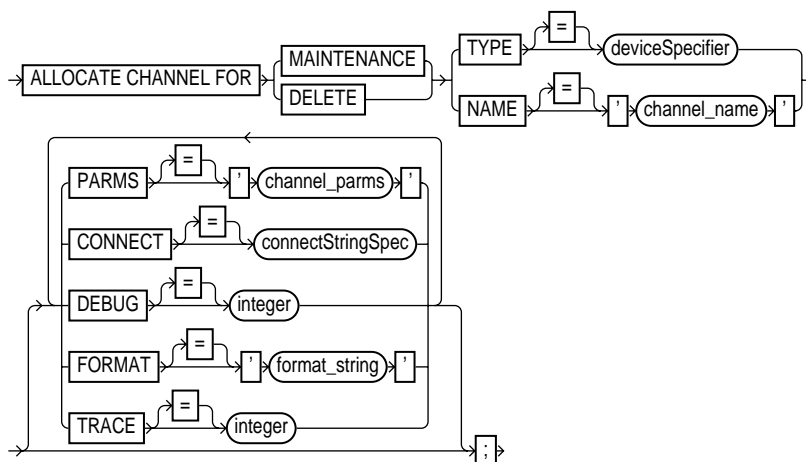
["allocateForMaint"](#) on page 11-13

["duplicate"](#) on page 11-69

["cmdLine"](#) on page 11-39

allocateForMaint

Syntax



Purpose

To allocate a channel in preparation for issuing a **change** or **crosscheck** command. The **maintenance** and **delete** options, which are completely synonymous, specify the device type appropriate for the file whose status you are changing.

Requirements

- Execute this command only at the RMAN prompt.
- Issue an **allocate channel for delete** or **allocate channel for maintenance** command before issuing the following:
 - **change backupset ... delete**
 - **change backuppiece ... delete**
 - **change backupset ... crosscheck**
 - **change backuppiece ... crosscheck**
 - **crosscheck**

- Do not specify a channel id.
- You must release an allocated channel before you can allocate a new one, i.e., you cannot allocate multiple maintenance channels.

Keywords and Parameters

See ["allocate"](#) on page 11-9.

Examples

Deleting a Backup Piece This example deletes a backup piece from the media management catalog:

```
allocate channel for maintenance type 'sbt_tape';  
change backuppiece '/oracle/dbs/01aj3q5012' delete;  
release channel;
```

Marking a File Unavailable This example crosschecks the backup set with primary key 828:

```
allocate channel for maintenance type disk;  
change backupset 828 crosscheck;  
release channel;
```

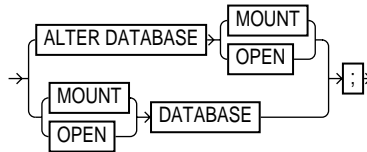
Related Topics

["allocate"](#) on page 11-9

["change"](#) on page 11-35

alterDatabase

Syntax



Purpose

To mount or open a database.

Requirements

- Execute this command either within the braces of a **run** command or at the RMAN prompt.
- Execute the command only if an Oracle instance is already started.

Keywords and Parameters

alter database	allows you to either mount or open the database.
mount	mounts the database without opening it.
open	opens the database.
mount database	mounts the database without opening it. This option is equivalent to the SQL statement ALTER DATABASE MOUNT. See Also: For ALTER DATABASE syntax, see the <i>Oracle8i SQL Reference</i> .
open database	mounts and opens database. This option is equivalent to the SQL statement ALTER DATABASE OPEN. See Also: For ALTER DATABASE syntax, see the <i>Oracle8i SQL Reference</i> .

Examples

Opening the Database after a Backup This example mounts the database, takes a whole database backup, then opens the database. At the RMAN prompt enter:

```
startup mount;
run {
    allocate channel ch1 type disk;
    backup database;
    # now that the backup is complete, open the database.
    alter database open;
}
```

Mounting the Database after Restoring the Control File To restore the control file to its default location enter the following:

```
startup nomount;
run {
    allocate channel ch1 type 'sbt_tape';
    restore controlfile;
}
# mount the database with the restored control file.
alter database mount;
```

Related Topics

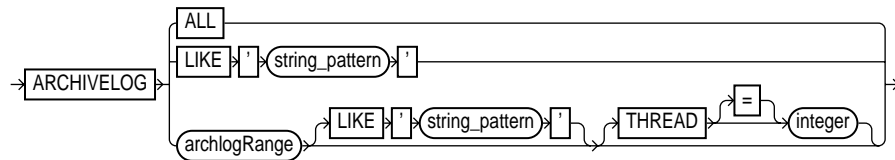
["sql" on page 11-140](#)

["startup" on page 11-142](#)

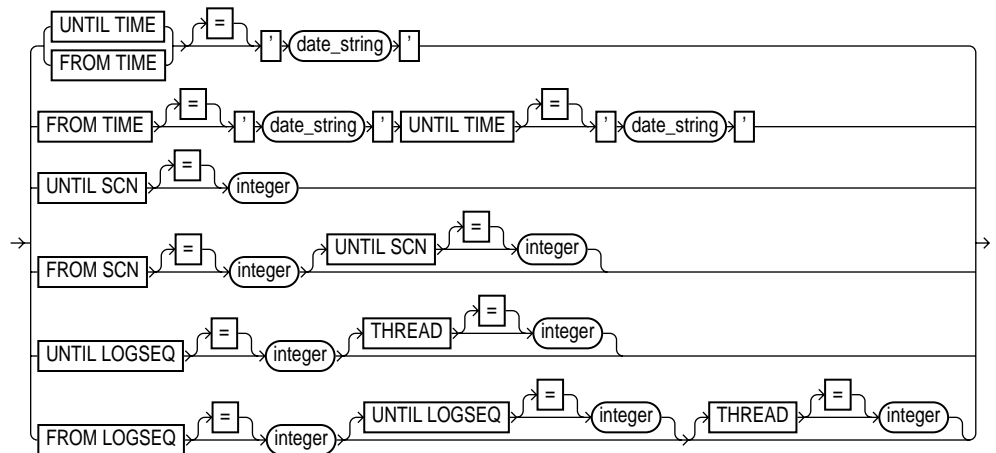
["shutdown" on page 11-137](#)

archivelogRecordSpecifier

Syntax



archlogRange::=



Purpose

A sub-clause used to specify a range of archived redo logs files for use in backup, restore, and maintenance operations.

Requirements

Use this clause only with the following commands:

- **backup**
- **change**

- **crosscheck**
- **deleteExpired**
- **list**
- **restore**

Specifying a range of archived redo logs does not guarantee that RMAN includes all redo data in the range: for example, the last available archived log file may end before the end of the range, or an archived log file in the range may be missing. RMAN includes the archived redo logs it finds and does not issue a warning for missing files.

Keywords and Parameters

Query the V\$ARCHIVED_LOG data dictionary view to determine the timestamps, SCNs, and log sequence numbers for an archived log. For information on using the NLS_LANG and NLS_DATE_FORMAT environment variables to specify the format for the time, see the *Oracle8i Reference*.

all	specifies exactly one copy of each distinct log sequence number. For example, if you execute backup archivelog all and you archive your logs to multiple destinations, RMAN backs up <i>one</i> copy of each log sequence number—not each archived copy of each log sequence number.
like 'string_pattern'	specifies a pathname for archived redo log files. Use this parameter when operating in OPS mode to specify which file system RMAN should access. See Also: For information about the Oracle Parallel Server configuration, see <i>Oracle8i Parallel Server Concepts and Administration</i> .
until time 'date_string'	specifies the end date for a sequence of archived redo log files. The time specified in the string must be formatted according to the NLS date format specification currently in effect. If you do not specify the from time parameter, the beginning time for the sequence will be the earliest available archived redo log. Query the V\$ARCHIVED_LOG data dictionary view to determine the timestamps for the first and last entries in a log. See Also: For information on using the NLS_LANG and NLS_DATE_FORMAT environment variables to specify the format for the time, see the <i>Oracle8i Reference</i> .
from time 'date_string'	specifies the beginning date for a sequence of archived redo log files. If you do not specify the until time parameter, RMAN will include all available log files beginning with the date specified in the from time parameter. Use the V\$ARCHIVED_LOG data dictionary view to determine the timestamps for the first and last entries in a log file. See Also: For information on using the NLS_LANG and NLS_DATE_FORMAT environment variables to specify the format for the time, see the <i>Oracle8i Reference</i> .

until SCN <i>integer</i>	specifies the ending SCN for a sequence of archived redo log files. If you do not specify the from SCN parameter, RMAN will use the lowest available SCN to begin the sequence.
from SCN <i>integer</i>	specifies the beginning SCN for a sequence of archived redo log files. If you do not specify the until SCN parameter, RMAN will include all available log files beginning with SCN specified in the from SCN parameter.
until logseq <i>integer</i>	specifies the terminating log sequence number for a sequence of archived redo log files. If you do not specify the from logseq parameter, RMAN uses the lowest available log sequence number to begin the sequence.
from logseq <i>integer</i>	specifies the beginning log sequence number for a sequence of archived redo log files. If you do not specify the until logseq parameter, RMAN will include all available log files beginning with log sequence number specified in the from logseq parameter.
thread <i>integer</i>	specifies the thread containing the archived redo log files you wish to include. You need only specify this parameter when running your database in an OPS configuration. Use the V\$ARCHIVED_LOG data dictionary view to determine the thread number for an archived redo log record.

Specifying Records by Time This example deletes all archived redo logs older than two weeks:

```
change archivelog until time 'SYSDATE-14' delete;
```

Specifying Records by SCN This example restores backup archived redo log files from tape that fall within a range of SCNs:

```
run {
  allocate channel dev1 type 'sbt_tape';
  restore archivelog
    from SCN 500 until SCN 700;
  release channel dev1;
}
```

Specifying Records by Log Sequence Number This example backs up all archived logs from sequence # 288 to sequence # 301 on thread 1 and deletes the archived logs after the backup is complete. If the backup fails, the logs are not deleted.

```
run {
  allocate channel dev1 type 'sbt_tape';
  backup archivelog
    from logseq 288 until logseq 301 thread 1
    # delete original archived redo logs after backup completes
    delete input;
}
```

Related Topics

["backup"](#) on page 11-21

["change"](#) on page 11-35

["crosscheck"](#) on page 11-57

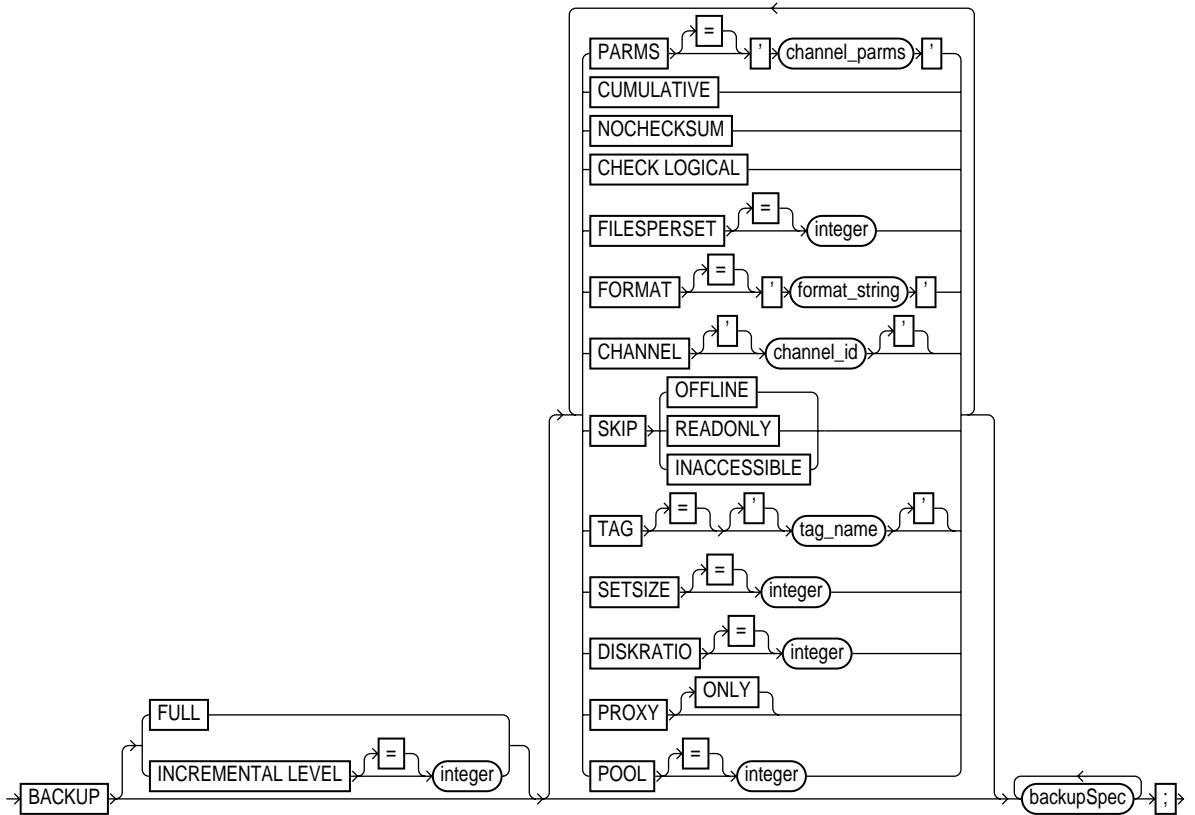
["deleteExpired"](#) on page 11-63

["list"](#) on page 11-76

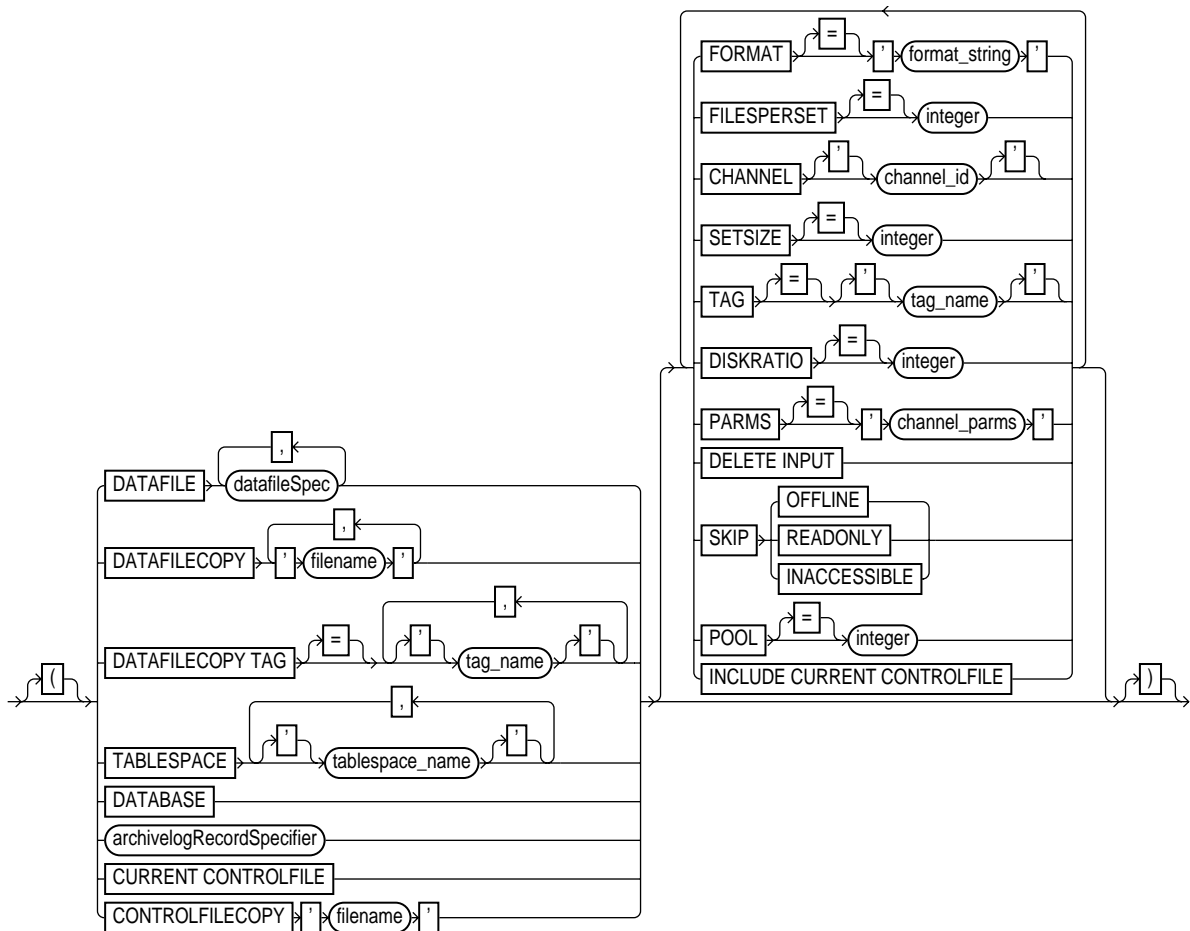
["restore"](#) on page 11-112

backup

Syntax



backupSpec::=



Purpose

To back up a database, tablespace, datafile, control file, or archived redo log file. When performing a backup, specify the files that you want to back up. RMAN puts the input files into a backup set, which is an RMAN-specific logical structure. Each backup set contains at least one backup piece. You can also use the **backup** command to generate a proxy copy, which is a backup created by a media manager.

You control the number of backup sets that Oracle produces as well as the number of input files that RMAN places into a single backup set. Any I/O errors when reading files or writing backup pieces cause Oracle to abort the jobs.

See Also: To learn how to back up files, see [Chapter 8, "Making Backups and Copies with Recovery Manager"](#).

Requirements

When using the **backup** command you must:

- Mount or open the target database. RMAN allows you to make an *inconsistent backup* if the database is in ARCHIVELOG mode, but you must apply redo logs to make the backups consistent for use in restore operations.
- Use a current control file.
- Execute the **backup** command within the braces of a **run** command.
- Allocate a channel for each execution of the **backup** command.
- Give each backup piece a unique name.
- Back up onto valid media. If you specify **type disk**, then you must back up to random-access disks. You can make a backup on any device that can store an Oracle datafile: in other words, if the statement `CREATE TABLESPACE tablespace_name DATAFILE 'filename'` works, then '*filename*' is a valid backup path name. If you specify **type sbt_tape**, then you can back up to any media supported by the media management software.

You *cannot* do the following:

- Make an open backup in NOARCHIVELOG mode.
- Stripe a single backup across multiple channels.
- Stripe a single input file across multiple backups.
- Combine archived redo log files and datafiles into a single backup.
- Execute the **set duplex** command (see "[set_run_option](#)" on page 11-133) *after* you **allocate** a channel for a **backup** command. The **set duplex** command must precede all allocated channels or you will receive an error.
- Specify the number of backup pieces that should go in a backup set.
- Create a backup set containing more than 100 backup pieces.
- Automate the creation of unique tag names for each backup. To create unique tag names every time, write a backup script and then edit it before execution using an O/S utility.

Keywords and Parameters

full	<p>copies all blocks into the backup set, skipping only datafile blocks that have never been used. RMAN makes full backups by default if neither full nor incremental is specified. The server session does not skip blocks when backing up archived redo logs or control files.</p> <p>A full backup has no effect on subsequent incremental backups, so it is not considered a part of the incremental backup strategy.</p>
incremental level <i>n</i> <i>integer</i>	<p>copies only those data blocks that have changed since the last incremental <i>n</i> backup, where <i>n</i> is any integer from 1 to 4. For example, in a level 2 backup RMAN backs up all blocks used since the most recent level2, level 1, or level 0 backup.</p> <p>This type of incremental backup is also called a <i>differential backup</i> to distinguish it from a cumulative backup. An incremental backup at level 0 is identical in content to a full backup, but unlike a full backup the level 0 backup is considered a part of the incremental strategy.</p> <p>Oracle performs checks when attempting to create an incremental backup at a level greater than 0. These checks ensure that the incremental backup will be usable by a subsequent recover command. Among the checks performed are:</p> <ul style="list-style-type: none">■ A level 0 backup set must exist, or level 0 datafile copies must exist for each datafile in the backup command. These backup sets must not be marked unavailable (see "change" on page 11-35).■ Sufficient incremental backups taken since the level 0 must exist and be available such that the incremental backup to be created is usable. <p>If you specify incremental, then in the <i>backupSpec</i> you must set one of the following parameters: datafile, datafilecopy, tablespace, or database. RMAN does not support incremental backups of control files, archived redo logs, or backup sets.</p>
parms ' <i>channel_parms</i> '	<p>specifies a quoted string containing O/S-specific information. RMAN passes the string to the OSD layer each time a backup piece is created.</p>
cumulative	<p>copies the data blocks used since the most recent backup at level <i>n</i>-1 or lower. For example, in a cumulative level 2 backup RMAN backs up all blocks used since the most recent level 1 or level 0 backup.</p>
nochecksum	<p>suppresses block checksums. A <i>checksum</i> is a number that is computed from the contents of a data block. If the DB_BLOCK_CHECKSUM initialization parameter is TRUE, Oracle computes a checksum for each block and stores it in the block before writing the block to disk. When Oracle reads the block from disk later, it makes sure that the block generates the same checksum. If it does not, then the block is damaged.</p> <p>Unless you specify the nochecksum option, Oracle computes a checksum for each block and stores it in the backup. The checksum is verified when restoring from the backup and also written to the datafile when restored. If the database is already maintaining block checksums, then this flag has no effect. The checksum is always verified and stored in the backup in this case.</p> <p>See Also: For more information about the DB_BLOCK_CHECKSUM initialization parameter, see the <i>Oracle8i Reference</i>.</p>

filesperset *integer* specifies the maximum number of input files to place in one backup set.

RMAN divides the total number of files requiring backups by the number of allocated channels to calculate the number of files to place in each backup set. When you specify the **filesperset** parameter, RMAN compares the **filesperset** value to this calculated value and takes the lowest of the two, thereby ensuring that all channels are used. If the number of files specified or implied by the combined *backupSpec* clauses is greater than **filesperset**, e.g., 8 files need backing up when **filesperset** = 4, RMAN creates multiple backup sets to maintain the correct ratio of files per backup set.

If you do not specify **filesperset**, then RMAN compares the calculated value (number of files / allocated channels) to the default value of 64 and takes the lowest of the two, again ensuring that all channels are used. The default value of 64 is high for most applications: specify a lower value or use the **setsize** parameter to limit the size of a backup set.

RMAN always attempts to create enough backup sets so that all allocated channels have work to do. An exception to the rule occurs when there are more channels than files to back up. For example, if RMAN backs up one datafile when three channels are allocated and **filesperset** = 1, then two channels are necessarily idle.

Note: The **setsize** parameter is often easier to use than **filesperset** when you make archive log backups and for datafile backups when the datafiles are striped or they reside on separate disk spindles. If both parameters are specified, both take effect. RMAN attempts to size the backup sets according to the **setsize** parameter, treating **filesperset** as an upper limit.

format
'*format_string*'

specifies the filename to use for the backup piece. Any name that is legal as a sequential filename on the platform is allowed, provided that each backup piece has a unique name. If backing up to disk, then any legal disk filename is allowed, provided it is unique. If you do not specify the **format** parameter, RMAN stores the backup pieces in a port-specific directory (\$ORACLE_HOME/dbs on UNIX).

Specify the **format** parameter in any of these places:

- The *backupSpec* clause
- The **backup** command
- The **allocate channel** command

If specified in more than one of these places, RMAN searches for the **format** parameter in the order shown above.

The following substitution variables are available in format strings to aid in generating unique filenames:

- | | |
|-----------|--|
| %c | specifies the copy number of the backup piece within a set of duplexed backup pieces. If you did not issue the set duplex command, then this variable will be 1 for regular backup sets and 0 for proxy copies. If you issued set duplex , the variable identifies the copy number: 1, 2, 3, or 4. |
| %p | specifies the backup piece number within the backup set. This value starts at 1 for each backup set and is incremented by 1 as each backup piece is created. |

%s	specifies the backup set number. This number is a counter in the control file that is incremented for each backup set. The counter value starts at 1 and is unique for the lifetime of the control file. If you restore a backup control file, then duplicate values can result. Also, CREATE CONTROLFILE initializes the counter back to 1.
%d	specifies the database name.
%n	specifies the database name, padded on the right with 'x' characters to a total length of 8 characters. For example, if PROD1 is the database name, then PROD1xxx is the padded database name.
%t	specifies the backup set timestamp, which is a 4-byte value derived as the number of seconds elapsed since a fixed reference time. The combination of %s and %t can be used to form a unique name for the backup set.
%u	specifies an 8-character name constituted by compressed representations of the backup set number and the time the backup set was created.
%U	specifies a convenient shorthand for %u_%p_%c that guarantees uniqueness in generated backup filenames. If you do not specify a format, RMAN uses %U by default.
channel <i>channel_id</i>	<p>specifies the name of a channel to use when creating the backup sets. Use any name that is meaningful, e.g., <i>ch1</i> or <i>dev1</i>. Oracle uses the channel id with the release channel command and to report I/O errors. If you do not specify this parameter, then RMAN dynamically assigns the backup sets to any available channels during job execution.</p> <p>Note: You can also specify this parameter in the <i>backupSpec</i> clause.</p>
skip	<p>excludes datafiles or archived redo logs from the backup set.</p> <p>Note: You can also specify this option in the <i>backupSpec</i> clause.</p>
offline	specifies that offline datafiles should be excluded from the backup set.
readonly	specifies that read-only datafiles should be excluded from the backup set.
inaccessible	<p>specifies that datafiles or archived redo logs that cannot be read due to I/O errors should be excluded from the backup set.</p> <p>Note that a datafile is only considered inaccessible if it cannot be read. Some offline datafiles can still be read because they still exist on disk. Others have been deleted or moved and so cannot be read, making them inaccessible.</p>

tag <i>tag_name</i>	<p>creates a user-specified tag for the backup set. Typically, a tag is a meaningful name such as <i>monday_evening_backup</i> or <i>weekly_full_backup</i>. Tags must be 30 characters or less. Note that tags are reusable, so that backup set 100 can have the tag <i>monday_evening_backup</i> one week while backup set 105 has the same tag the next week.</p> <p>You can also specify the tag at the <i>backupSpec</i> level. If you specify the tag at:</p> <ul style="list-style-type: none">■ The command level, then all backup sets created by this command are given this tag.■ The <i>backupSpec</i> level, then backup sets created as a result of different backup specifications can have different tags.■ Both levels, then the tag in the <i>backupSpec</i> takes precedence. <p>Note: You cannot automatically assign a different tag name to each backup. The easiest way to give each backup a new tag is to write a backup script and then edit it with an O/S utility before each execution.</p>
setsize <i>integer</i>	<p>specifies a maximum size for a backup set in units of 1K (1024 bytes). Thus, to limit a backup set to 3Mb, specify setsize = 3000. RMAN attempts to limit all backup sets to this size, which is useful in media manager configurations when you want each backup set no larger than one tape.</p> <p>The setsize parameter is easier to use than filesperset when you make archived redo log backups. You should configure your backup sets so that they fit on one tape volume rather than span multiple tape volumes. Otherwise, if one tape of a multi-volume backup set fails, then you lose the data on all the tapes rather than just one.</p> <p>Note: If both setsize and filesperset are specified, both take effect. RMAN attempts to set the size of the backup sets according to the setsize parameter, treating filesperset as an upper limit.</p>
diskratio <i>integer</i>	<p>directs RMAN to assign datafiles (and only datafiles) to backup sets spread across the specified number of drives. For example, assume that your system uses 10 disks, the disks supply data at 10 bytes/second, and the tape drive requires 50 bytes/second to keep streaming. In this case, set diskratio equal to 5 to spread the backup load across 5 disks.</p> <p>The diskratio parameter is also easier for datafile backups when your datafiles are striped or reside on separate disk spindles and you either:</p> <ul style="list-style-type: none">■ Use a high-bandwidth tape drive that requires several datafiles to be multiplexed in order to keep the tape drive streaming.■ Make backups while the database is open and you want to spread the I/O load across several disk spindles in order to leave bandwidth for online operations. <p>If you tune backup performance by specifying filesperset but not diskratio, diskratio defaults to the same value as filesperset. If neither is specified, diskratio defaults to 4.</p> <p>Note: Do not spread the I/O over more than the minimum number of disks to keep the tape streaming, since otherwise you do not improve performance and also increase restore time for a single file.</p>

proxy	<p>backs up the specified files using the proxy copy functionality, which gives the media management software control over the data transfer between storage devices and the Oracle datafiles on disk. The media manager—not RMAN—decides how and when to move data.</p> <p>For each file that you attempt to proxy copy, RMAN queries the media manager to determine whether it can copy the file. If the media manager cannot proxy copy the file, then RMAN uses conventional backup sets to back up the file.</p>
	<p>only causes Oracle to issue an error message when it cannot proxy copy rather than creating conventional backup sets.</p>
pool <i>integer</i>	<p>specifies the media pool in which the backup should be stored. Consult your media management documentation to see whether the pool option is supported.</p>
<i>backupSpec</i>	<p>A <i>backup_specification_list</i> contains a list of one or more <i>backupSpec</i> clauses. A <i>backupSpec</i> clause minimally contains a <i>backup_object_list</i>, which is a list of one or more objects to be backed up.</p> <p>Each <i>backupSpec</i> clause generates one or more backup sets. A <i>backupSpec</i> clause will generate multiple backup sets if the number of datafiles specified in or implied by its <i>backup_object_list</i> exceeds the filesper limit.</p>
	<p>datafile specifies a list of one or more datafiles (see "datafileSpec" on page 11-60). <i>datafileSpec</i></p> <p>Note: If you back up <code>datafile 1</code>, which is the first file of the SYSTEM tablespace, RMAN automatically includes the control file in the backup set.</p>
	<p>datafile copy specifies the filenames of one or more datafile image copies. <i>'filename'</i></p>
	<p>datafile copy tag specifies a list of one or more datafile copies, identified by tag. If multiple datafile copies with this tag exist, then Oracle backs up only the most current datafile copy of any particular datafile. <i>tag_name</i></p>
	<p>tablespace specifies the names of one or more tablespaces. RMAN backs up all datafiles that are currently part of the tablespaces. <i>tablespace_name</i></p> <p>This keyword is provided merely as a convenience; Oracle translates the tablespace name internally into a list of datafiles.</p>
	<p>database specifies the control file and all datafiles in the database. This keyword is provided merely as a convenience; Oracle translates the tablespace name internally into a list of datafiles.</p>
	<p><i>archivelogRecord-Specifier</i> clause specifies a range of archived redo logs. See "archivelogRecordSpecifier" on page 11-17.</p>
	<p>current controlfile specifies the current control file.</p>
	<p>controlfile copy specifies the filename of a control file copy. <i>'filename'</i></p>

parms <i>'channel_parms'</i>	<p>specifies parameters regarding the device to allocate. Do not use this port-specific string if you have specified type disk.</p> <p>If you use parms in conjunction with type 'sbt_tape', then you can specify environment variables. Following are models for acceptable syntax:</p> <pre>PARMS="ENV=(var1=value1,var2=value2,var3=value3 . . .)" PARMS="BLKSIZE=integer"</pre> <p>For example, you can specify:</p> <pre>PARMS="BLKSIZE=16384,ENV=(NSR_SERVER=tape_server,NSR_CLIENT=oracleclnt,NSR_GROUP=oracle_tapes)"</pre> <p>The maximum length of the quoted string is 1000 bytes.</p>
format <i>'format_string'</i>	Specifies the filename for the backup piece. See the description of the format parameter at the command level.
filesperset <i>integer</i>	specifies the maximum number of datafiles to place in one backup set. See the discussion of filesperset at the command level.
channel <i>channel_id</i>	specifies the name of a channel to use when creating the backup set for this <i>backupSpec</i> clause. See the discussion of channel at the command level.
setsize <i>integer</i>	specifies a maximum size for a backup set in units of 1K (1024 bytes). See the description of the setsize parameter at the command level.
tag <i>tag_name</i>	creates a tag for the backup set. See the discussion of the tag parameter at the command level for more information.
diskratio <i>integer</i>	specifies the number of disks involved in the backup. See the discussion of the diskratio parameter at the command level for more information.
delete input	<p>deletes the input files upon successful creation of the backup set. Specify this option only when backing up archived redo logs or datafile copies. It is equivalent to issuing a change ... delete command for all of the input files.</p> <p>Note: The backup command only backs up one copy of each distinct log sequence number, so if the delete input option is requested, RMAN only deletes the copy of the file that it backed up.</p>
skip	skips datafiles that are offline , readonly , or inaccessible . See the description of the skip option at the command level.
pool	specifies the media pool in which the backup should be stored. See the description of pool at the command level.
include current controlfile	creates a snapshot of the current control file and places it into each backup set produced by this clause.

Examples

Backing up Tablespaces and Datafiles This command uses two *backupSpec* clauses to back up tablespaces and datafiles:

```
run {
  allocate channel dev1 type disk;
  allocate channel dev2 type disk;
  backup
    (tablespace system,sales1,sales2,sales3
     filesperset 20
     skip readonly
     channel dev1)
    (datafile 1, 4, 5
     channel dev2);
}
```

Performing a Cumulative Incremental Backup of a Database This example backs up all blocks changed in the database since the most recent level 0 or level 1 backup:

```
run {
  allocate channel dev1 type 'sbt_tape';
  backup
    incremental level 2 cumulative
    # do not include inaccessible datafiles in the backup
    skip inaccessible
    database;
}
```

Duplexing a Backup Set When duplexing backup sets, specify the **set duplex** command before allocating a channel:

```
run {
  # generate four identical backup sets of datafile 1
  set duplex=4;
  allocate channel dev1 type 'sbt_tape';
  backup datafile 1;
}
```

Checking for Corruption This example backs up datafile 3 and specifies that no more than 2 blocks with physical or logical corruption will be tolerated:

```
run {
  set maxcorrupt for datafile 3 to 2;
  allocate channel dev1 type 'sbt_tape';
  backup check logical
    datafile 3;
}
```

Related Topics

["allocate"](#) on page 11-9

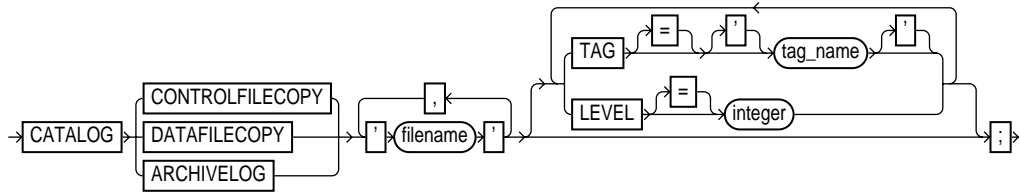
["archivelogRecordSpecifier"](#) on page 11-17

["printScript"](#) on page 11-86

["set_run_option"](#) on page 11-133

catalog

Syntax



Purpose

Use the **catalog** command to:

- Add information about an O/S datafile copy, archived redo log, or control file copy to the recovery catalog and control file.
- Catalog a datafile copy as a level 0 backup, which enables you to use it as part of an incremental backup strategy.
- Record the existence of backups of version 8 databases created before RMAN was installed.
- Record the existence of Oracle7 backups of read-only or offline normal files made before migrating to Oracle 8 or Oracle8i.

Requirements

- Execute the **catalog** command only at the RMAN prompt.
- Operate RMAN with a recovery catalog.
- For an O/S backup to be cataloged, it must be:
 - Accessible on disk.
 - A complete image copy of a single file.
 - A consistent or inconsistent whole database, tablespace, datafile, control file, or archived redo log backup. If inconsistent, it must have been created using the BEGIN BACKUP/END BACKUP statements. If a control file backup, it should have been made using the ALTER DATABASE BACKUP CONTROLFILE statement.

RMAN treats all such O/S backups as datafile copies.

You *cannot* use **catalog** to perform the following operations:

- Catalog archived redo logs and control file copies that were created in Oracle7 unless the file belongs to a tablespace that was offline normal or read-only when you migrated the database to Oracle version 8.0 or later.
- Re-catalog backup pieces or backup sets.

Keywords and Parameters

controlfilecopy <i>'filename'</i>	specifies the filename of a control file copy to be added to or updated in the recovery catalog and control file.
datafilecopy <i>'filename'</i>	specifies the filename of a datafile copy to be added to or updated in the recovery catalog and control file.
archivelog <i>'filename'</i>	specifies the filename of an archivelog copy to be added to or updated in the recovery catalog and control file.
tag <i>tag_name</i>	specifies the tag of the input file, e.g., <i>Sunday_PM_Backup</i> .
level <i>integer</i>	indicates that the file copy should be recorded as an incremental backup at the specified level, typically level 0. You can perform incremental backups using a datafile copy as the base level 0 backup.

Examples

Cataloging an Archived Redo Log This statement catalogs the archived redo logs log1, log2, and log3:

```
catalog archivelog 'log1', 'log2', 'log3';
```

Cataloging a File Copy as an Incremental Backup The following example catalogs datafile copy tbs_2.c as an incremental level 0 backup:

```
catalog datafile '/oracle/copy/tbs_2.c' level 0;
```

Cataloging an O/S Copy The following makes an O/S copy of a datafile using the RMAN **host** command and then catalogs the copy (sample output included):

```
host 'cp $ORACLE_HOME/dbs/sales.f $ORACLE_HOME/dbs/sales.bak';
catalog datafilecopy '$ORACLE_HOME/dbs/sales.bak';
```

```
RMAN-03022: compiling command: catalog
RMAN-03023: executing command: catalog
```

```
RMAN-08050: cataloged datafile copy
RMAN-08513: datafile copy filename=/oracle/dbs/sales.bak recid=121 stamp=342972501
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete
```

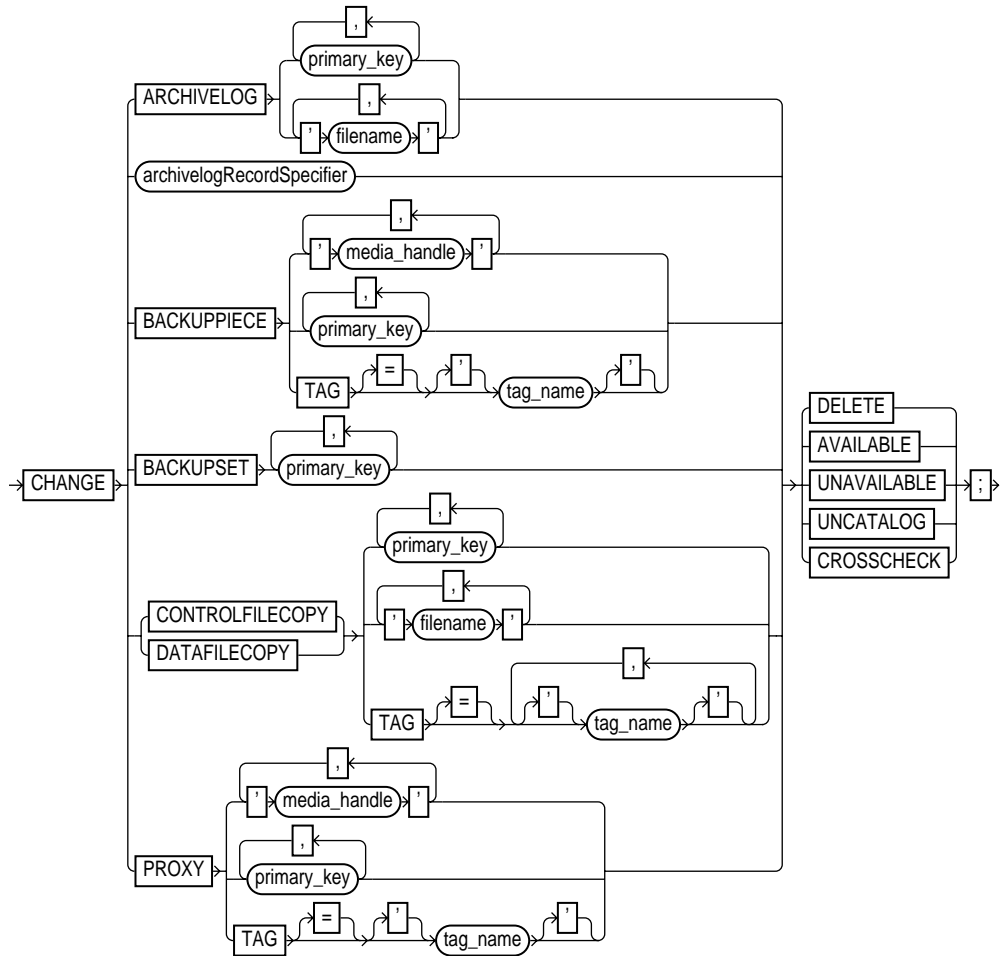
Related Topics

["list"](#) on page 11-76

["report"](#) on page 11-102

change

Syntax



Purpose

To change the status of backups and copies in the RMAN metadata. Use this command to:

- Mark a backup or copy as having the status **unavailable** or **available**.
- Delete a backup or copy from the O/S and update its status to **deleted**.
- Check whether backups, image copies, and archived redo logs are available and, if they are not, mark them as **expired**.

Requirements

- Execute the **change** command at the RMAN prompt or within a **run** command.
- Use the command only on files that are recorded in the RMAN metadata and belong to the current database incarnation.
- The following options require a recovery catalog:
 - **available**
 - **unavailable**
 - **uncatalog**
 - **crosscheck** (only required for backup sets and backup pieces)
- Issue an **allocate channel for delete** or **allocate channel for maintenance** command before issuing the following:
 - **change backupset ... delete**
 - **change backuppiece ... delete**
 - **change backupset ... crosscheck**
 - **change backuppiece ... crosscheck**

Keywords and Parameters

To obtain the primary keys of the records whose status you want to change, issue a **list** command or query the recovery catalog views.

archivelog	specifies an archived redo log by either <i>primary_key</i> or <i>'filename'</i> .
<i>archivelogRecord-Specifier</i> clause	specifies a range of archived redo logs. See " archivelogRecordSpecifier " on page 11-17.
backuppiece	specifies a backup piece by <i>primary_key</i> , <i>'media_handle'</i> , or <i>tag_name</i> .

backupset <i>primary_key</i>	specifies a backup set by <i>primary_key</i> .
controlfilecopy	specifies a control file copy by <i>primary_key</i> , <i>'filename'</i> , or <i>tag_name</i> . If you crosscheck a control file copy, you must specify a filename rather than a primary key.
datafilecopy	specifies a datafile copy by either <i>primary_key</i> , <i>'filename'</i> , or <i>tag_name</i> .
proxy	specifies a proxy copy by <i>primary_key</i> or <i>'filename'</i> , or <i>tag_name</i> .
delete	changes the status of a backup or copy to deleted and physically deletes the file from the O/S.
available	marks a backup or copy as having the status available . View the status in the list output.
unavailable	marks a backup or copy as having the status unavailable . View the status in the list output. This option is provided for cases when the file cannot be found or has migrated offsite. A file that is marked unavailable will not be used in a restore or recover command. If the file is later found or returns to the main site, then you can use the available option to reflect this change.
uncatalog	removes references to a datafile copy or archived redo log (but not a backup piece or backup set) from the recovery catalog. Use this command to notify RMAN when a file is deleted by some means other than a change ... delete command. If you attempt to use the uncatalog option on a backup piece or backup set, RMAN returns an error message. WARNING: If you resynchronize from a backup control file, uncataloged records can reappear in the metadata. Note: If you want to remove all records with deleted status at once, execute the <code>prgrmanc.sql</code> script located in the <code>\$ORACLE_HOME/admin</code> directory. See " Deleting Backups and Copies and Updating Their Status in the RMAN Metadata " on page 6-13 for instructions.
crosscheck	checks whether the specified backups and copies exist. If RMAN cannot find backup pieces, it marks them as having the status expired . It marks all other types of absent files—image copies and archived redo logs—as deleted . If the files are on disk, RMAN queries the file headers. For other device types, RMAN queries the media manager to see whether the file exists in the media management catalog. Note: RMAN considers archived redo logs as copies, so issue the change archivelog all crosscheck command if one or more logs become unavailable. If the archived logs become unavailable again, you must issue catalog archivelog to re-catalog them. Note: If you crosscheck a control file copy, specify a filename rather than a primary key.

Examples

Deleting a Backup Piece This example deletes a backup piece stored by a media manager and changes its status to **deleted** in the metadata (note that the **list** output does not display records with **deleted** status; you must access these records through the recovery catalog views):

```
allocate channel for delete type 'sbt_tape';
change backuppiece '$ORACLE_HOME/dbs/testdb_87fa39e0' delete;
release channel;
```

Marking a Backup Set as Unavailable This example marks a backup set as having the status **unavailable**. You do not need to allocate a maintenance channel:

```
change backupset 100 unavailable;
```

Crosschecking Files This example checks to see whether all of the registered archived redo logs still exist; if not, RMAN changes their status in the metadata to **expired**:

```
allocate channel for maintenance type disk;
change archivelog all crosscheck;
release channel;
```

Related Topics

["allocateForMaint"](#) on page 11-13

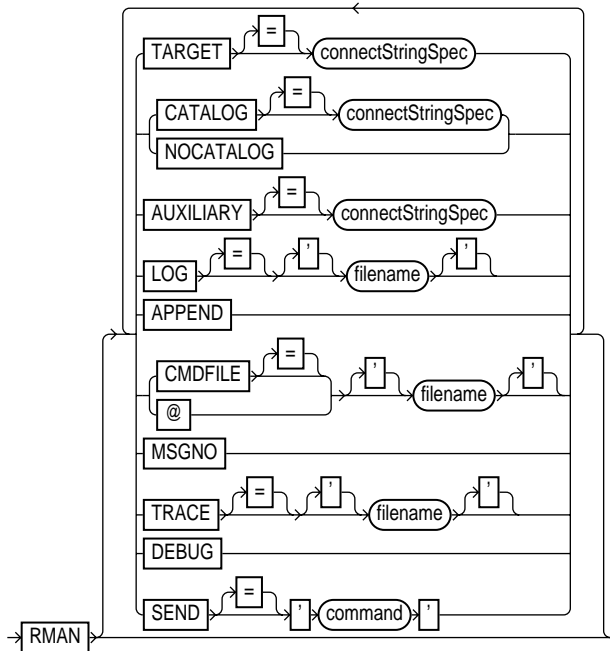
["archivelogRecordSpecifier"](#) on page 11-17

["crosscheck"](#) on page 11-57

["deleteExpired"](#) on page 11-63

cmdLine

Syntax



Purpose

To start RMAN from the O/S command line. Use these arguments to:

- Connect to the target, recovery catalog, or auxiliary database.

Note: On some platforms, you may choose to connect at the command line because the password is visible to other users on the system. The **connect** command is an alternative method that avoids this problem.

- Specify that you are using RMAN without a recovery catalog.
- Run a command file, which is a user-defined file containing RMAN commands.

- Specify the file in which RMAN records the results of processed commands.
- Add to rather than overwrite the contents of the command file.
- Generate debugging output and specify its location.
- Send a command to the media manager.
- Cause RMAN to print message numbers in the **list** output.

Requirements

Use these arguments at the O/S command line rather than at the RMAN prompt.

Keywords and Parameters

target <i>connectStringSpec</i>	specifies a connect string to the target database, e.g., target <i>sys/change_on_install@inst1</i> . See " connectStringSpec " on page 11-46.
catalog <i>connectStringSpec</i>	specifies a connect string to the database containing the recovery catalog, e.g., catalog <i>rman/rman@inst2</i> . See " connectStringSpec " on page 11-46.
nocatalog	indicates that you are using RMAN without a recovery catalog. You must use this argument when starting Recovery Manager without a recovery catalog.
auxiliary <i>connectStringSpec</i>	specifies a connect string to an auxiliary database, e.g., auxiliary <i>sys/change_on_install@dupdb</i> . See " connectStringSpec " on page 11-46.
log <i>filename</i>	specifies the file where Recovery Manager will record RMAN output, i.e., the commands that were processed and their results. If you do not specify this argument, then Recovery Manager writes its message log file to standard output.
append	causes new output to be appended to the end of the message log file. If you do not specify this parameter and a file with the same name as the message log file already exists, RMAN overwrites it.
cmdfile <i>filename</i>	runs a file containing a user-defined list of RMAN commands. If the first character of the filename is alphabetic, then you can omit the quotes around the filename. The contents of the command file should be identical to commands entered at the RMAN prompt. For example, the following file contents will cause RMAN to connect to a target database and recovery catalog RCAT: <code>connect target; connect catalog rman/rman@rcat;</code> RMAN terminates after running the command file.
@filename	equivalent to cmdfile .
msgno	causes RMAN to print message numbers, i.e., RMAN-xxxx, for the output of the list command. By default, list does not print the RMAN-xxxx prefix.

trace <i>filename</i>	specifies the name of the file in which RMAN logs debugging information. You must also specify the debug option to generate debugging output. If you specify debug without also specifying trace , then RMAN writes the debugging output to standard output or the message log if one is specified.
debug	activates the debugging feature. Use this option only for problem diagnosis under the direction of Oracle World Wide Support.
send ' <i>command</i> '	sends a vendor-specific command string to all allocated channels. See your media management documentation to determine whether this feature is supported. See Also: To send a string to specific channels, see " send " on page 11-127.

Examples

Connecting without a Recovery Catalog This example connects to the target database PROD1 without a recovery catalog:

```
% rman target sys/sys_pwd@prod1 nocatalog
```

Connecting to an Auxiliary Instance This example connects to the target database PROD1, the recovery catalog database RCAT, and the auxiliary instance AUX1:

```
% rman target sys/sys_pwd@prod1 catalog rman/rman@rcat auxiliary sys/aux_pwd@aux1
```

Specifying a Command File This example connects to the target database PROD1 and the recovery catalog database RCAT, and then runs the command file `b_whole_10.rcv`:

```
% rman target sys/sys_pwd@prod1 catalog rman/rman@rcat @'/oracle/dbs/b_whole_10.rcv'
```

Specifying a Message Log in Append Mode This example connects to the target database PROD1 without a recovery catalog and then specifies that RMAN should append messages to the message log:

```
% rman target sys/sys_pwd@prod1 nocatalog log = $ORACLE_HOME/dbs/log/msglog.f append
```

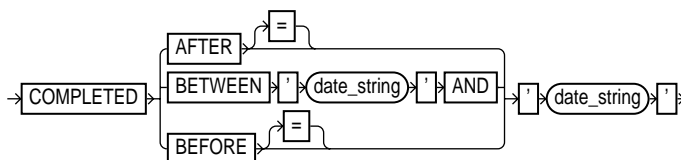
Related Topics

["connect"](#) on page 11-44

["rmanCmd"](#) on page 11-121

completedTimeSpec

Syntax



Purpose

A sub-clause that specifies when a backup or copy completed.

Requirements

All date strings must be either:

- Formatted according to the NLS date format specification currently in effect.
- Created by a SQL expression that returns a DATE value, e.g., 'SYSDATE-30'.

Use this sub-clause in conjunction with the following commands:

- [crosscheck](#)
- [deleteExpired](#)
- [list](#)

Note: The **from time** and **until time** parameters no longer work with commands that use *completedTimeSpec*. If you run a script that specifies these parameters, the job will fail.

Keywords and Parameters

after <i>'date_string'</i>	specifies the time after which the backup was completed.
between <i>'date_string'</i> and <i>'date_string'</i>	specifies a time range during which the backup was completed.
before <i>'date_string'</i>	specifies the time before which the backup was completed.

Examples

Crosschecking Backups within a Time Range This example crosschecks the backup sets of the database made last month:

```
crosscheck backup of database between 'SYSDATE-62' and 'SYSDATE-31';
```

Deleting Expired Backups This example deletes expired backup sets of datafile 1 made in the last two weeks:

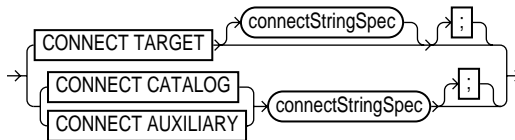
```
delete expired backup of datafile 1 after 'SYSDATE-14';
```

Listing Copies This example lists image copies of `/oracle/dbs/tbs_22.f` made before December 13, 1998:

```
list copy of datafile '/oracle/dbs/tbs_22.f' before 'Dec 13 1998 20:31:10';
```

connect

Syntax



Purpose

To establish a connection between RMAN and a target, auxiliary, or recovery catalog database.

Note: When connecting from the command line, the password may be visible to other users on the system. The **connect** command avoids this problem.

See Also: For command line connection options, see "[cmdLine](#)" on page 11-39.

Requirements

You can only use the **connect** command if you are at the RMAN prompt and if you are not already connected.

Keywords and Parameters

connect target <i>connectStringSpec</i>	establishes a connection between RMAN and the target database. See " connectStringSpec " on page 11-46.
connect catalog <i>connectStringSpec</i>	establishes a connection between RMAN and the recovery catalog database. See " connectStringSpec " on page 11-46.
connect auxiliary <i>connectStringSpec</i>	establishes a connection between RMAN and an auxiliary instance. An auxiliary instance can be used with the duplicate command or used during TSPITR. See " connectStringSpec " on page 11-46.

Examples

Connecting Without a Recovery Catalog This example starts RMAN and then connects to the target database with a Net8 service name `prod1`:

```
% rman nocatalog
RMAN> connect target sys/change_on_install@prod1;
```

Connecting with a Recovery Catalog This example starts RMAN and then connects to the target database `PROD1` using O/S authentication and the recovery catalog database `RCAT` using a password file:

```
% rman
RMAN> connect target /; connect catalog rman/rman@rcat;
```

Connecting to Target, Recovery Catalog, and Duplicate Databases This example connects to three different databases specifying a username and password for each:

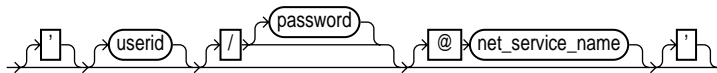
```
% rman
RMAN> connect target sys/sysdba@prod1;
RMAN> connect catalog rman/rman@rcat;
RMAN> connect auxiliary sys/sysdba@dupdb;
```

Related Topics

["cmdLine"](#) on page 11-39

connectStringSpec

Syntax



Purpose

A sub-clause specifying the username, password, and net service name for connecting to a target, recovery catalog, or auxiliary database. The connection is necessary to authenticate the user and identify the database.

Requirements

- You must have SYSDBA privileges on the target and auxiliary databases.
- Do not connect to the recovery catalog database as user SYS.

Keywords and Parameters

/	<p>if you do not specify a userid or password when connecting to the target database, a forward slash establishes a connection as SYS using O/S authentication. For example, enter the following to connect to the target database:</p> <pre>% rman target /</pre> <p>Note: The forward slash depends on the ORACLE_SID environment variable to know which database you want to connect to. The ORACLE_SID can point to either the auxiliary or target database, but not both at the same time. You cannot connect to the recovery catalog database using only the forward slash.</p>
userid	<p>establishes a connection to the database for the specified user. If you do not specify a password, RMAN obtains the password interactively by displaying a prompt. The characters will not be echoed to the terminal.</p> <p>You must have SYSDBA authority when connecting to the target database, but must <i>not</i> connect as SYS to the recovery catalog database.</p> <p>Note: The connect string must not contain any white space, but it can contain punctuation characters such as “/” and “@”.</p>
/password	<p>establishes a connection for the specified user using a password. If the target database is not open, then a password file must exist.</p>

`@net_service_name` establishes a connection to the database using an optional Net8 net service name. The service name must be valid as specified in the `tnsnames.ora` file.

Examples

Connecting Without a Recovery Catalog This example connects to the target database using a password and the Net8 service name PROD1:

```
% rman target sys/change_on_install@prod1 nocatalog
```

Entering the Password Interactively This example connects to the target database as user SYS but without specifying a password at the command line:

```
% rman target sys
```

```
Recovery Manager: Release 8.1.1.5.0.0
```

```
target database Password:
```

Connecting with O/S Authentication This example starts RMAN and then connects to the target database PROD1 using O/S authentication and the recovery catalog database RCAT using a password file:

```
% rman
RMAN> connect target /
RMAN> connect catalog rman/rman@rcat
```

Connecting to a Target Database, Recovery Catalog, and Auxiliary Instance This example connects to three different databases from the command line, specifying a username, password, and net service name for each:

```
% rman target sys/sysdba@prod1 catalog rman/rman@rcat auxiliary sys/sysdba@dupdb
```

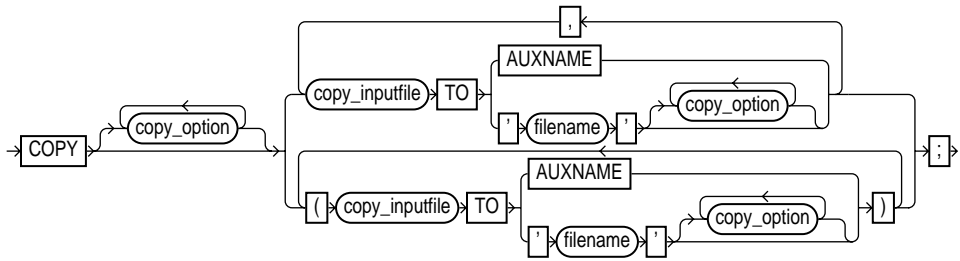
Related Topics

["cmdLine"](#) on page 11-39

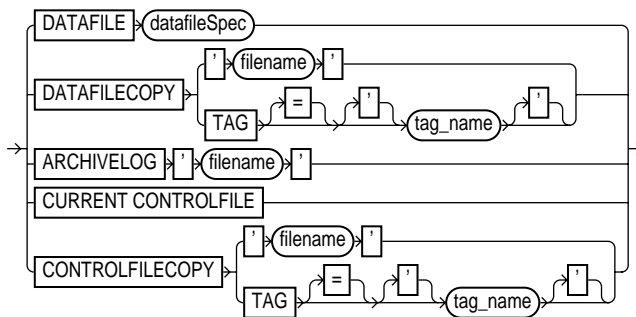
["connect"](#) on page 11-44

copy

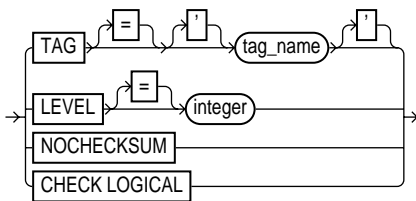
Syntax



copy_inputfile::=



copy_option::=



Purpose

Create an image copy of a file. The output file is always written to disk. You can copy the following types of files:

- Datafiles (current or copies)
- Archived redo logs
- Control files (current or copies)

In many cases, copying datafiles is more beneficial than backing them up, since the output is suitable for use without any additional processing. In contrast, you must process a backup set with a **restore** command before it is usable. So, you can perform media recovery on a datafile copy, but not directly on a backup set, even if it backs up only one datafile and contains a single backup piece.

Requirements

- Execute the command from within the braces of a **run** command.
- Precede a **copy** command with at least one **allocate channel** command specifying the **type disk** option.
- You cannot make incremental copies.

Keywords and Parameters

<i>copy_option</i>	specifies optional parameters affecting either the input or output files or both.
tag <i>tag_name</i>	specifies the tag of the input file or output file copy.
level <i>integer</i>	includes the input file or output file copy in the incremental backup strategy by making it serve as a basis for subsequent incremental backup sets. Typically, you specify level 0 . If you do not use the level option, then the datafile copy has no impact on the incremental backup strategy.
nochecksum	suppresses block checksums. Unless you specify this option, Oracle computes a checksum for each block. RMAN verifies the checksum when restoring the copy. If the database is already maintaining block checksums, then this flag has no effect.

check logical	<p>tests data and index blocks that pass physical corruption checks for logical corruption, e.g., corruption of a row piece or index entry. If RMAN finds logical corruption, it logs the block in the <code>alert.log</code> and server session trace file.</p> <p>Provided the sum of physical and logical corruptions detected for a file remain below its maxcorrupt setting, the RMAN command completes and Oracle populates <code>V\$BACKUP_CORRUPTION</code> and <code>V\$COPY_CORRUPTION</code> with corrupt block ranges. If maxcorrupt is exceeded, the command terminates without populating the views.</p> <p>Note: For copy and backup the maxcorrupt setting represents the total number of physical and logical corruptions permitted on a file.</p>
<i>copy_inputfile</i> clause	specifies the type of input file, i.e., the file that you want to copy.
datafile <i>datafileSpec</i>	<p>specifies a list of one or more datafiles as input. See "datafileSpec" on page 11-60.</p> <p>Note: If you specify a filename, then it must be the name of a current datafile as listed in the control file.</p>
datafilecopy	<p>specifies a list of one or more datafile copies as input. Specify the datafile copies by '<i>filename</i>' or tag = <i>tag_name</i>. The filename must <i>not</i> be the name of a current datafile listed in the control file. The existing copy may have been created by either a previous copy command or by an external O/S utility.</p>
archivelog <i>'filename'</i>	<p>specifies the filename of an input archived redo log. The archived log may have been created by the Oracle archiving session or by a previous copy command. Specify the archived redo log by filename.</p>
current controlfile	specifies the current control file.
controlfilecopy <i>'filename'</i>	<p>specifies the filename of a control file copy. You can also set tag = <i>tag_name</i> to specify a list of one or more control file copies.</p> <p>Note: The control file copy is marked as a backup control file, so media recovery will be necessary if you mount the control file copy. This command is equivalent to the <code>ALTER DATABASE BACKUP CONTROLFILE TO '...'</code> statement.</p>
to 'filename'	specifies the filename of the output file copy.
to auxname	specifies that Oracle should copy the input datafile to the filename specified in an earlier set auxname command for the input datafile.

Examples

Copying a Datafile This example copies the datafile `tbs_01.f` with the **nochecksum** option to the output file `temp3.f`, marking it as a level 0 backup:

```
run {
  allocate channel dev1 type disk;
  copy
    nochecksum
    datafile '$ORACLE_HOME/dbs/tbs_01.f'
      to '$ORACLE_HOME/copy/temp3.f'
    level 0;
}
```

Copying the Control File This example copies the current control file and gives the copy the tag *weekly_cf_copy*:

```
run {
  allocate channel dev1 type disk;
  copy
    current controlfile
      to '$ORACLE_HOME/copy/cf1.f'
    tag = 'weekly_cf_copy';
}
```

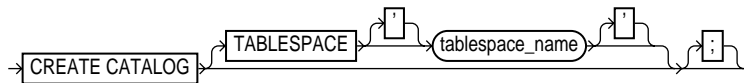
Related Topics

["allocate"](#) on page 11-9

["backup"](#) on page 11-21

createCatalog

Syntax



Purpose

To create a schema for the recovery catalog. Typically, you create this schema in a separate recovery catalog database.

Note: In releases prior to 8.1.5, you created the recovery catalog schema by connecting to the recovery catalog database and executing the `catrman.sql` script.

Requirements

- Execute this command only at the RMAN prompt.
- The recovery catalog owner must be granted the `RECOVERY_CATALOG_OWNER` role, and also be granted space privileges in the tablespace where the recovery catalog tables will reside.
- RMAN must be connected to the recovery catalog either via the **catalog** command-line option (see "[cmdLine](#)" on page 11-39) or the **connect catalog** command.
- Do not create the recovery catalog in the SYS schema.

See Also: For more information about the `RECOVERY_CATALOG_OWNER` role, see the *Oracle8i Administrator's Guide*.

Keywords and Parameters

tablespace	specifies the tablespace in which to store the recovery catalog schema. The catalog owner
<i>tablespace_name</i>	must be granted quota privileges. If you do not specify a tablespace, RMAN stores the recovery catalog in the SYSTEM tablespace.

Examples

Creating a Catalog Schema This example creates a user RMAN, grants RMAN the RECOVERY_CATALOG_OWNER role, then creates the recovery catalog in the schema RMAN.RCVCAT of the database RCAT:

```
% sqlplus sys/change_on_install@rcat;

SQL> CREATE USER rman IDENTIFIED BY rman
      2> DEFAULT TABLESPACE rcvcat QUOTA UNLIMITED ON rcvcat;
SQL> GRANT recovery_catalog_owner TO rman;
SQL> exit

% connect rman/rman@rcat;
RMAN> create catalog tablespace rcvcat;
```

Related Topics

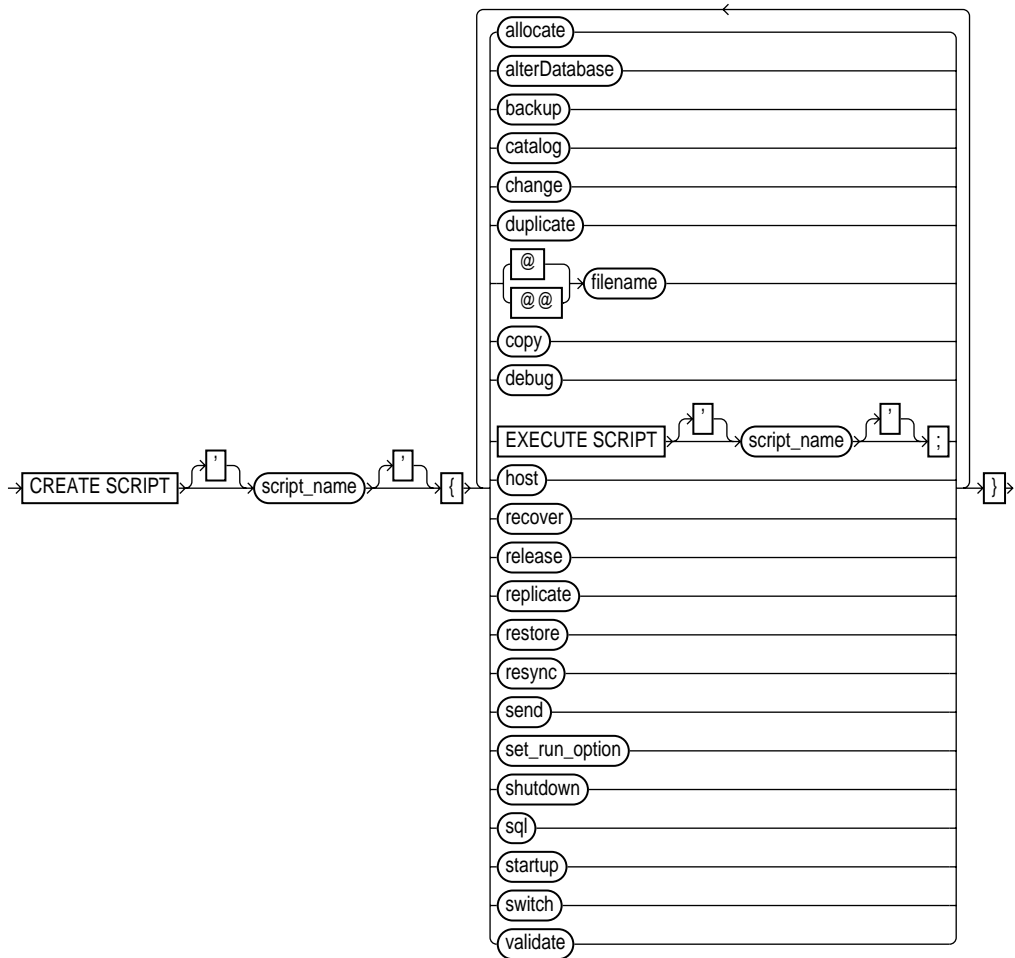
["connect" on page 11-44](#)

["dropCatalog" on page 11-68](#)

["upgradeCatalog" on page 11-148](#)

createScript

Syntax



Purpose

To create a script and store it in the recovery catalog for future reference. Stored scripts provide a common repository for frequently executed collections of RMAN commands: use any command legal within a **run** command in the script. The script is not executed immediately; use the **execute script** command (see "run" on page 11-124) to run it.

Requirements

Note the following restrictions:

- Execute **create script** only at the RMAN prompt.
- You must be connected to the recovery catalog.
- You cannot execute a **run** command within a stored script. When you run an **execute script** command within a **run** command, RMAN places the contents of the script between the braces of **run**. For this reason, you should not allocate a channel at the **run** command level if you already allocated it in the script.

Keywords and Parameters

For descriptions of the individual commands that you can use in a stored script, see the appropriate entry, e.g., "backup" on page 11-21. Note that the @@ command exhibits special behavior when you execute it within a script. For information on the **execute script** command, see "run" on page 11-124.

<i>script_name</i>	<p>creates a stored script with the specified name. The statements allowable within the parentheses of the create script 'script_name' (...) command are the same allowable within the run command. The statements within the braces constitute the <i>job_command_list</i>.</p> <p>Note: To execute the stored script, use the execute script command within the braces of the run command.</p>
<i>@filename</i>	<p>executes a series of RMAN commands stored in an O/S file with the specified full pathname, e.g., @\$ORACLE_HOME/dba/cmd/cmd1.f. Do not use quotes around the string or leave whitespace between the @ and filename. RMAN processes the specified file as if its contents had appeared in place of the @ command.</p> <p>Note: The file must contain only complete Recovery Manager commands. A syntax error will result if the file contains a partial command.</p>

@@filename specifies the relative filename of an O/S file containing a series of RMAN commands, e.g., `cmd1.f`. The command file specified by @@ is assumed to be in the same directory as the parent script. Do not use quotes around the string or leave whitespace between the @@ and filename. RMAN processes the specified file as if its contents had appeared in place of the @@ command.

Note: The file must contain only complete Recovery Manager commands. A syntax error will result if the file contains a partial command

Examples

Creating a Script This example creates a script called `B_WHOLE_10` that backs up the database and archived redo logs, then executes it:

```
create script b_whole_10 {
  allocate channel d1 type disk;
  allocate channel d2 type disk;
  allocate channel d3 type disk;
  backup
    incremental level 0
    tag b_whole_10
    filesperset 6
    database;
  sql 'ALTER SYSTEM ARCHIVE LOG CURRENT';
  backup
    filesperset 20
    archivelog all
    delete input;
}
```

```
RMAN-03022: compiling command: create script
RMAN-03023: executing command: create script
RMAN-08085: created script b_whole_10
```

```
run { execute script b_whole_10; }
```

Related Topics

["deleteScript" on page 11-65](#)

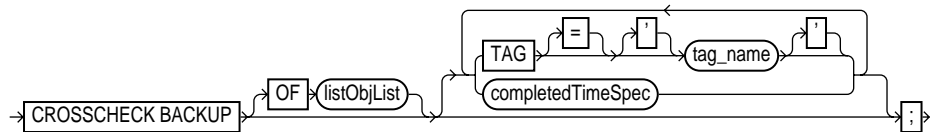
["printScript" on page 11-86](#)

["replaceScript" on page 11-97](#)

["run" on page 11-124](#)

crosscheck

Syntax



Purpose

To determine whether backups stored on disk or tape exist. Backups are either backup sets or media-managed proxy copies.

The **crosscheck** command checks only backup sets marked **available** or **expired**, either by examining the backup pieces on disk when the channel is **type disk**, or by querying the media manager when the channel is **type 'sbt_tape'**. It only processes backups created on the specified channel.

RMAN does not delete any backup pieces that it is unable to find, but updates their metadata records to **expired** status. If some backup pieces were erroneously marked as **expired**, e.g., because the media manager was misconfigured, then after ensuring that the files really do exist in the media manager, run the **crosscheck backup** command again to restore those files to **available** status.

Requirements

- Execute **crosscheck backup** only at the RMAN prompt.
- Allocate a maintenance channel before issuing **crosscheck backup**.
- You must use a recovery catalog.

Keywords and Parameters

of <i>listObjList</i>	restricts the list of objects operated on to the object type specified in the <i>listObjList</i> clause. If no objects are specified, the command checks all objects: of database controlfile archivelog all . See " listObjList " on page 11-84.
tag <i>tag_name</i>	specifies the tag for the backup set.
<i>completedTimeSpec</i>	specifies a time range for backup completion. See " completedTimeSpec " on page 11-84.

Examples

Crosschecking All Backups The following example queries the status of all backups on disk and includes sample output:

```
RMAN> allocate channel for maintenance type disk;

RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: delete
RMAN-08500: channel delete: sid=15 devtype=DISK

RMAN> crosscheck backup;

RMAN-03022: compiling command: XCHECK
RMAN-03023: executing command: XCHECK
RMAN-08517: backup piece handle=/vobs/oracle/dbs/01a8t4pq_1_1 recid=1 stamp=3448
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/vobs/oracle/dbs/05a9cfs3_1_1 recid=5 stamp=3453
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/vobs/oracle/dbs/06a9cfv8_1_1 recid=6 stamp=3453
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/vobs/oracle/dbs/07a9ck4t_1_1 recid=7 stamp=3453
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'
RMAN-08517: backup piece handle=/vobs/oracle/dbs/08a9cl23_1_1 recid=8 stamp=3453
RMAN-08074: crosschecked backup piece: found to be 'EXPIRED'
RMAN-08517: backup piece handle=/vobs/oracle/dbs/09a9cl2b_1_1 recid=9 stamp=3453
RMAN-08074: crosschecked backup piece: found to be 'AVAILABLE'

RMAN> release channel;

RMAN-03022: compiling command: release
RMAN-03023: executing command: release
RMAN-08031: released channel: delete
```

Crosschecking Within a Range of Dates The following example queries the media manager for the status of the backup sets for datafile 3 in a given six-month range. Note that RMAN uses the date format specified in the `NLS_DATE_FORMAT` parameter, which is 'DD-MON-YY' in this example:

```
allocate channel for maintenance type 'sbt_tape';
crosscheck backup of datafile 3 device type 'sbt_tape' completed between '01-JAN-98' and
'01-JUL-98';
release channel;
```


Related Topics

["allocateForMaint"](#) on page 11-13

["change"](#) on page 11-35

["list"](#) on page 11-76

["report"](#) on page 11-102

datafileSpec

Syntax



Purpose

A sub-clause that specifies a datafile by filename or absolute file number.

Keywords and Parameters

<i>'datafile'</i>	specifies the datafile using either the full path or a relative filename. If you specify a relative filename, the filename is qualified in a port-specific manner by the target database.
<i>integer</i>	specifies the datafile using its absolute file number. Obtain the file number from the V\$DATAFILE, V\$DATAFILE_COPY, or V\$DATAFILE_HEADER views or report schema command output.

Examples

Specifying a Datafile by Filename This example copies datafile `/oracle/dbs/tbs_12` to disk, specifying it by filename:

```
run {
  allocate channel ch1 type disk;
  copy datafile '/oracle/dbs/tbs_12.f'
    to '/oracle/copy/tbs_1.copy';
}
```

Specifying a Datafile by Absolute File Number This example copies datafile `/oracle/dbs/tbs_31.f` to disk, specifying it by file number:

```
RMAN> report schema;
```

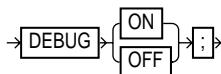
```
RMAN-03022: compiling command: report
Report of database schema
File K-bytes    Tablespace          RB segs Name
-----
1          47104 SYSTEM              YES    /vobs/oracle/dbs/tbs_01.f
2           978 SYSTEM              YES    /vobs/oracle/dbs/tbs_02.f
```

3	978 TBS_1	NO	/vobs/oracle/dbs/tbs_11.f
4	978 TBS_1	NO	/vobs/oracle/dbs/tbs_12.f
5	978 TBS_2	NO	/vobs/oracle/dbs/tbs_21.f
6	978 TBS_2	NO	/vobs/oracle/dbs/tbs_22.df
7	500 TBS_1	NO	/vobs/oracle/dbs/tbs_13.f
8	500 TBS_2	NO	/vobs/oracle/dbs/tbs_23.f
9	500 TBS_2	NO	/vobs/oracle/dbs/tbs_24.f
10	500 TBS_3	NO	/vobs/oracle/dbs/tbs_31.f

```
run {
  allocate channel ch1 type disk;
  copy datafile 10
  to '/oracle/copy/tbs_31.copy';
}
```

debug

Syntax



Purpose

To turn RMAN's debugging feature off and on. Use this feature under the guidance of Oracle World Wide Support, since its purpose is to diagnose RMAN problems.

Requirements

Execute this command at the RMAN prompt or within the braces of a **run** command.

Keywords and Parameters

on	activates the debugging feature on.
off	deactivates the debugging feature.

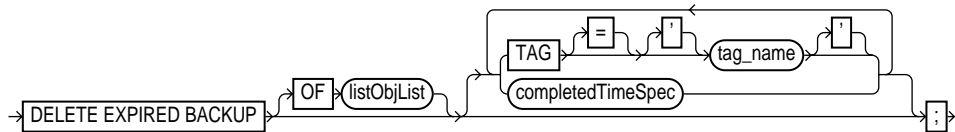
Examples

Activating the Debugging Feature In this example, debug output will be displayed during the backup of datafile 3, but not datafile 4:

```
run {
  allocate channel c1 type disk;
  debug on;
  backup datafile 3;
  debug off;
  backup datafile 4;
}
```

deleteExpired

Syntax



Purpose

To update backup set records from status **expired** to status **deleted** in the RMAN metadata. This command will operate only on the recovery catalog records for the backup pieces marked **expired** by the **crosscheck** command. Use the **list** command or query the recovery catalog views to obtain the status of backup sets.

Note: If for some reason a backup set marked **expired** exists when you run the **delete expired backup** command, RMAN deletes the physical file.

Requirements

- You must be using a recovery catalog.
- Execute **delete expired backup** only at the RMAN prompt.
- Precede **delete expired backup** with an **allocate channel for delete** or an **allocate channel for maintenance** command (see "[allocateForMaint](#)" on page 11-13).

Keywords and Parameters

<i>of listObjList</i>	restricts the list of objects to the object type specified in the <i>listObjList</i> clause. If no objects are specified, the command deletes all expired objects: of database controlfile archivelog all . See " listObjList " on page 11-84.
tag <i>tag_name</i>	specifies the tag for the backup set.
<i>completedTimeSpec</i>	specifies a time range for backup completion. See " completedTimeSpec " on page 11-84.

Examples

Deleting Expired Backups The following example checks the media manager for expired backups of the tablespace USER_DATA that are more than one month old and updates their records to **deleted** status:

```
allocate channel for delete chl type 'sbt_tape';
crosscheck backup of tablespace user_data completed before 'SYSDATE-31';
delete expired backup of tablespace user_data' completed before 'SYSDATE-31';
release channel;
```

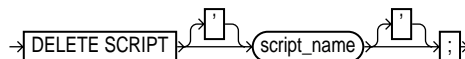
Related Topics

["allocateForMaint"](#) on page 11-13

["crosscheck"](#) on page 11-57

deleteScript

Syntax



Purpose

To delete a stored script from the recovery catalog.

Requirements

- Execute **delete script** only at the RMAN prompt.
- You must be using a recovery catalog.

Keywords and Parameters

<i>script_name</i>	deletes the specified script. The script name must be one of the names specified in a previous create script or replace script command (see " createScript " on page 11-54).
--------------------	--

Examples

Deleting a Script This example deletes the script `b_whole_10`:

```
delete script 'b_whole_10';
```

Related Topics

["createScript"](#) on page 11-54

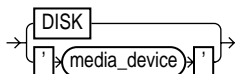
["printScript"](#) on page 11-86

["replaceScript"](#) on page 11-97

["run"](#) on page 11-124

deviceSpecifier

Syntax



Purpose

A sub-clause specifying the type of storage for a backup or copy.

Keywords and Parameters

disk	specifies disk storage.
<i>'media_device'</i>	specifies a sequential I/O device or access method for storage. The syntax and semantics of sequential I/O device types are platform-specific. Currently, the only available value is <i>'sbt_tape'</i> , which functions with a third-party tape sub-system interface.

Examples

Allocating a Tape Channel This example allocates a maintenance channel for a media management device:

```
allocate channel for maintenance type 'sbt_tape';
```

Backing Up to Disk This example backs up the database to disk:

```
run {
  allocate channel ch1 type disk;
  backup database;
}
```

Restoring from Disk and Tape This example recovers the database using backups from disk and tape:

```
run {
  allocate channel d1 type disk;
  allocate channel t1 type 'sbt_tape'
  restore database;
  recover database;
}
```


Related Topics

["allocate"](#) on page 11-9

["allocateForMaint"](#) on page 11-13

["list"](#) on page 11-76

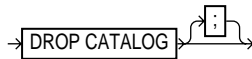
["releaseForMaint"](#) on page 11-96

["report"](#) on page 11-102

["restore"](#) on page 11-112

dropCatalog

Syntax



Purpose

To remove the schema from the recovery catalog.

WARNING: This command deletes all information from the recovery catalog; if you have no backups of the recovery catalog, then all backups of all databases managed by this recovery catalog become unusable after you execute this command.

Requirements

- Execute this command only at the RMAN prompt.
- You must be connected to the recovery catalog via the **catalog** command-line option (see "[cmdLine](#)" on page 11-39) or the **connect catalog** command.
- Enter the command twice to confirm that you want to drop the schema.

Examples

Deleting the Catalog This example drops the schema from the recovery catalog (you must enter the command twice to confirm):

```
RMAN> drop catalog
```

```
RMAN-06435: recovery catalog owner is rman
```

```
RMAN-06436: enter DROP CATALOG command again to confirm catalog removal
```

```
RMAN> drop catalog
```

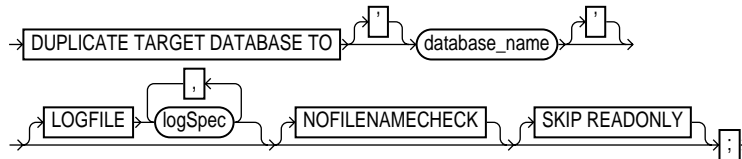
Related Topics

["createCatalog"](#) on page 11-52

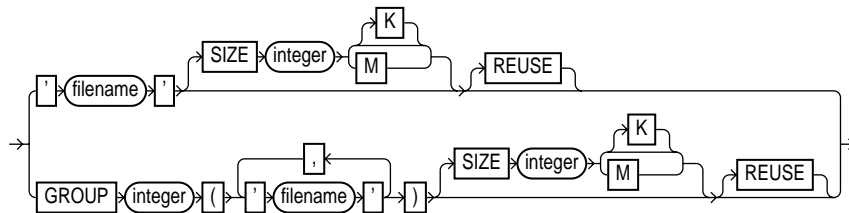
["upgradeCatalog"](#) on page 11-148

duplicate

Syntax



logSpec::=



Purpose

To use backups of the target database to create a duplicate database. A duplicate database provides a safe environment for testing backup and recovery procedures.

See Also: To learn how to duplicate a database, see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#).

Requirements

- Execute this command only within the braces of a **run** command.
- This command does **not** require the use of a recovery catalog.
- Issue one or more **allocate auxiliary channel** commands before executing the **duplicate** command. If you are duplicating from image copies or disk backups, then the more channels you allocate, the faster the duplicating operation.
- You must be connected to both the target database and auxiliary instance.
- The auxiliary instance must be started with the NOMOUNT option.

- The target database must be open.
- If your target and duplicate databases reside on the same host, set the `CONTROL_FILES` parameter appropriately so that the duplicate database control file does not overwrite the target control file. See the *Oracle8i Reference* for a description of this parameter.
- If your target and duplicate databases share the same host, set all `*_PATH` and `*_DEST` initialization parameters appropriately so that your target database files are not overwritten by the duplicate database files. See the *Oracle8i Reference* for descriptions of these parameters.
- If your target and duplicate databases reside on different hosts, then you must do one of the following:
 - Move disk copies and backups from the target host to the duplicate host and re-catalog them.
 - Make sure that all backups and copies on the target host are remotely accessible from the duplicate host.
- Specify new filenames or convert target filenames for the datafiles and online redo logs. If you do not specify filenames, RMAN reuses the target datafile names. You *must* use **`nofilenamecheck`** in this case.
- The **`duplicate`** command automatically restores the appropriate backup or copy of each target datafile. If any target datafile does not have a backup, then the duplicate operation will abort.

Keywords and Parameters

<i>database_name</i>	<p>specifies the name of the duplicate database. You must specify the database name because the new database is started but not mounted, so the database name cannot be obtained. The name should match the name in the <code>init.ora</code> file of the duplicate database or Oracle will signal an error when creating the control file.</p> <p>Note: You can use the same database name for the target and duplicate databases since RMAN generates a new DBID for the duplicate database.</p>
logfile <i>logSpec</i>	<p>specifies the online redo logs. The syntax is the same used in the <code>LOGFILE</code> option of the <code>CREATE DATABASE</code> statement.</p> <p>If you do not specify the <code>logfile</code> clause, then RMAN uses <code>LOG_FILE_NAME_CONVERT</code> if it is set. If neither <code>logfile</code> nor <code>LOG_FILE_NAME_CONVERT</code> is specified, RMAN uses the original target redo log filenames for the duplicate files. You must use the <code>nofilenamecheck</code> option in this case.</p> <p>See Also: For more about the <code>CREATE DATABASE</code> statement, see the <i>Oracle8i SQL Reference</i>.</p>

<i>'filename'</i>	specifies the filename of the online redo log.
size <i>integer</i>	specifies the size of the file in kilobytes (K) or megabytes (M). If you omit this parameter, the file must already exist.
reuse	allows Oracle to reuse an existing file. If the file already exists, Oracle verifies that its size matches the value of the size parameter. If the file does not exist, Oracle creates it. If you omit the size parameter, the file must already exist. The reuse option is significant only when used in conjunction with the size parameter. If you omit the size parameter, Oracle expects the file to exist already.
group <i>integer</i> (<i>'filename', ...</i>)	specifies a redo log group containing one or more members. Each filename specified within the parentheses indicates a member of the group.
nofilenamecheck	prevents RMAN from checking whether target datafiles sharing the same names as the duplicated files are in use. The user is responsible for determining that the duplicate operation will not overwrite useful data. This option is necessary when you are creating a duplicate database in a different host that has the same disk configuration, directory structure, and filenames as the host of the target database. For example, imagine a small database that in the <code>/dbs</code> directory of HOST1: <pre> /oracle/dbs/system_prod1.dbf /oracle/dbs/users_prod1.dbf /oracle/dbs/tools_prod1.dbf /oracle/dbs/rbs_prod1.dbf /oracle/dbs/users2_prod1.dbf </pre> Assume that you want to duplicate the database in machine HOST2, which happens to have the same file system <code>/oracle/dbs/*</code> , and you want to use the same filenames in the duplicate as in the primary. In this case, use the nofilenamecheck option to avoid renaming all the files. Because RMAN is not aware of the different hosts, RMAN cannot determine automatically that it should not check the filenames.
skip readonly	excludes the datafiles in read-only tablespaces from the duplicate database. Note: A record for the skipped read-only tablespace still appears in <code>DBA_TABLESPACES</code> . This feature allows you to activate the read-only tablespace later. For example, you can store the read-only tablespace data on a CD-ROM instead of on disk, then mount the CD-ROM later and view the data.

Examples

Setting New Filenames Manually This example assumes that your target database is on HOST1 and you wish to duplicate your database to NEWDB on `host2` with the file structure `/oracle/dbs/*`. Because the filenames in HOST1 are irregularly named and located in various sub-directories, you use **set newname** commands to

re-name the files consistently. The **duplicate** command uses backup sets stored on tape to duplicate the target database to database NEWDB:

```
connect target;
connect catalog rman/rman@rmancat;
connect auxiliary sys/change_on_install@newdb;
run {
    allocate auxiliary channel newdb1 type 'sbt_tape';
    allocate auxiliary channel newdb2 type 'sbt_tape';
    allocate auxiliary channel newdb3 type 'sbt_tape';
    allocate auxiliary channel newdb4 type 'sbt_tape';
    set newname for datafile 1 TO '$ORACLE_HOME/dbs/newdb_data_01.f';
    set newname for datafile 2 TO '$ORACLE_HOME/dbs/newdb_data_02.f';
    set newname for datafile 3 TO '$ORACLE_HOME/dbs/newdb_data_11.f';
    set newname for datafile 4 TO '$ORACLE_HOME/dbs/newdb_data_12.f';
    set newname for datafile 5 TO '$ORACLE_HOME/dbs/newdb_data_21.f';
    set newname for datafile 6 TO '$ORACLE_HOME/dbs/newdb_data_22.f';
    duplicate target database to newdb logfile
        group 1 ('$ORACLE_HOME/dbs/newdb_log_1_1.f',
                '$ORACLE_HOME/dbs/newdb_log_1_2.f') size 200K,
        group 2 ('$ORACLE_HOME/dbs/newdb_log_2_1.f',
                '$ORACLE_HOME/dbs/newdb_log_2_2.f') size 200K reuse;
}
```

Reusing the Target Filenames This example assumes that you are restoring to a new host and that:

- The target host and duplicate host have the same file structure.
- You wish to name the duplicate files exactly like the target database files.
- You are not using a recovery catalog.
- You do not want to duplicate read-only tablespaces.
- You want to prevent RMAN from checking whether files on the target database that have the same names as the duplicated files are in use.

```
connect target
connect auxiliary sys/aux_pwd@newdb
run {
    allocate auxiliary channel ndbnewh1 type disk;
    allocate auxiliary channel ndbnewh2 type disk;
    duplicate target database to ndbnewh
    logfile
        '$ORACLE_HOME/dbs/log_1.f' size 200K,
        '$ORACLE_HOME/dbs/log_2.f' size 200K
    skip readonly
    nofilenamecheck;
}
```

Related Topics

["allocate"](#) on page 11-9

["connect"](#) on page 11-44

["copy"](#) on page 11-48

["set"](#) on page 11-129

["startup"](#) on page 11-142

host

Syntax



Purpose

To invoke an O/S command-line sub-shell from within RMAN.

Requirements

Execute this command at the RMAN prompt or within the braces of a **run** command.

Keywords and Parameters

host	enables you to execute an O/S command. Use this parameter:
	<ul style="list-style-type: none">with a <i>'command'</i>, in which case RMAN runs the command in the specified string and then continues.without a <i>'command'</i>, in which case RMAN displays a command prompt and resumes after you exit the sub-shell.

Examples

Executing an O/S Copy Within RMAN This example shuts down the database, makes a backup of datafile `tbs_01.f` using a media manager, then makes an image copy of the same file on disk using a UNIX command. The database needs to be shut down cleanly to prevent fractured blocks:

```
shutdown immediate;
run {
  allocate channel ch1 type disk;
  allocate channel ch2 type 'sbt_tape';
  backup datafile '$ORACLE_HOME/dbs/tbs_01.f' channel ch2;
  host 'cp $ORACLE_HOME/dbs/tbs_01.f $ORACLE_HOME/dbs/copy/tbs_01.f';
}
```


Hosting to the O/S Within a Copy Job This example makes an image copy of datafile 3, hosts out to the UNIX prompt to check that the copy is in the directory, then resumes the **run** job:

```
RMAN> run {
2> allocate channel c1 type disk;
3> copy datafile 3 to 'df.3';
4> host;
5> release channel c1;
6> }

RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel c1: sid=17 devtype=DISK

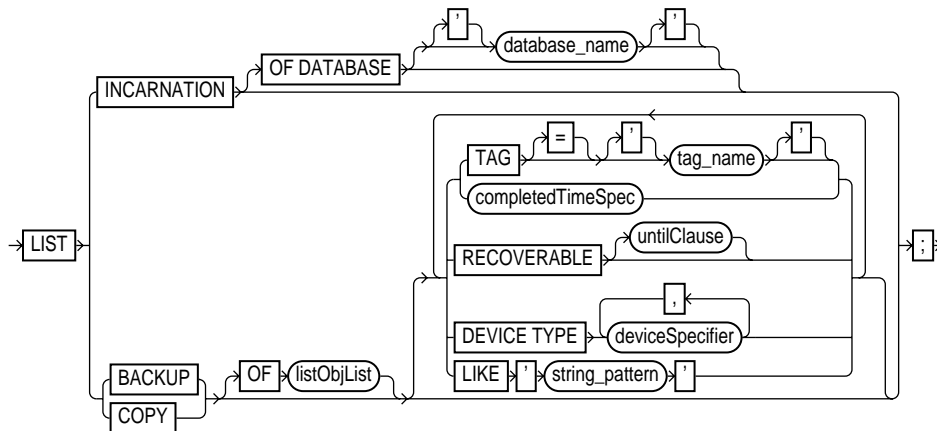
RMAN-03022: compiling command: copy
RMAN-03023: executing command: copy
RMAN-08000: channel c1: copied datafile 3
RMAN-08501: output filename=/oracle/dbs/df.3 recid=102 stamp=352745706
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete

RMAN-03022: compiling command: host
% ls df.3
df.3
% exit
exit
RMAN-06134: host command complete

RMAN-03022: compiling command: release
RMAN-03023: executing command: release
RMAN-08031: released channel: c1
```

list

Syntax



Purpose

To produce a detailed listing of specified backups (either backup sets or media-managed proxy copies) or image copies recorded in the recovery catalog or target control file. RMAN records the output to either standard output or the message log (see "[cmdLine](#)" on page 11-39), but not to both at the same time. Use this command to list:

- Backups or image copies of a specified list of datafiles.
- Backups or image copies of any datafile that is a member of a specified list of tablespaces.
- All backups or image copies of all datafiles in the database, optionally restricted by time, datafile copy filename, device name, recoverability, or tag.
- Backups or copies of archived redo logs with a specified name and/or within a specified range.
- Incarnations of a specified database or of all databases known to the recovery catalog.

See Also: To learn how to make lists and reports, see [Chapter 7, "Generating Lists and Reports with Recovery Manager"](#).

Requirements

- Execute **list** only at the RMAN prompt.
- The **incarnation** option requires the use of a recovery catalog.
- You must be connected to the target database. If you use a recovery catalog, you must also be connected to it.
- The **list** command will not show any records with **deleted** status. To see the records with **deleted** status, query the recovery catalog views. See [Chapter 12, "Recovery Catalog Views"](#).

Keywords and Parameters

incarnation	<p>displays information about the incarnations of a database. See Table 11–8 for an explanation of the column headings of the list incarnation output table.</p> <p>The listing includes the primary keys of all database incarnation records for the specified database name. Use the key in a reset database command to change the incarnation that RMAN considers to be current to a previous incarnation. If you do not specify the of database option, then the command lists all databases registered in the recovery catalog.</p> <p>of database specifies the name of the database. <i>database_name</i></p>
copy	<p>displays information about datafile copies, archived redo logs, and image copies of archived redo logs. By default, list will display copies of all files in the database. Both usable and unusable image copies are included in the output, even those that cannot be restored or are expired or unavailable.</p> <p>See Also: See Table 11–6 and Table 11–7 for an explanation of the column headings of the list copy output tables.</p>
backup	<p>displays information about backups: backup sets, backup pieces, and proxy copies. The output displays a unique key for each. By default, backups of the whole database are listed. Both usable and unusable backups are included in the output, even those that cannot be restored, are expired or unavailable, or are incremental backups that cannot be restored because their parent full backup or copy no longer exists.</p> <p>See Also: See Table 11–2, Table 11–3, Table 11–4, and Table 11–5 for an explanation of the column headings of the list backup output tables. Use the KEY column of the output to obtain the primary key usable in the change and delete expired backupset commands.</p>
of listObjList	<p>restricts the list of objects operated on to the object type specified in the <i>listObjList</i> clause. See "listObjList" on page 11-84. If you do not specify an object, list defaults to of database.</p>
completedTimeSpec	<p>specifies a range of time for completion of the backup or copy. See "completedTimeSpec" on page 11-42.</p>

tag <i>tag_name</i>	restricts the datafile copies and backups by specifying the tag of the copy or backup. If you specify tag , only copies or backups with the specified tag will be listed.
recoverable	<p>specifies only backups or copies of datafiles that are available and can possibly be used in a restore operation. To be a candidate for restore operations a backup must meet these criteria. If the backup is:</p> <ul style="list-style-type: none"> ■ Incremental, then a valid parent must exist to which this incremental can be applied. ■ In a prior incarnation, then there must be no further changes to the files in that incarnation. In other words, the files must be offline and must not have come online again in that incarnation. <p><i>untilClause</i> specifies an end time, SCN, or log sequence number. See "<i>untilClause</i>" on page 11-146.</p>
device type <i>deviceSpecifier</i>	lists only backup sets residing on one of the specified device types (see " <i>deviceSpecifier</i> " on page 11-66). If not specified, all available backup sets will be listed. This option applies only to the list backup command.
like <i>string_pattern</i>	restricts datafile copies and archived redo logs by specifying a filename pattern. The pattern can contain Oracle pattern matching characters '%' and '_'. RMAN lists only files whose name matches the pattern.

List Output

The status information that appears in the output is shown in Table 11-2:

Table 11-2 List of Backup Sets

Column	Indicates
KEY	<p>a unique key identifying this backup set.</p> <p>Note: If the target database control file is used instead of the recovery catalog, then this field is a unique identifier that specifies this backup set in the target database control file (and is equal to the RECID, which serves this purpose when a recovery catalog is not used). Use this key in a change ... backupset statement to change the status of the backup set.</p>
RECID	<p>when combined with the STAMP column, a unique key that identifies this backup set in the target database control file. The RECID will be invalid when a new control file record occupies the space used by the old record. For this reason, issue resync commands often so that the new records are copied to the recovery catalog as soon as possible.</p>
STAMP	<p>when combined with the RECID column, a unique key that identifies this backup set in the target database control file.</p>

Table 11–2 List of Backup Sets

LV	the level of the backup: NULL for non-incrementals, level 0-4 for incrementals.
SET STAMP	<p>when combined with the SET COUNT column, a unique key that identifies this backup set in the target database control file. Use these values to access the control file records in the V\$BACKUP_SET, V\$BACKUP_PIECE, V\$BACKUP_DATAFILE, and V\$BACKUP_REDOLOG views.</p> <p>The SET STAMP value is valid at all times, both in the control file (when not using a recovery catalog) and when using a recovery catalog. SET STAMP values are never entered by a user because they are part of a two-value key. Oracle World Wide Support may request this value if your database requires recovery when no recovery catalog exists and control file records are gone.</p> <p>See Also: For more information about data dictionary views, see the <i>Oracle8i Reference</i>.</p>
SET COUNT	<p>when combined with the SET STAMP column, a unique key that identifies this backup set in the target database control file. Use these values to access the control file records in the V\$BACKUP_SET, V\$BACKUP_PIECE, V\$BACKUP_DATAFILE, and V\$BACKUP_REDOLOG views.</p> <p>The SET COUNT value is valid at all times, both in the control file (when not using a recovery catalog) and when using a recovery catalog. SET COUNT values are never entered by a user because they are part of a two-value key. Oracle World Wide Support may request this value if your database requires recovery when no recovery catalog exists and control file records are gone.</p>
COMPLETION TIME	the date and time that the backup set completed. Note that the format of this field depends on the NLS_LANG and NLS_DATE_FORMAT environment settings.

Table 11–3 List of Backup Pieces

Column	Indicates
KEY	a unique identifier for this backup piece in the recovery catalog or target database control file. Note: The values for KEY in the recovery catalog and the control file are different.
PC#	the piece number of this backup piece within the backup set.
CP#	the copy number of this backup piece in a duplexed backup. For example, if set duplex = 4 , then CP# will range from 1 to 4. Note: If the backup is not duplexed, then CP# = 1.
STATUS	the backup piece status: AVAILABLE, UNAVAILABLE, or EXPIRED (see the change command for an explanation of each status).
COMPLETION TIME	the date and time when the piece was created.
PIECE NAME	the name of the backup piece.

Table 11–4 Controlfile Included

Column	Indicates
CKP SCN	the SCN of the backup control file checkpoint. All database changes recorded in the redo records before the specified SCN are reflected in this control file.
CKP TIME	the time of the backup control file checkpoint. All database changes recorded in the redo records before the specified time are reflected in this control file.

Table 11–5 List of Datafiles Included

Column	Indicates
FILE	the number of the file that was backed up.
NAME	the location where this file would be restored now if it were restored from this backup set and no set newname command (see " set_run_option " on page 11-133) was entered.
LV	the level of the backup: NULL for non-incrementals, level 0-4 for incrementals.
TYPE	whether the backup was FULL or INCR (incremental).

Table 11–5 List of Datafiles Included

CKP SCN	the checkpoint of the datafile at the time it was backed up. All database changes prior to the SCN have been written to the file; changes after the specified SCN have not been written to the file.
CKP TIME	the checkpoint of the datafile at the time it was backed up. All database changes prior to the time have been written to the file; changes after the specified time have not been written to the file.

Table 11–6 List of Datafile Copies

Column	Indicates
KEY	the unique identifier for the datafile copy. Use this value in a change command to alter the status of the datafile copy. Note: The values for KEY in the recovery catalog and the control file are different.
FILE	the file number of the original datafile from which this copy was made.
S	the backup piece status: AVAILABLE, UNAVAILABLE, or EXPIRED (see the change command for an explanation of each status).
COMPLETION TIME	the date and time that the copy completed. Note that the value of this field is sensitive to the NLS_LANG and NLS_DATE_FORMAT environment variables.
CKP SCN	the checkpoint of this datafile when it was copied. All database changes prior to the SCN have been written to this file.
CKP TIME	the checkpoint of this datafile when it was copied. All database changes prior to the time have been written to this file.
NAME	the filename of the datafile copy.

Table 11–7 List of Archived Log Copies

Column	Indicates
KEY	the unique identifier for this archived redo log copy. Use this value in a change command to alter the status of the copy. Note: The values for KEY in the recovery catalog and the control file are different.
THRD	the redo log thread number.
SEQ	the log sequence number.

Table 11–7 List of Archived Log Copies

COMPLETION TIME	the date and time that the copy completed. Note that the value of this field is sensitive to the NLS_LANG and NLS_DATE_FORMAT environment variables.
NAME	the filename of the archived redo log copy.

Table 11–8 List of Database Incarnations

Column	Indicates
DB KEY	when combined with the INC KEY, the unique key by which RMAN identifies the database incarnation in the recovery catalog. Use this key to unregister a database, i.e., delete all the rows associated with that database from the recovery catalog.
INC KEY	when combined with DB KEY, the unique key by which RMAN identifies the database incarnation in the recovery catalog. Use this key in the reset database to incarnation command, which you must use if you want to recover your database to a time prior to the most recent RESETLOGS.
DB NAME	the database name as listed in the DB_NAME parameter.
DB ID	the database identification number, which Oracle generates automatically at database creation.
CUR	whether the incarnation is the current incarnation of the database.
RESET SCN	the SCN at which the incarnation was created.
RESET TIME	the time at which the incarnation was created.

Examples

Listing Copies The following example lists datafile copies and archived redo logs recorded in the recovery catalog:

```
list copy of database archivelog all;
```

```
List of Datafile Copies
```

```
Key      File S Completion time Ckp SCN    Ckp time   Name
-----
1262    1    A 18-AUG-98          219859    14-AUG-98  /vobs/oracle/dbs/copy/tbs_01.f
```

```
List of Archived Log Copies
```

```
Key      Thrd Seq    S Completion time Name
-----
789     1    1        A 14-JUL-98          /vobs/oracle/work/arc_dest/arcr_1_1.arc
```



```

790    1    2    A 11-AUG-98    /vobs/oracle/work/arc_dest/arcr_1_2.arc
791    1    3    A 12-AUG-98    /vobs/oracle/work/arc_dest/arcr_1_3.arc

```

Listing Backups The following example lists backups of two datafiles recorded in the recovery catalog:

```
list backup of datafile '/oracle/dbs/tbs_01.f', '/oracle/dbs/tbs_02.f';
```

List of Backup Sets

Key	Recid	Stamp	LV	Set Stamp	Set Count	Completion Time
1174	12	341344528	0	341344502	16	14-SEP-98

List of Backup Pieces

Key	Pc#	Cp#	Status	Completion Time	Piece Name
1176	1	1	AVAILABLE	14-AUG-98	/vobs/oracle/dbs/0ga5h07m_1_1

Controlfile Included

Ckp SCN	Ckp time
219857	14-AUG-98

List of Datafiles Included

File Name	LV	Type	Ckp SCN	Ckp Time
1 /oracle/dbs/tbs_01.f	0	Full	199843	14-AUG-98
2 /oracle/dbs/tbs_02.f	0	Full	199843	14-AUG-98

Listing Database Incarnations The following example lists all database incarnations recorded in the recovery catalog:

```
list incarnation;
```

List of Database Incarnations

DB Key	Inc Key	DB Name	DB ID	CUR	Reset SCN	Reset Time
1	2	PROD1	1224038686	NO	1	02-JUL-98
1	582	PROD1	1224038686	YES	59727	10-JUL-98

Related Topics

["crosscheck"](#) on page 11-57

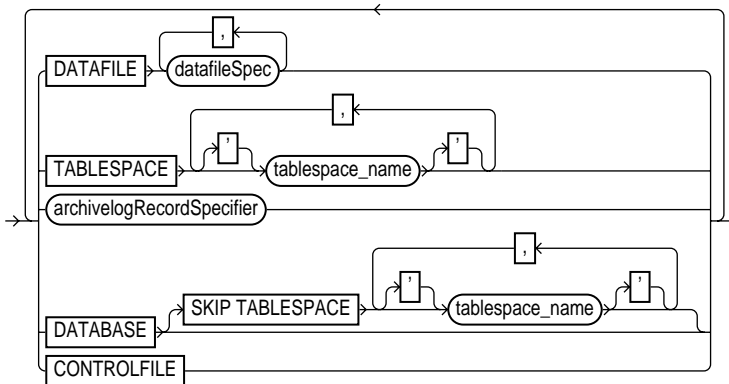
["listObjList"](#) on page 11-84

["report"](#) on page 11-102

["validate"](#) on page 11-150

listObjList

Syntax



Purpose

A sub-clause used to specify database files and archived redo logs.

Requirements

Use this clause in the following commands:

- **list**
- **crosscheck**
- **deleteExpired**

Keywords and Parameters

datafile <i>datafileSpec</i>	specifies datafiles by filename for file number. The clause specifies datafile image copies or backup sets that contain at least one of the datafiles. See " datafileSpec " on page 11-60.
tablespace <i>tablespace_name</i>	specifies tablespace names. The clause specifies datafile image copies or backup sets that contain at least one of the datafile from the specified tablespace.
<i>archiveLogRecord- Specifier</i>	specifies a range of archived redo logs. See " archiveLogRecordSpecifier " on page 11-17.

database	specifies backup sets or image copies of all files in the current database.
skip tablespace <i>tablespace_name</i>	omits the specified tablespaces from the database specification.
controlfile	specifies the current control file.

Examples

Listing Datafile Copies The following command lists image copies of all the files in the database, skipping the TEMP tablespace:

```
list copy of database skip tablespace temp;
```

Crosschecking Archived Redo Logs The following example queries the media manager for the status of archived redo log backup sets created in the last 90 days:

```
allocate channel for maintenance type 'sbt_tape';
crosscheck
  backup
  of archivelog
  from time 'SYSDATE-90';
release channel;
```

Deleting Expired Control File Backup Sets The following command deletes expired backups of the control file:

```
delete expired backup of controlfile;
```

Related Topics

["archivelogRecordSpecifier"](#) on page 11-17

["crosscheck"](#) on page 11-57

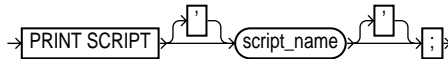
["datafileSpec"](#) on page 11-60

["deleteExpired"](#) on page 11-63

["list"](#) on page 11-76

printScript

Syntax



Purpose

To print a stored script to standard output or the RMAN message log. Specify the log filename with the **log** argument at the command line. If you do not specify this argument, Recovery Manager writes message output to standard output.

Note: You can also display the individual lines of your stored scripts by querying the RC_STORED_SCRIPT_LINE recovery catalog view.

Requirements

- Use this command only at the RMAN prompt.
- You must be using a recovery catalog.

Keywords and Parameters

script_name prints a stored script with the specified name to standard output or a message log. To obtain a listing of all stored scripts, use SQL*Plus connect to the recovery catalog as the catalog owner and issue the following query:

```
select * from rc_stored_script;
```

Note: To run the script, use **execute script** within the braces of the **run** command.

See Also: To learn about RC_STORED_SCRIPT, see "[RC_STORED_SCRIPT](#)" on page 12-25.

Examples

Printing a Script to the Message Log This example connects to the target database PROD1 and the recovery catalog database RCAT, and directs the RMAN log output to a message log file. It then creates the `backup_db` script and prints it to `rman_log`. Finally, it executes the script:

```
rman target sys/change_on_install@prod1 catalog rman/rman@rcat log rman_log
create script backup_db {
    allocate channel d1 type disk;
    backup database;
}
print script backup_db;

run{ execute script backup_db;};
```

Printing a Script to the Screen This example prints a stored script to the screen:

```
print script tbs1_b;

RMAN-03027: printing stored script: tbs1_b
{
allocate channel ch1 type disk;
backup tablespace tbs1;
}
```

Related Topics

["cmdLine"](#) on page 11-39

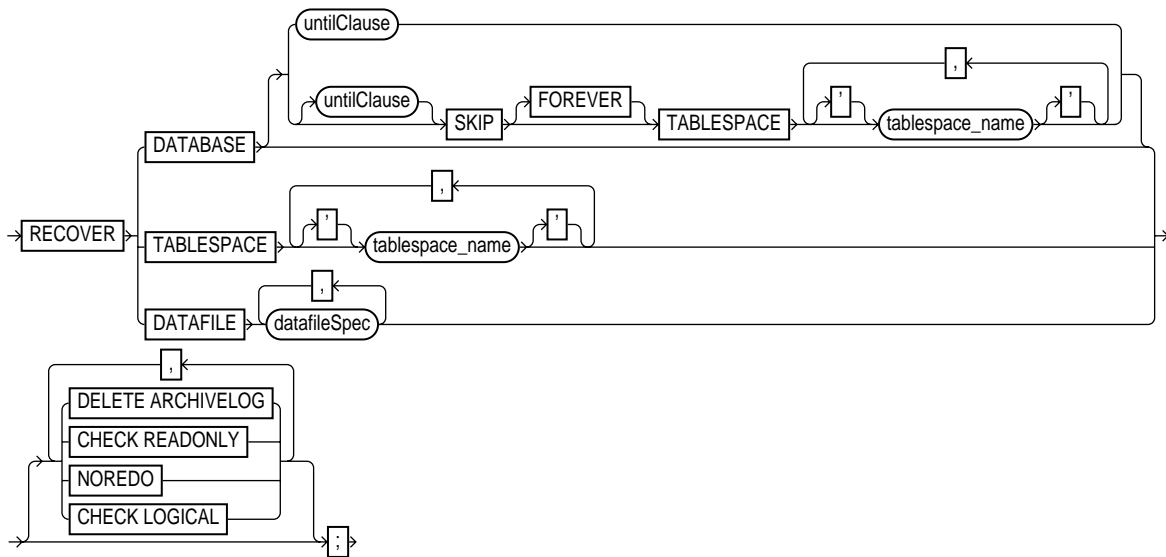
["createScript"](#) on page 11-54

["deleteScript"](#) on page 11-65

["run"](#) on page 11-124

recover

Syntax



Purpose

To apply redo logs or incremental backups to one or more restored datafiles in order to update them to a specified time.

RMAN uses online redo records and restores backup sets of archived redo logs as needed to perform the media recovery. If RMAN has a choice between applying an incremental backup or applying redo, then it always chooses the incremental backup. If overlapping levels of incremental backup are available, RMAN automatically chooses the one covering the longest period of time.

See Also: To learn how to recover datafiles, see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#).

Requirements

- Execute this command only within the braces of a **run** command.
- You can use this command without a recovery catalog, but only if control file recovery is not required. RMAN cannot operate when neither the recovery catalog nor the target database control file are available.
- For datafile and tablespace recovery, the target database must be mounted. If it is open, then the datafiles or tablespaces to be recovered must be offline. For database recovery, the database must be mounted but not open.
- At least one **allocate channel** command must precede **recover** unless you do not need to restore archived redo log or incremental datafile backup sets.
- Allocate the appropriate type of device for the backups that you need to restore. If the appropriate type of device is not available, then the **recover** command aborts.
- Only the current datafiles may be recovered or have incremental backups applied to them.
- Typically, you should enter a **set until** command before both the **restore** and **recover** command. If you specify a **set until** command *after* a **restore** and *before* a **recover**, you may not be able to perform media recovery on the database to the time required because the restored files may have timestamps later than the specified time.
- The **recover database** command will not recover any files that are offline normal or read-only at the point in time to which the files are being recovered. RMAN omits offline normal files with no further checking. If **check readonly** is specified, then RMAN checks each read-only file on disk to ensure that it is already current at the desired point in time. If **check readonly** is not specified, then RMAN omits read-only files.
- Open with the **RESETLOGS** option after incomplete recovery or recovery with a backup control file.

Keywords and Parameters

database	specifies that the entire database is to be recovered. You can specify an optional <i>untilClause</i> that causes the recovery to stop when the specified condition has been reached.
<i>untilClause</i>	specifies a non-current time, SCN, or log sequence number for the recover command. See " untilClause " on page 11-146.
skip [forever] tablespace <i>tablespace_name</i>	<p>lists tablespaces that should not be recovered, which is useful for avoiding recovery of tablespaces containing only temporary data or for postponing recovery of some tablespaces. The skip clause takes the datafiles in the specified tablespaces offline before starting media recovery. These files are left offline after the media recovery is complete.</p> <p>If you perform incomplete recovery, then skip is not allowed. Instead, use skip forever, with the intention of dropping the skipped tablespaces after opening the database with the RESETLOGS option. The skip forever clause causes RMAN to take the datafiles offline using the DROP option. Only use skip forever when the specified tablespaces will be dropped after opening the database.</p>
tablespace <i>tablespace_name</i>	specifies tablespaces by tablespace name.
datafile <i>datafileSpec</i>	<p>specifies a list of one or more datafiles to recover. Specify datafiles by filename using a quoted string or absolute datafile number using an <i>integer</i> (see "datafileSpec" on page 11-60).</p> <p>If you are using the control file as the exclusive repository for RMAN metadata, then the filename must be the name of the datafile as known in the control file.</p> <p>If you are using a recovery catalog, then the filename of the datafile must be the most recent name recorded in the catalog. For example, assume that a datafile was renamed in the control file. The database then crashes before you can resynchronize the catalog. Specify the old name of the datafile in the recover command, since this is the name recorded in the catalog.</p>
delete archivelog	deletes restored archived logs that are no longer needed. RMAN does not delete archived logs that were already on disk before the restore command started.
check readonly	checks the headers of read-only files to ensure that they are current before omitting them from the recovery.
noredo	suppresses the application of redo logs—only incremental backups are applied. This option is intended for recovery of NOARCHIVELOG databases using incremental backups. If you do not specify noredo when recovering a NOARCHIVELOG database, Oracle aborts a recovery and issues an error.

Examples

Recovering a Tablespace in an Open Database The following example takes tablespace TBS_1 offline, restores and recovers it, then brings it back online:

```
run {
  allocate channel dev1 type 'sbt_tape';
  sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE";
  restore tablespace tbs_1;
  recover tablespace tbs_1;
  sql "ALTER TABLESPACE tbs_1 ONLINE";
}
```

Recovering Datafiles Restored to New Locations The following example allocates one disk channel and one media management channel to use datafile copies on disk and backups on tape, and restores one of the datafiles in tablespace TBS_1 to a different location:

```
run {
  allocate channel dev1 type disk;
  allocate channel dev2 type 'sbt_tape';
  sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE";
  set newname for datafile 'disk7/oracle/tbs11.f'
    to 'disk9/oracle/tbs11.f';
  restore tablespace tbs_1;
  switch datafile all;
  recover tablespace tbs_1;
  sql "ALTER TABLESPACE tbs_1 ONLINE";
}
```

Performing Incomplete Recovery Using a Backup Control File Assume that both the database and archived redo log 1234 were lost due to a disk crash. Because you do not have incremental backups, you need to recover the database using available archived redo logs. There is no need to restore tablespace READONLY1 because it has not changed since log 1234.

```
run {
  # Recover database until log sequence 1234
  allocate channel dev1 type disk;
  allocate channel dev2 type 'sbt_tape';
  set until logseq 1234 thread 1;
  restore controlfile to '/vobs/oracle/dbs/cf1.f' ;
  # Because you specified a restore destination, you must manually replicate the
  # control file. The restore command replicates automatically when no destination is
  # specified.
  replicate controlfile from '/vobs/oracle/dbs/cf1.f';
  alter database mount;
```

```
restore database skip tablespace temp1, readonly1;
recover database skip forever tablespace temp1;
sql "ALTER DATABASE OPEN RESETLOGS";
sql "DROP TABLESPACE temp1";
sql "CREATE TABLESPACE temp1 DATAFILE '/vobs/oracle/dbs/temp1.f' SIZE 10M";
release channel dev1;
release channel dev2;
}
```

Related Topics

["allocate" on page 11-9](#)

["set_run_option" on page 11-133](#)

["restore" on page 11-112](#)

["untilClause" on page 11-146](#)

register

Syntax

```
→ REGISTER DATABASE <id>
```

Purpose

To register the target database in the recovery catalog so that RMAN can access it. RMAN obtains all information it needs to register the target database from the target database itself.

Requirements

- Execute this command only at the RMAN prompt.
- A database can only be registered once.
- You must be using a recovery catalog.
- The target database must be mounted.
- The **register database** command fails when RMAN detects duplicate database identifiers. This situation can arise when databases are created by copying files from an existing database rather than using the **duplicate** command.

If this failure occurs, you can create a second recovery catalog in another user's schema by executing **create catalog** (see "[createCatalog](#)" on page 11-52) using a different user id. Then, register the database with the duplicate database identifier into the newly created recovery catalog in the new schema.

Note: If you are using RMAN with different target databases that have the same database name and identifier, be extremely careful to always specify the correct recovery catalog schema when invoking Recovery Manager.

Examples

Registering a Database This example registers the target database, catalogs an existing datafile copy, then opens the database for use:

```
connect target / catalog rman/rman@rcat;  
startup mount;  
register database;  
catalog datafilecopy '/vobs/oracle/dbs/foo.f';  
sql 'alter database open';
```

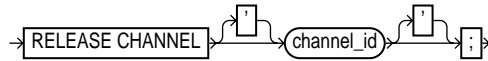
Related Topics

["catalog"](#) on page 11-32

["duplicate"](#) on page 11-69

release

Syntax



Purpose

To release a channel while maintaining the connection to the target database instance. Specify the channel name with the same identifier used in the **allocate channel** command. This command is optional; RMAN automatically releases all channels allocated when the **run** command terminates.

Requirements

Execute this command only within the braces of a **run** command.

Keywords and Parameters

<i>channel_id</i>	specifies the channel id used in the allocate channel command.
-------------------	---

Examples

Releasing a Channel This example makes three identical backup sets of `datafile 1` to tape, then releases the tape channel. RMAN then makes three identical backups of `datafile 2` to disk and then releases the disk channel:

```

run {
  set duplex=3;
  allocate channel ch1 type 'SBT_TAPE';
  allocate channel ch2 type disk;
  backup channel ch1 datafile 1;
  release channel ch1;
  backup datafile 2;
}
  
```

Related Topics

["allocate" on page 11-9](#)

releaseForMaint

Syntax

```
→ RELEASE CHANNEL <n> ; >
```

Purpose

To release a sequential I/O device specified in an **allocate channel** command with the **for delete** or **for maintenance** options. Note that maintenance channels are unaffected by **allocate channel** and **release channel** command issued within a **run** command.

Requirements

- Execute this command only at the RMAN prompt.
- You must have a maintenance channel allocated in order to release it.

Examples

Releasing a Maintenance Channel After a Delete Operation This example allocates and then releases a maintenance channel to the media manager:

```
allocate channel for delete type 'sbt_tape';
change backuppiece 100 delete;
run {
    allocate channel ch1 type disk;
    backup datafile 1;
    release channel ch1; # releases run channel
}
release channel; # releases maintenance channel
```

Related Topics

["allocateForMaint"](#) on page 11-13

replaceScript

Syntax



Purpose

To replace an existing script stored in the recovery catalog. If the script does not exist, **replace script** creates it.

The stored script feature is provided primarily to provide a common repository for frequently executed collections of RMAN commands: use any command legal within a **run** command in the script. The script is not executed immediately; use the **execute script** command (see "[run](#)" on page 11-124) to run it.

Requirements

- Execute **replace script** only at the RMAN prompt.
- You must be using a recovery catalog.

Keywords and Parameters

For descriptions of the individual commands that you can use in a stored script, see the appropriate entry, e.g., "[backup](#)" on page 11-21. For information about the @ and @@ arguments, see "[createScript](#)" on page 11-54. For information about the **execute script** command, see "[run](#)" on page 11-124.

replace script
script_name

replaces the specified stored script with the new commands. The statements allowable within the parentheses of the **replace script 'filename' (...)** command are the same allowable within the **run** command.

To obtain a listing of all stored scripts, use SQL*Plus to connect to the recovery catalog database as the catalog owner and issue the following query:

```
select * from rc_stored_script;
```

Note: To run the script, issue **execute script** within the braces of the **run** command.

See Also: For more information about RC_STORED_SCRIPT, see "[RC_STORED_SCRIPT](#)" on page 12-25.

Examples

Replacing a Script This example creates a script called `backup_full`, replaces it with a different script, and then executes it:

```
create script backup_full {
  allocate channel ch1 type 'SBT_TAPE';
  allocate channel ch2 type 'SBT_TAPE';
  allocate channel ch3 type 'SBT_TAPE';
  backup database;
}
```



```
replace script backup_full {  
    allocate channel ch1 type disk;  
    backup database;  
}  
run { execute script backup_full; }
```

Related Topics

["createScript" on page 11-54](#)

["deleteScript" on page 11-65](#)

["printScript" on page 11-86](#)

["run" on page 11-124](#)

replicate

Syntax

```
→ REPLICATE CONTROLFILE FROM 'filename' ;
```

Purpose

To copy a control file to the locations specified in the CONTROL_FILES initialization parameter of the target database.

After restoring the control file, you can use the **replicate controlfile** statement to prepare the database for mounting. This operation is equivalent to executing multiple **copy controlfile** statements.

Note: The **restore** command will automatically replicate the control file to all CONTROL_FILES locations if no restore destination is specified.

Requirements

- Execute **replicate controlfile** only within the braces of a **run** command.
- At least one **allocate channel** statement specifying the **type disk** option must precede a **replicate controlfile** statement.

Keywords and Parameters

<i>'filename'</i>	specifies the location of the control file to be replicated. For example, if you restore a control file backup to <code>/oracle/temp/cf.bak</code> , then you would also specify this filename in the replicate command.
-------------------	---

Examples

Replicating a Restored Control File This example restores a control file and then replicates it:

```
startup nomount;
run {
  set until time 'Jun 18 1998 16:32:36';
  allocate channel chl type disk;
  # restore a backup controlfile to a temporary location.
  restore controlfile to '/tmp/cf.tmp';
  replicate controlfile from '/tmp/cf.tmp';
  startup force mount;
}
```

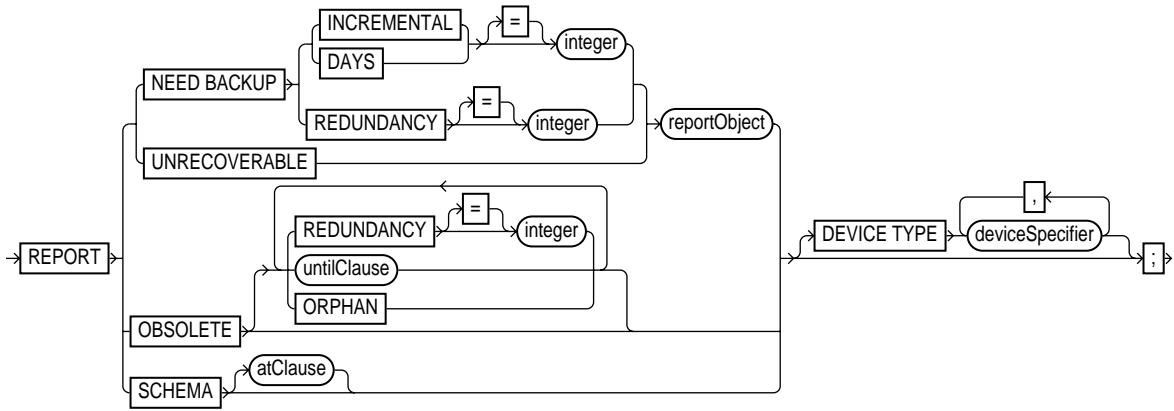
Related Topics

["copy" on page 11-48](#)

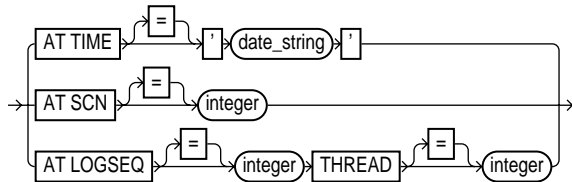
["restore" on page 11-112](#)

report

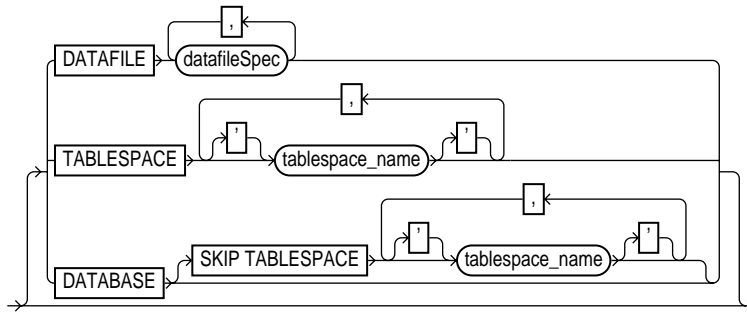
Syntax



atClause ::=



reportObject::=



Purpose

To perform detailed analyses of the RMAN metadata. Oracle writes the output from the **report** command to standard output or the message log file (see ["connect"](#) on page 11-44).

Use the **report** command to answer questions such as the following:

- Which files need a backup?
- Which files have not had a backup in a while?
- Which files are not recoverable due to unrecoverable operations?
- Which backup files can be deleted?
- What was the physical schema of the database at a previous time?

Requirements

- Execute this command only at the RMAN prompt.
- You must use a recovery catalog when issuing a **report schema** command with the **at time**, **at scn**, or **at logseq** options. Otherwise, a recovery catalog is not required for the **report** command.

Keywords and Parameters

need backup	lists all datafiles in need of a new backup. The report assumes that you will use the most recent backup for restore operations.
incremental <i>integer</i>	<p>specifies a threshold number of incremental backups. If complete recovery of a datafile requires more than the specified number of incremental backups, then the datafile requires a new full backup. The report command, like the recover command, uses the lowest level of incremental backup whenever there is a choice. This is the same strategy that RMAN would use if the file were actually being recovered by the recover command.</p> <p>Note: Files for which no backups exist will not appear in this list: issue the report need backup redundancy command to display them.</p>
days <i>integer</i>	<p>specifies a threshold number of days of redo log files that need application during recovery of this file. For example, report need backup days 7 database shows the datafiles whose recovery requires more than one week's worth of archived redo logs.</p> <p>If the target database control file is mounted and current, RMAN makes the following optimizations to this report:</p> <ul style="list-style-type: none"> ■ Files that are offline and whose most recent backup contains all changes to the file will not be included. ■ Files that were offline and are now online, and whose most recent backup contains all changes up to the offline time, will only be reported if they have been online for more than the specified number of days.
redundancy <i>integer</i>	specifies the minimum number of backups or copies that must exist for a datafile to be considered <i>not</i> in need of a backup. In other words, a datafile needs a backup if there are fewer than <i>integer</i> backups or copies of this file. For example, redundancy 2 means that if there are fewer than two copies or backups of a datafile, then it needs a new backup.
unrecoverable	<p>lists all unrecoverable datafiles. A datafile is considered unrecoverable if an unrecoverable operation has been performed against an object residing in the datafile since the last backup of the datafile.</p> <p>Note: The non-existence of any backup of a datafile is not sufficient reason to consider it unrecoverable. Such datafiles can be recovered through the use of the CREATE DATAFILE command, provided that redo logs starting from when the file was created still exist.</p>
<i>reportObject</i> clause	specifies the datafiles to be included in the report. The report can include the entire database (optionally skipping certain tablespaces), a list of tablespaces, or a list of datafiles.
datafile <i>datafileSpec</i>	lists the specified datafiles. RMAN reports on backups or datafile copies that contain at least one of the specified datafiles.

tablespace <i>tablespace_name</i>	lists datafiles in the specified tablespace. RMAN reports on backups or datafile copies that include at least one datafile from a specified tablespace.
database	lists backups or datafile copies of all files in the current database.
skip tablespace <i>tablespace_name</i>	excludes the specified tablespaces from the database specification.
obsolete	lists full backups and datafile copies recorded in the RMAN metadata that can be deleted because they are no longer needed. If you do not specify further parameters, redundancy defaults to 1. If you use this option in conjunction with device type , RMAN only considers backups and copies on the specified type.
redundancy <i>integer</i>	specifies the minimum level of redundancy considered necessary for a backup or copy to be obsolete. A datafile copy is obsolete if there are at least <i>integer</i> more recent backups or image copies of this file; a datafile backup set is obsolete if there are at least <i>integer</i> more recent backups or image copies of each file contained in the backup set. For example, redundancy 2 means that there must be at least two more recent backups or copies of a datafile for any other backup or copy to be obsolete.
<i>untilClause</i>	specifies that no backup or copy will be considered obsolete if there are at least <i>n</i> (where <i>n</i> is the value for redundancy) backups or copies that are more recent but do not contain changes later than the specified time, SCN, or log sequence number. For example, obsolete redundancy 2 until 'SYSDATE-7' means that a backup or copy is obsolete if there are at least two backups or copies that are more recent and those copies were checkpointed more than a week ago. This clause is useful if the database must be recoverable to non-current time, SCN, or log sequence number. See " untilClause " on page 11-146.
orphan	specifies as obsolete those backups and copies that are unusable because they belong to incarnations of the database that are not direct ancestors of the current incarnation. For an explanation of orphaned backups, see " Reporting on Orphaned Backups " on page 4-23.
schema	lists the names of all datafiles and tablespaces at the specified point in time.
<i>atClause</i>	specifies a point in time as a time, an SCN, or a log sequence number.
at time <i>date_string</i>	specifies a date. The NLS_LANG and NLS_DATE_FORMAT environment variables specify the format for the time.
at scn <i>integer</i>	specifies an SCN.
at logseq <i>integer</i>	specifies a log sequence number for a specified redo thread. The integer indicates the time when the specified log and thread were first opened.
device type <i>deviceSpecifier</i>	specifies the type of storage device. RMAN only considers backups and copies available on the specified device for its report. See " deviceSpecifier " on page 11-66.

Report Output

The fields in each report are described below:

Table 11–9 Report of Database Schema

Column	Indicates
FILE	the absolute datafile number.
K-BYTES	the size of the file in kilobytes.
TABLESPACE	the tablespace name.
RB SEGS	YES if rollback segments exist in the tablespace and NO if they do not (only if connected to the recovery catalog). If RMAN is not connected to the catalog, then '***' is displayed.
NAME	the filename of the datafile.

Table 11–10 Report of Obsolete Backups and Copies

Column	Indicates
TYPE	whether the object is a backup set, backup piece, proxy copy, or datafile copy.
KEY	a unique key that identifies this backup in the target database control file.
COMPLETION TIME	the time that the backup or copy completed.
FILENAME/HANDLE	the filename or media handle of the backup or datafile copy.

Table 11–11 Report of Files that Need Backup Due to Unrecoverable Operations

Column	Indicates
FILE	the absolute number of the datafile that needs a new backup due to unrecoverable operations.
TYPE OF BACKUP REQUIRED	FULL or INCREMENTAL, depending on which type of backup is necessary to ensure the recoverability of all of the data in this file. If FULL, then create a full backup, level-0 backup, or a datafile copy. If INCREMENTAL, then a full or incremental backup will also suffice.
NAME	the name of the datafile.

Table 11–12 Report of Files with Less Than *n* Redundant Backups

Column	Indicates
FILE	the absolute datafile number of a datafile with less than <i>n</i> redundant backups.
#BKPS	the number of backups that exist for this file.
NAME	the name of the file.

Table 11–13 Report of Files Whose Recovery Needs More Than *n* Days of Archived Logs

Column	Indicates
FILE	the absolute file number of a datafile that requires more than <i>n</i> days of archived redo logs for recovery.
DAYS	the number of days of archived redo data required for recovery.
NAME	the name of the datafile.

Table 11–14 Report of Files That Need More than *n* Incrementals During Recovery

Column	Indicates
FILE	the absolute file number of a datafile that requires more than <i>n</i> incrementals for complete recovery.
INCREMENTALS	the number of incremental backups required for complete recovery.
NAME	the name of the datafile.

Examples

Reporting Database Schema This example reports the names of all datafiles and tablespaces in the database one week ago:

```
report schema at time 'SYSDATE-7';
```

```
Report of database schema
```

```
File K-bytes  Tablespace          RB segs Name
-----
```

1	47104	SYSTEM	YES	/vobs/oracle/dbs/tbs_01.f
2	978	SYSTEM	YES	/vobs/oracle/dbs/tbs_02.f
3	978	TBS_1	NO	/vobs/oracle/dbs/tbs_11.f
4	978	TBS_1	NO	/vobs/oracle/dbs/tbs_12.f

5	978 TBS_2	NO	/vobs/oracle/dbs/tbs_21.f
6	978 TBS_2	NO	/vobs/oracle/dbs/tbs_22.f
7	500 TBS_3	NO	/vobs/oracle/dbs/tbs_31.f
8	500 TBS_3	NO	/vobs/oracle/dbs/tbs_32.f
9	5120 SYSTEM	YES	/vobs/oracle/dbs/tbs_03.f

Reporting Datafiles Needing Incremental Backups This example reports all datafiles in the database that require the application of five or more incremental backups to be recovered to their current state:

```
report need backup incremental 5 database;
```

Report of files that need more than 5 incrementals during recovery

File Incrementals Name

```
-----
1  9      /vobs/oracle/dbs/tbs_01.f
2  9      /vobs/oracle/dbs/tbs_02.f
3  9      /vobs/oracle/dbs/tbs_11.f
4  9      /vobs/oracle/dbs/tbs_12.f
5  9      /vobs/oracle/dbs/tbs_21.f
6  9      /vobs/oracle/dbs/tbs_22.f
7  9      /vobs/oracle/dbs/tbs_23.f
8  9      /vobs/oracle/dbs/tbs_03.f
```

Reporting Datafiles Needing Backups The following example reports all datafiles from tablespace SYSTEM that will need more than two days of archived redo logs to be applied during recovery after being restored from the most recent backup:

```
report need backup days 2 tablespace system;
```

Report of files whose recovery needs more than 2 days of archived logs

File Days Name

```
-----
1  3      /vobs/oracle/dbs/tbs_01.f
2  3      /vobs/oracle/dbs/tbs_02.f
16 3      /vobs/oracle/dbs/tbs_03.f
```

Reporting Unrecoverable Datafiles The following example reports all datafiles that cannot be recovered from existing backups because redo may be missing:

```
report unrecoverable;
```

Report of files that need backup due to unrecoverable operations

File Type of Backup Required Name

```
-----
4  FULL      /vobs/oracle/dbs/tbs_12.f
```

Reporting Obsolete Backups and Copies The following example reports obsolete backups and copies with a redundancy of 1:

```
report obsolete;
```

```
Report of obsolete backups and copies
```

Type	Key	Completion Time	Filename/Handle
Backup Set	836	04-DEC-98	
Backup Piece	839	04-DEC-98	/vobs/oracle/dbs/05aetj6b_1_1
Backup Set	807	04-DEC-98	
Backup Piece	810	04-DEC-98	/vobs/oracle/dbs/03aetj1f_1_1
Backup Set	835	04-DEC-98	
Backup Piece	838	04-DEC-98	/vobs/oracle/dbs/04aetj6b_1_1

Related Topics

["list"](#) on page 11-76

["untilClause"](#) on page 11-146

["validate"](#) on page 11-150

reset

Syntax



Purpose

To create a new database incarnation record in the recovery catalog. RMAN considers the new incarnation as the current incarnation of the database. All subsequent backups and redo log archiving operations performed by the target database will be associated with the new database incarnation.

Requirements

- Execute **reset database** only at the RMAN prompt.
- You must be using a recovery catalog.
- You must issue a **reset database** command before you can use RMAN with a target database that has been opened with the RESETLOGS option. If you do not, then RMAN refuses to access the recovery catalog, since RMAN cannot distinguish between a RESETLOGS and an accidental restore of an old control file. The **reset database** command gives confirmation to RMAN that you issued a RESETLOGS command.

Keywords and Parameters

to incarnation <i>primary_key</i>	changes the incarnation that RMAN considers to be current to an older incarnation. This option is useful in the rare circumstance in which you want to undo the effects of a RESETLOGS by restoring backups of a prior incarnation of the database. Specify the primary key of the DBINC record for the database incarnation. Obtain the key value using the list incarnation of database command. After you issue the reset database to incarnation command, issue restore and recover commands to restore the database files from the prior incarnation and recover them.
---	--

Examples

Resetting a Database After RESETLOGS The following example resets a database after performing incomplete media recovery:

```
run {
    allocate channel dev1 type disk;
    set until logseq 1234 thread 1;
    restore database skip tablespace readonly;
    recover database;
    sql "ALTER DATABASE OPEN RESETLOGS";
    release channel dev1;
}
reset database;
```

Resetting an Old Incarnation The following command makes an old incarnation of database PROD1 current again:

```
# obtain primary key of old incarnation
list incarnation of database prod1;
```

List of Database Incarnations

DB Key	Inc Key	DB Name	DB ID	CUR	Reset SCN	Reset Time
1	2	PROD1	1224038686	NO	1	02-JUL-98
1	582	PROD1	1224038686	YES	59727	10-JUL-98

```
shutdown immediate;
# reset database to old incarnation
reset database to incarnation 2;
# recover it
run {
    allocate channel dev1 type disk;
    restore controlfile;
    startup mount;
    restore database;
    recover database;
    sql "ALTER DATABASE OPEN RESETLOGS";
    release channel dev1;
}
```

Related Topics

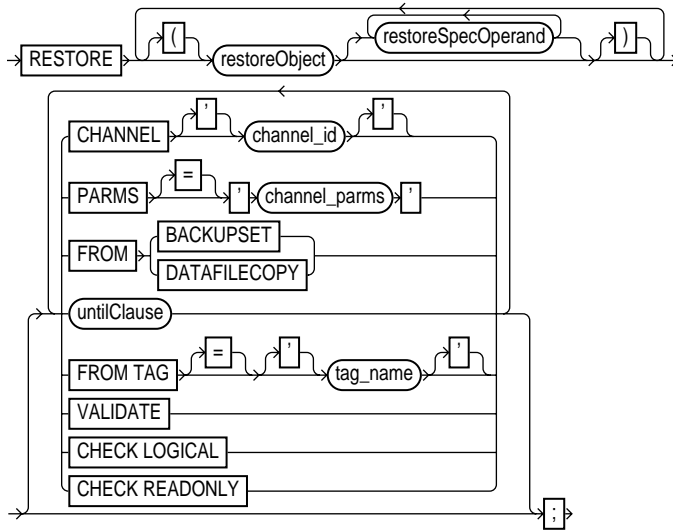
["list" on page 11-76](#)

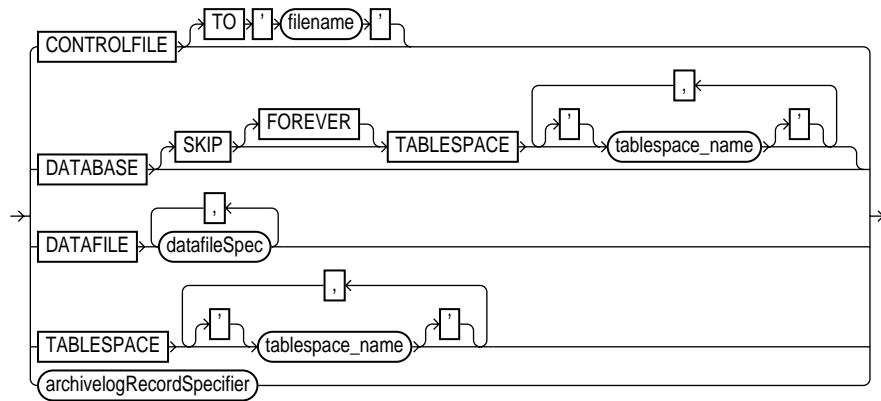
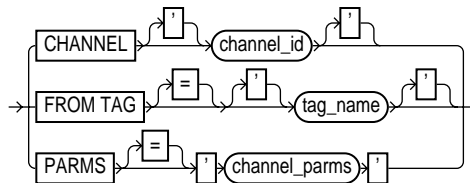
["restore" on page 11-112](#)

["recover" on page 11-88](#)

restore

Syntax



restoreObject::=**restoreSpecOperand::=****Purpose**

To restore files from backups or image copies to the current location, overwriting the files with the same name. RMAN restores backups from disk or tape, but only restores images copies from disk.

Typically, you restore when a media failure has damaged a current datafile, control file, or archived redo log file or prior to performing a point-in-time recovery.

This command restores full backups, incremental level 0 backups, or copies of:

- Database
- Tablespaces
- Datafiles
- Control files
- Archived redo logs

Note: Because the **recover** command automatically restores archived redo logs as needed, you should seldom need to restore archived logs with the **restore** command. Possible reasons for manually restoring archived redo logs are to speed up recovery or stage the logs to multiple destinations.

The **restore** command does not restore incremental backups at levels greater than 0; incremental backups are applied by the **recover** command.

Note that when you perform a restore operation using a backup control file and use a recovery catalog, RMAN automatically adjusts the control file to reflect the structure of the restored backup.

See Also: To learn how to restore files, see [Chapter 9, "Restoring and Recovering with Recovery Manager"](#).

Requirements

- Execute **restore** only within the braces of a **run** command.
- To restore datafiles to their current location, the database must be mounted or the datafiles must be offline. If the entire database is to be restored, then it must be mounted. Use the **set newname** command to restore datafiles to different locations.
- At least one **allocate channel** command must precede a **restore**.
- If you use the **from datafilecopy** option, then the allocated channels must be of **type disk**.
- If you use the **from backupset** operand, then the appropriate type of storage devices must be allocated for the backup sets that need to be restored. If the appropriate device is not allocated, then you may not be able to find a candidate backup set or copy to restore, and the **restore** command fails.
- RMAN only restores backups that were created on the same type of channels that are allocated for the **restore** command.

For example, if you made some backups of a datafile to **disk** channels and others to *'sbt_tape'* channels, and only a **disk** channel is allocated for the **restore** command, then RMAN will not restore from any backups that were created on *'sbt_tape'* channels.

- Open the database with the RESETLOGS option after restoring with a backup control file.

- Do not specify a datafile more than once in a restore job. For example, the following command is illegal since `datafile 1` is both specified explicitly and implied by the SYSTEM tablespace:

```
restore
  tablespace system
  datafile 1;
```

Keywords and Parameters

<i>restoreObject</i>	specifies the objects to be restored.
controlfile	restores the current control file and automatically replicates it to all CONTROL_FILES locations in the parameter file. If you specify a new pathname with the to 'filename' option, RMAN restores the control file to the new location: you must replicate it manually using the replicate command..
database	restores all datafiles in the database except those that are offline or read-only. If you specify the check readonly option, then RMAN will examine the headers of all read-only files and restore those that need restoring. Unlike a backup database , a restore database does not also include the control file. Use an optional skip ... tablespace argument to avoid restoring certain tablespaces, which is useful when you want to avoid restoring tablespaces containing temporary data.
datafile <i>datafileSpec</i>	restores the datafiles specified by filename or absolute datafile number. See " <i>datafileSpec</i> " on page 11-60.
tablespace <i>tablespace_name</i>	restores all datafiles in the specified tablespaces.
<i>archivelogRecord-Specifier</i> clause	restores the specified range of archived redo logs. See " <i>archivelogRecordSpecifier</i> " on page 11-17.
<i>restoreSpec-Operand</i>	specifies options for the <i>restoreObject</i> clause. Note: These parameters override the parameters with the same name at the restore command level.
channel <i>channel_id</i>	specifies the name of a channel to use for this restore operation. If you do not specify a channel, restore uses any available channel allocated with the correct device type.
from tag <i>tag_name</i>	overrides the default selection of the most recent backups or file copy available. The tag restricts the automatic selection to backup sets or file copies that have the specified tag. If multiple backup sets or file copies have a matching tag, then RMAN selects the most recent one.

parms <i>channel_parms</i>	specifies a quoted string containing O/S-specific information. The string is passed to the OSD layer each time a backup piece is restored.
channel <i>channel_id</i>	See the <i>restoreSpecOperand</i> clause.
from tag <i>tag_name</i>	See the <i>restoreSpecOperand</i> clause.
parms <i>channel_parms</i>	See the <i>restoreSpecOperand</i> clause.
from	specifies whether RMAN should restore from a datafilecopy on disk or a backupset . By default restore chooses the most recent backup set or file copy, i.e., the file copy or backup set that needs the least media recovery.
untilClause	limits the selection to those backup sets or file copies that would be suitable for performing a point-in-time recovery. In the absence of any other criteria, RMAN selects the most current file copy or backup set to restore. See " untilClause " on page 11-146.
validate	causes RMAN to decide which backup sets, datafile copies, and archived logs need to be restored and then scans them to verify their contents. This operation creates no output files. Specify this option periodically to verify that the copies and backup sets required to restore the specified files are intact and usable.
check logical	tests data and index blocks that pass physical corruption checks for logical corruption, e.g., corruption of a row piece or index entry. If RMAN finds logical corruption, it logs the block in the <code>alert.log</code> and server session trace file. Provided the sum of physical and logical corruptions detected for a file remain below its maxcorrupt setting, the RMAN command completes and Oracle populates V\$BACKUP_CORRUPTION and V\$COPY_CORRUPTION with corrupt block ranges. If maxcorrupt is exceeded, the command terminates without populating the views. Note: The maxcorrupt setting represents the total number of physical and logical corruptions permitted on a file.
check readonly	checks the datafiles to make sure they exist, are readable, and have the appropriate checkpoint. If any of these conditions is not met, then RMAN restores the files—whether or not they are read-only. By default, RMAN does not restore read-only files when you issue the restore database command.

Examples

Restoring a Tablespace This example takes a tablespace offline, restores it, then performs media recovery:

```
run {  
  # recover tablespace tbs_1 while the database is open  
  allocate channel ch1 type 'sbt_tape';  
  sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE" ;  
}
```

```
restore tablespace tbs_1 ;
recover tablespace tbs_1 ;
sql "ALTER TABLESPACE tbs_1 ONLINE" ;
release channel ch1 ;
}
```

Restoring the Control File This example restores the control file to its default location, replicates it to all CONTROL_FILES locations, and mounts the database:

```
startup nomount;
run {
    allocate channel ch1 type 'sbt_tape';
    restore controlfile;
    alter database mount;
}
```

Restoring the Database Using a Backup Control File This example restores the control file to a new location, replicates it to all control file locations specified in the parameter file, and then mounts the control file in order to restore the database:

```
startup nomount;
run {
    allocate channel ch1 type 'sbt_tape';
    restore controlfile to '/oracle/dbs/cf1.ctl';
    replicate controlfile from '/oracle/dbs/cf1.ctl';
    alter database mount;
    restore database;
}
```

Restoring Archived Redo Logs to a New Location This example restores all archived redo logs to the /oracle/temp_restore directory:

```
run {
    set archivelog destination to '/oracle/temp_restore';
    allocate channel ch1 type disk;
    restore archivelog all;
}
```

Related Topics

["allocate" on page 11-9](#)

["recover" on page 11-88](#)

["untilClause" on page 11-146](#)

resync

Syntax

```
RESYNC CATALOG FROM CONTROLFILECOPY 'filename';
```

Purpose

To perform a full *resynchronization* of the recovery catalog. Resynchronizations can be *full* or *partial*.

In a full resynchronization, RMAN updates all changed records for the *physical schema*: datafiles, tablespaces, redo threads, and online redo logs. If the database is open, RMAN also obtains information about rollback segments.

In a partial resynchronization, RMAN reads the current control file to update changed information, but does not resynchronize metadata about the *physical schema* or rollback segments.

When resynchronizing, RMAN creates a snapshot control file in order to obtain a read-consistent view of the control file, then updates the catalog with any new information from the snapshot.

The **resync catalog** command updates these classes of records:

Record Type	Description
Log history	records that are created whenever a redo log switch occurs. Note that log history records describe an online log switch, not a log archival.
Archived redo logs	Records associated with archived logs that were created by archiving an online redo log, copying an existing archived redo log, or restoring backups of archived redo logs.
Backups	Records associated with backup sets, backup pieces, backup set members, proxy copies, and image copies.
Physical schema	Records associated with datafiles and tablespaces. If the target database is open, then rollback segment information is also updated.

The following commands update the recovery catalog automatically when the target database control file is mounted and the recovery catalog database is available at command execution:

- **backup**
- **change**
- **copy**
- **crosscheck**
- **deleteExpired**
- **duplicate**
- **restore**
- **switch**
- **recover**
- **list**
- **report**

When you run these commands, RMAN automatically executes a full or partial resynchronization as needed. RMAN reads the current control file and does not resynchronize metadata about physical schema unless it determines that this information has changed. If RMAN does detect a change, it performs a full resynchronization.

Use **resync catalog** to perform manual full resynchronizations when:

- The recovery catalog is unavailable when you issue any of the commands that automatically perform a resynchronization.
- You are running in ARCHIVELOG mode, since the recovery catalog is *not* updated automatically when a log switch occurs or when an online redo log is archived.
- You have made changes to the physical structure of the target database such as adding or dropping a tablespace. As with archive operations, the recovery catalog is *not* updated automatically when the physical schema changes.

Requirements

- Execute **resync catalog** at the RMAN prompt or within the braces of a **run** command.
- You must be using a recovery catalog.
- RMAN updates physical schema information in the recovery catalog only when the target database has the current control file mounted. If the target database has mounted a backup control file, a freshly created control file, or a control file that is less current than a control file that was seen previously, then RMAN does not update physical schema information in the recovery catalog.

Keywords and Parameters

from controlfilecopy 'filename'	specifies the name of the control file copy to use for reynchronization. Physical schema information is not updated when you use this option.
--	---

Examples

Resynchronizing After a Structural Change This example adds datafile `sales.f` to tablespace `TBS_1` and then resynchronizes the recovery catalog to reflect the physical database change:

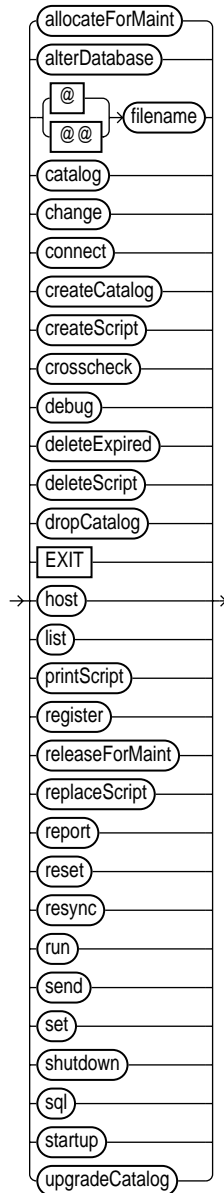
```
startup mount;  
sql "ALTER TABLESPACE tbs_1 ADD DATAFILE ''sales.f'' NEXT 10K MAXSIZE 100K";  
resync catalog;
```

Resynchronizing in ARCHIVELOG Mode This example performs a manual full resync for an ARCHIVELOG database after archiving all unarchived redo logs:

```
sql "ALTER SYSTEM ARCHIVE LOG ALL";  
resync catalog;
```

rmanCmd

Syntax



Purpose

To execute *stand-alone* commands, which are run from the RMAN prompt.

Requirements

Refer to individual entries for information about commands that you can execute from the RMAN prompt.

Keywords and Parameters

@filename executes a series of RMAN commands stored in an O/S file with the specified full pathname, e.g., @\$ORACLE_HOME/dbs/cmd/cmd1.rman. If you do not specify the full pathname, the current working directory is assumed, e.g., @cmd1.rman. Do not use quotes around the string or leave whitespace between the @ and filename. RMAN processes the specified file as if its contents had appeared in place of the @ command.

Note: The file must contain only complete Recovery Manager commands. A syntax error will result if the file contains a partial command.

@@filename is identical to @filename unless used within a script. If contained in a script, @@filename directs RMAN to look for the specified filename in the same path as the command file from which it was called.

For example, assume that your working directory on UNIX is \$ORACLE_HOME, and you invoke RMAN as follows:

```
% rman @$ORACLE_HOME/rdbms/admin/dba/scripts/cmd1.rman
```

Assume that the command @@cmd2.rman appears inside the cmd1.rman script. In this case, the @@ command directs RMAN to look for the file cmd2.rman in the directory \$ORACLE_HOME/rdbms/admin/dba/scripts.

exit exits Recovery Manager.

Examples

Running a Command File This example connects to the target database and recovery catalog from the O/S command line, then runs the command file cmd1.f:

```
% rman target / catalog rman/rman@rcvcat
RMAN> @$ORACLE_HOME/dbs/cmd/cmd1.f
```

Crosschecking Backups This example allocates a maintenance channel, crosschecks backups of tablespace FOO, then backs up tablespace FOO:

```
allocate channel for maintenance type disk;
crosscheck backup of tablespace foo;
delete expired backup of tablespace foo;
run {
```



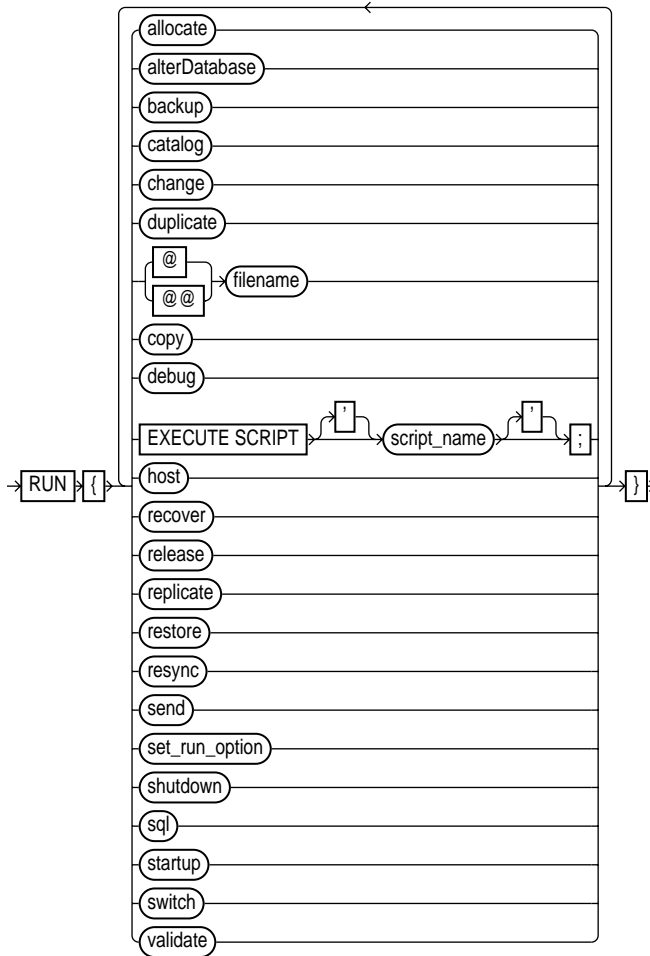
```
        allocate channel c1 type disk;  
        backup tablespace foo;  
    }  
release channel;
```

Related Topics

["run"](#) on page 11-124

run

Syntax



Purpose

To compile and execute *job commands*, which are one or more statements executed within the braces of **run**. The **run** command compiles the list of job commands into one or more job steps and then executes them immediately. RMAN compiles and executes each command before processing the next one.

Requirements

- Execute this command only at the RMAN prompt.
- You must precede and follow the list of job commands with an opening and closing brace.

Keywords and Parameters

Refer to individual entries for information about commands that you can run from the RMAN prompt.

@filename	<p>executes a series of RMAN commands stored in an O/S file with the specified full pathname, e.g., @\$ORACLE_HOME/dbs/cmd/cmd1.rman. If you do not specify the full pathname, the current working directory is assumed, e.g., @cmd1.rman. Do not use quotes around the string or leave whitespace between the @ and filename. RMAN processes the specified file as if its contents had appeared in place of the @ command.</p> <p>Note: The file must contain only complete Recovery Manager commands. A syntax error will result if the file contains a partial command.</p>
@@filename	<p>is identical to @filename unless used within a script. If contained in a script, @@filename directs RMAN to look for the specified filename in the same path as the command file from which it was called.</p> <p>For example, assume that your working directory on UNIX is \$ORACLE_HOME, and you invoke RMAN from the command line as follows:</p> <pre>% rman @\$ORACLE_HOME/rdbms/admin/dba/scripts/cmd1.rman</pre> <p>Assume that the @@cmd2.rman command appears inside the cmd1.rman script. In this case, the @@ command directs RMAN to look for the file cmd2.rman in the directory \$ORACLE_HOME/rdbms/admin/dba/scripts.</p>
execute script script_name	<p>runs the specified stored script. To obtain a listing of all stored scripts, use SQL*Plus to connect to the recovery catalog database as the catalog owner and issue the following query:</p> <pre>select * from rc_stored_script;</pre> <p>See Also: For more information about RC_STORED_SCRIPT, see "RC_STORED_SCRIPT" on page 12-25. For information about creating scripts, see "createScript" on page 11-54.</p>

Examples

Making a Backup This example backs up a database using a single server process to perform the backup:

```
run{
    allocate channel c1 type disk;
    backup database;
}
```

Restoring a Tablespace This example takes tablespace `tbs_1` offline, restores it, then performs complete media recovery:

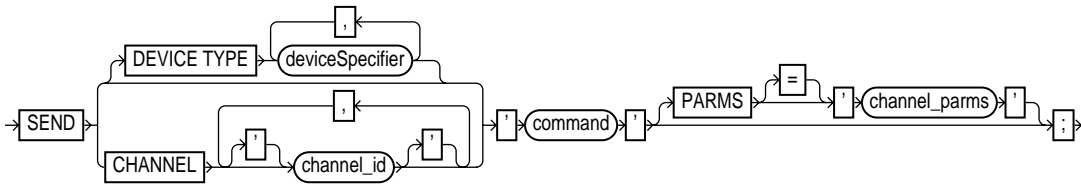
```
run {
    allocate channel ch1 type 'sbt_tape';
    sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE" ;
    restore tablespace tbs_1 ;
    recover tablespace tbs_1 ;
    sql "ALTER TABLESPACE tbs_1 ONLINE" ;
    release channel ch1 ;
}
```

Executing a Script This example executes the stored script `backupdb`:

```
run { execute script backupdb; }
```

send

Syntax



Purpose

To send a vendor-specific quoted string to one or more specific channels. See your media management documentation to determine which commands are supported.

Requirements

- Execute **send** at the RMAN prompt or within the braces of a **run** command.
- You must use a media manager to use **send** and only with the commands supported by the media manager. The contents of the quoted string are not interpreted by Oracle, but are passed unaltered to the media management sub-system.

Keywords and Parameters

<i>command</i>	specifies a vendor-specific media management command. See your media management documentation to determine which commands are supported.
device type <i>deviceSpecifier</i>	specifies the type of storage device and sends the command to all channels of the specified type. See " deviceSpecifier " on page 11-66.
channel <i>channel_id</i>	specifies which channel to use. If you do not specify this keyword, RMAN uses all allocated channels. You must specify a channel id, which is the name of the channel, after the channel keyword. Oracle uses the channel id with the release channel command and also to report I/O errors.

send

parms specifies parameters affecting the device you have allocated. Do not use this port-specific string if you have specified **type disk**.

'channel_parms'

If you use **parms** in conjunction with **type 'sbt_tape'**, then you can specify environment variables with the following syntax:

```
PARMS = "ENV = (var1=value1, var2=value2, var3=value3 . . . )"
```

The maximum length of the quoted string is 1000 bytes.

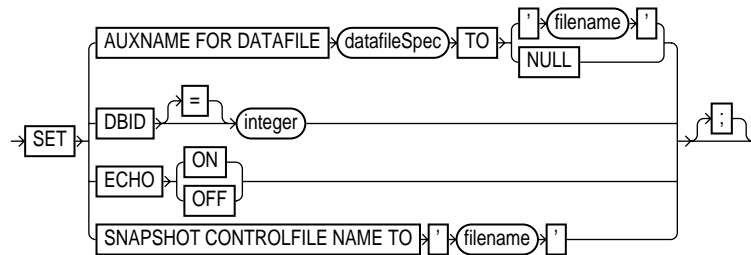
Examples

Sending a String to the Media Manager This example sends a vendor-specific command to a media manager:

```
send device type 'sbt_tape' 'A223dr';
```

set

Syntax



Purpose

Use **set** to:

- Specify the filenames for the auxiliary database during TSPITR.
- Specify new datafile filenames for use in the **duplicate** command.
- Display executed RMAN commands in the message log.
- Specify a database's db identifier.
- Set the filename of the snapshot control file.

Requirements

- Use **set** only at RMAN prompt.
- Use **set dbid** only if you have not yet connected to the target database.
- You must have a recovery catalog to use the **auxname for datafile to 'filename'** option.

Keywords and Parameters

auxname for datafile
datafileSpec to
 'filename'

sets the target database datafile to its new filename on the auxiliary database. If you are performing TSPITR or using the **duplicate** command, setting this option allows you to pre-configure the filenames for use on the auxiliary database without manually specifying the auxiliary filenames during the procedure.

For example, use this command during TSPITR if your datafiles are on raw disks and you want to restore auxiliary datafiles to raw disk for performance reasons. Typically, you set the **auxname** parameter in TSPITR for the datafiles of the SYSTEM tablespace and the tablespaces containing rollback segments. Take care not to overlay the files that are in use by the production database and which can be discarded after TSPITR completes. In essence, the **auxname** of a datafile is the location where TSPITR can create a temporary copy of it.

When renaming files with the **duplicate** command, **set auxname** is an alternative to **set newname**. The difference is that once you set the **auxname**, you do not need to reset it when you issue another **duplicate** command: it remains in effect until you issue **set auxname ... to null**. In contrast, you must reissue the **set newname** command every time you rename files.

See Also: To learn how to perform RMAN TSPITR, see [Appendix A, "Performing Tablespace Point-in-Time Recovery with Recovery Manager"](#). To learn how to duplicate a database, see [Chapter 10, "Creating a Duplicate Database with Recovery Manager"](#).

to null unspecifies the current value for **auxname** for the specified datafile.

dbid *integer*

specifies the *db identifier*, which is a unique 32-bit identification number computed when the database is created. The DBID column of the V\$DATABASE data dictionary view displays the identifier. The DBID is also stored in the DB table of the recovery catalog.

The **set dbid** command is useful for restoring the control file when each of these conditions is met:

- The control file has been lost and must be restored from a backup.
- You are using a recovery catalog.
- Multiple databases registered in the recovery catalog share a database name.
- You receive the "RMAN-20005: target database name is ambiguous" message when you attempt to restore the control file.

If these conditions are *not* met, RMAN will correctly identify the control file to restore, so you do not need to use the **set dbid** command.

RMAN accepts **set dbid** only if you have not yet connected to the target database, i.e., **set dbid** must precede the **connect target** command. If the target database is mounted, then RMAN verifies that the user-specified DBID matches the DBID from the database; if not, RMAN signals an error. If the target database is not mounted, RMAN uses the user-specified DBID to restore the control file. Once you have restored the control file, you can mount the database to restore the rest of the database.

echo [on off]	<p>controls whether RMAN commands are displayed in the message log. When reading commands from a command file, RMAN automatically echoes those commands to the message log. When reading commands from STDIN, RMAN does not echo those commands to the message log unless the set echo on command is used.</p> <p>The command is useful only when <code>stdin</code> and <code>stdout</code> have been redirected. For example, in UNIX you can re-direct RMAN's input and output in this manner:</p> <pre>% rman target sys/sys_pwd@prodl catalog rman/rman@rcat < input_file > output_file</pre> <p>By specifying set echo on, you enable the commands contained in <code>input_file</code> to be visible in <code>output_file</code>.</p>
snapshot controlfile name to 'filename'	<p>sets the snapshot control file name in the target database to the specified location. RMAN uses the snapshot control file to resynchronize the recovery catalog. For more information about snapshot control files, see "Determining the Snapshot Control File Location" on page 5-3.</p>

Examples

Restoring the Control File This example uses the user-specified DBID to restore the control file. Once you have restored the control file, you can mount the database to restore the rest of the database.

```
set dbid = 862893450;
connect target;
startup nomount;
run {
  allocate channel dev1 type disk;
  # restoring the control file from its default location automatically replicates it
  restore controlfile;
  alter database mount;
}
```

Specifying the Snapshot Control File Location This example specifies a new location for the snapshot control file and then resynchronizes the recovery catalog.

```
set snapshot controlfile name to '/oracle/dbs/snap.cf';
resync catalog;
```

Specifying the Snapshot Control File Location This example duplicates a database to a remote host with a different directory structure, using **set auxname** to specify new filenames for the datafiles:

```
# set auxiliary names for the datafiles
set auxname for datafile 1 to '/oracle/auxfiles/aux_1.f';
set auxname for datafile 2 to '/oracle/auxfiles/aux_2.f';
set auxname for datafile 3 to '/oracle/auxfiles/aux_3.f';
set auxname for datafile 4 to '/oracle/auxfiles/aux_4.f';
```

```
run {
  allocate auxiliary channel dupdb type disk;
  duplicate target database to dupdb
  logfile
    group 1 ('$ORACLE_HOME/dbs/dupdb_log_1_1.f',
             '$ORACLE_HOME/dbs/dupdb_log_1_2.f') size 200K,
    group 2 ('$ORACLE_HOME/dbs/dupdb_log_2_1.f',
             '$ORACLE_HOME/dbs/dupdb_log_2_2.f') size 200K reuse;
}
# Un-specify the auxiliary names for your datafiles so that they will not be overwritten
# by mistake:
set auxname for datafile 1 to null;
set auxname for datafile 2 to null;
set auxname for datafile 3 to null;
set auxname for datafile 4 to null;
```

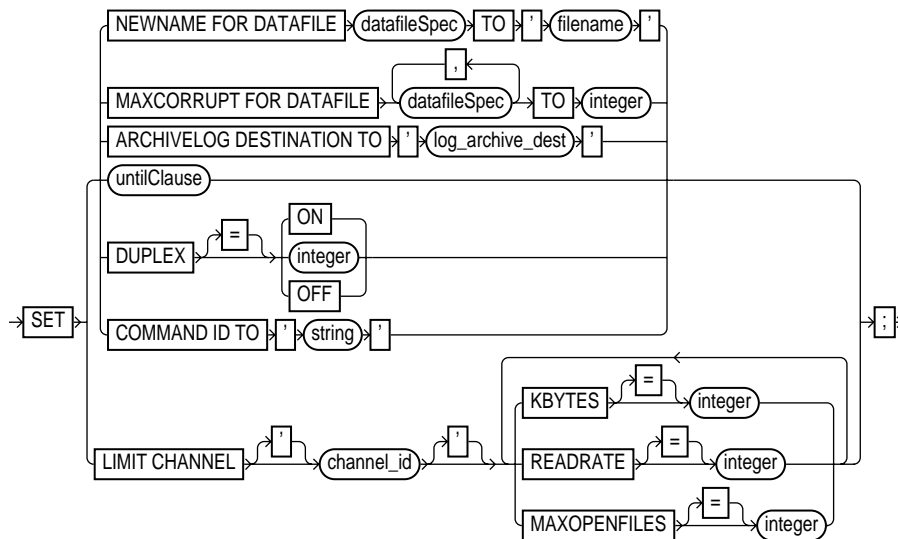
Related Topics

["set_run_option" on page 11-133](#)

["duplicate" on page 11-69](#)

set_run_option

Syntax



Purpose

To set attributes for a **run** command that persist until the end of the job. The specified attributes affect all statements within **run** that follow the **set** command. Use **set** to:

- Specify new filenames for datafiles.
- Specify a limit for the number of permissible block corruptions.
- Override default archived redo log destinations.
- Set an end time, SCN, or log sequence number for recovery.
- Specify that backups should be duplexed.
- Determine which server session corresponds to which channel.
- Limit the number of buffers per second that will be read from each input datafile on a specified channel.

- Limit the number of input files that a **backup** operation can have open at any given time for a specified channel.
- Limit the size of the backup pieces for a specified channel.

Requirements

- Execute this command only within the braces of a **run** command.
- The **set duplex** command affects all channels allocated after issuing the command and is in effect until explicitly disabled or changed. The **set duplex** command does not affect previously allocated channels.

Keywords and Parameters

newname for datafile <i>datafileSpec to filename</i>	sets the default name for all subsequent restore or switch commands that affect the specified datafile (see " datafileSpec " on page 11-60). If you do not issue this command before the datafile restore operation, then RMAN restores the file to its default location.
maxcorrupt for datafile <i>datafileSpec to integer</i>	<p>sets a limit on the number of previously undetected physical block corruptions that Oracle will allow in a specified datafile or list of datafiles (see "datafileSpec" on page 11-60). If a backup or copy command detects more than the specified number of corruptions, the command aborts. The default limit is zero, meaning that RMAN tolerates no corrupt blocks.</p> <p>Note: If you specify check logical, then the maxcorrupt limit applies to logical corruptions as well.</p>
archivelog destination to <i>log_archive_dest'</i>	<p>overrides the LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_1 initialization parameter in the target database when forming names for restored archive logs during subsequent restore and recover commands. RMAN restores the logs to the destination specified in '<i>log_archive_dest</i>'. Use this parameter to restore archived redo logs that are not already on disk.</p> <p>Use this command to stage many archived logs to different locations while a database restore is occurring. RMAN knows where to find the newly restored archive logs; it does not require them to be in the destination specified by LOG_ARCHIVE_DEST_1 or LOG_ARCHIVE_DEST. For example, if you specify a different destination from the one in the parameter file and restore archived redo log backups, subsequent restore and recovery operations will detect this new location. RMAN always looks for archived redo logs on disk first before restoring them from backup sets.</p>
<i>untilClause</i>	specifies an end time, SCN, or log sequence number for a subsequent restore or recover command. See " untilClause " on page 11-146.
duplex	specifies the number of copies of each backup piece that the channels should create: 1, 2, 3, or 4. The set duplex command, which affects only the backup command, affects all channels allocated after issuing the command and is in effect until explicitly disabled (off) or changed during the session. By default duplex is off , i.e., RMAN produces a single backup set. If you specify on , RMAN produces two identical backup sets.

command id to <i>'string'</i>	<p>enters the specified string into the V\$SESSION.CLIENT_INFO column of all channels. Use this information to determine which Oracle server sessions correspond to which RMAN channels.</p> <p>The V\$SESSION.CLIENT_INFO column contains information for each RMAN server session. The data appears in one of the following formats:</p> <ul style="list-style-type: none"> ■ id=<i>string</i> ■ id=<i>string</i>, ch=<i>channel_id</i> <p>The first form appears in the RMAN target database connection. The second form appears in all allocated channels. When the current job is complete, the V\$SESSION.CLIENT_INFO column will be cleared.</p> <p>See Also: For more information on V\$SESSION.CLIENT_INFO, see the <i>Oracle8i Reference</i>.</p>
limit channel <i>channel_id</i>	<p>sets parameters that specify limits applying to any backup or copy command that executes using the allocated channel.</p> <p>kbytes <i>integer</i> specifies the maximum size in kilobytes of the backup pieces created on this channel.</p> <p>readrate <i>integer</i> specifies the maximum number of buffers (each of size DB_BLOCKSIZE * DB_FILE_DIRECT_IO_COUNT) per second that will be read for backup or copy from each of the input datafiles. Use this parameter to ensure that the command does not consume excessive disk bandwidth and thereby degrade online performance.</p> <p>maxopenfiles <i>integer</i> controls the maximum number of input files that a backup command can have open at any given time. Use this parameter to prevent "Too many open files" O/S error messages when backing up a large number of files into a single backup set. If you do not specify maxopenfiles, then a maximum of 32 input files can be open concurrently.</p>

Examples

Setting the Command ID This example sets the command ID, backs up the DATA_1 tablespace, hosts out to the O/S, then archives the online redo logs:

```
run {
  set command id to 'rman';
  allocate channel t1 type 'SBT_TAPE'
  allocate channel t2 type 'SBT_TAPE';
  backup
    incremental level 0
    filesperset 5
    tablespace data_1;
  host;
  sql 'ALTER SYSTEM ARCHIVE LOG ALL';
}
```

Duplexing a Backup Set This example makes two identical backup sets of datafile 1:

```
run {
  set duplex = ON;
  allocate channel dev1 type disk;
  backup
    filesperset 1
    datafile 1;
}
```

Setting Channel Limits This example allocates three channels and sets the maximum size for backup pieces created on each channel. It also makes three identical backups of the database:

```
startup mount;
run {
  set duplex=3;
  allocate channel ch1 type 'sbt_tape';
  allocate channel ch2 type 'sbt_tape';
  allocate channel ch3 type 'sbt_tape';

  set limit channel ch1 kbytes 2097150;
  set limit channel ch2 kbytes 2097150;
  set limit channel ch3 kbytes 2907150;

  backup
    filesperset 5
    database;
  alter database open;
}
```

Related Topics

["recover" on page 11-88](#)

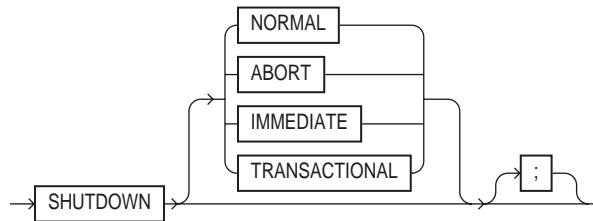
["restore" on page 11-112](#)

["set" on page 11-129](#)

["untilClause" on page 11-146](#)

shutdown

Syntax



Purpose

To shut down the target database without exiting RMAN. This command is equivalent to using the SQL*Plus SHUTDOWN statement.

See Also: For information on how to start up and shut down a database, see the *Oracle8i Administrator's Guide*. For SHUTDOWN syntax, see the *SQL*Plus User's Guide and Reference*.

Requirements

- Execute this command at the RMAN prompt or within the braces of a **run** command.
- You cannot use the RMAN **shutdown** command to shut down the recovery catalog database. To shut down this database, start a SQL*Plus session and issue a SHUTDOWN statement.
- The **normal**, **transactional**, and **immediate** options all perform a clean close of the database. The **abort** option does not cleanly close the database; Oracle will perform instance recovery at startup.
- If your database operates in NOARCHIVELOG mode, then you must shut down the database cleanly and then issue a **startup mount** before making a backup.

Keywords and Parameters

normal	shuts down the database with normal priority (default option), which means: <ul style="list-style-type: none">■ No new connections are allowed after the statement is issued.■ Before the database is shut down, Oracle waits for all currently connected users to disconnect from the database.■ The next startup of the database will not require instance recovery.
abort	aborts the target instance, with the following consequences: <ul style="list-style-type: none">■ All current client SQL statements are immediately terminated.■ Uncommitted transactions are not rolled back until next startup.■ Oracle disconnects all connected users.■ Oracle will perform crash recovery on the database at next startup.
immediate	shuts down the target database immediately, with the following consequences: <ul style="list-style-type: none">■ Current client SQL statements being processed by Oracle are allowed to complete.■ Uncommitted transactions are rolled back.■ All connected users are disconnected.
transactional	shuts down the target database while minimizing interruption to clients, with the following consequences: <ul style="list-style-type: none">■ Clients currently conducting transactions are allowed to complete, i.e., either commit or abort before shutdown.■ No client can start a new transaction on this instance; any client attempting to start a new transaction is disconnected.■ After all transactions have either committed or aborted, any client still connected is disconnected.

Examples

Shutting Down a Database in Immediate Mode This example waits for current SQL transactions to be processed before shutting down, then mounts the database:

```
shutdown immediate;  
startup mount;
```

Shutting Down a Database in NOARCHIVELOG Mode This example backs up a database running in NOARCHIVELOG mode:

```
shutdown abort;  
startup dba;
```



```
shutdown;  
startup mount;  
# executing the above commands ensures that database is in proper state for noarchive  
# backup  
run {  
    set duplex = 4;  
    allocate channel dev1 type 'sbt_tape';  
    backup database;  
    alter database open;  
}
```

Related Topics

["alterDatabase"](#) on page 11-15

["startup"](#) on page 11-142

sql

Syntax

```
→ SQL > 'command' > ;
```

Purpose

To execute a SQL statement from within Recovery Manager. For valid SQL syntax, see the *Oracle8i SQL Reference*.

Requirements

- Execute the **sql** command at the RMAN prompt or within the braces of a **run** command.
- If the string that RMAN passes to PL/SQL contains a filename, then the filename must be enclosed in duplicate *single* quotes and the entire string following the **sql** parameter must be enclosed in *double* quotes. For example, use the following syntax:

```
sql "CREATE TABLESPACE temp1 DATAFILE ' '$ORACLE_HOME/dbs/temp1.f' ' "
```

If you attempt to use single quotes for the string following the SQL parameter or single quotes for the filename, then the command fails.

- You cannot execute SELECT statements.

Keywords and Parameters

<i>'command'</i>	specifies a SQL statement for execution. For example, issuing the following: <pre>sql 'ALTER SYSTEM ARCHIVE LOG ALL';</pre> at the RMAN prompt archives the online redo logs.
------------------	--

Examples

Making an O/S Copy of an Online Tablespace This example hosts out to the operating system to make an O/S copy of online tablespace TBS_1 and then catalogs it:

```
sql 'ALTER TABLESPACE tbs_1 BEGIN BACKUP';
host 'cp $ORACLE_HOME/dbs/tbs_1.f/dbs/tbs_1.f $ORACLE_HOME/copy/temp3.f';
sql 'ALTER TABLESPACE tbs_1 END BACKUP';
catalog datafilecopy '$ORACLE_HOME/copy/temp3.f';
sql 'ALTER SYSTEM ARCHIVE LOG ALL';
```

Specifying a Filename within a Quoted String This example specifies a filename using duplicate single quotes within the context of a double-quoted string:

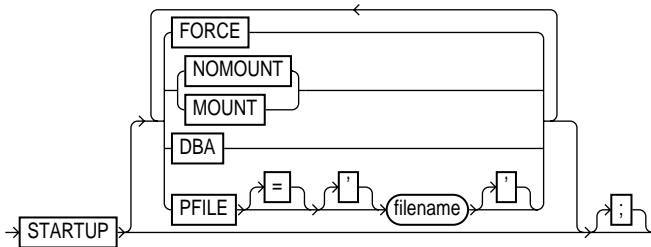
```
sql "ALTER TABLESPACE tbs_1 ADD DATAFILE ' '/oracle/dbs/tbs_7.f' ' NEXT 10K MAXSIZE 100k;"
```

Opening with the RESETLOGS Option This example performs incomplete recovery and opens the database with the RESETLOGS option:

```
run {
  set until scn 1000;
  allocate channel c1 type 'sbt_tape';
  restore database;
  recover database;
  sql 'ALTER DATABASE OPEN RESETLOGS';
  reset database;
}
```

startup

Syntax



Purpose

To start the database from within the RMAN environment. This command is equivalent to using the SQL*Plus `STARTUP` command. You can:

- Start the instance without mounting a database.
- Start the instance and mount the database, but leave it closed.
- Start the instance, and mount and open the database in:
 - unrestricted mode (accessible to all users).
 - restricted mode (accessible to DBAs only).

See Also: To learn how to start up and shut down a database, see the *Oracle8i Administrator's Guide*. For SQL*Plus `STARTUP` syntax, see the *SQL*Plus User's Guide and Reference*.

Requirements

- Execute this command either at the RMAN prompt or within the braces of a `run` command.
- You cannot use the RMAN `startup` command to open the recovery catalog database. To start this database, start a SQL*Plus session and execute a `STARTUP` statement.

Keywords and Parameters

If you do not specify any options, RMAN mounts and opens the database.

force	executes either of these operations: <ul style="list-style-type: none"> ■ If the database is open, this option first shuts down the database with a shutdown abort statement before re-opening it. ■ If the database is closed, this option opens the database.
nomount	starts the instance without mounting the database.
mount	starts the instance, then mounts the database without opening it. If you specify neither the mount nor nomount options, the startup command opens the database.
dba	restricts access to the database to users with the RESTRICTED SESSION privilege.
pfile = filename	specifies the filename of the <code>init.ora</code> file for the target database. If this parameter is not specified, the default <code>init.ora</code> filename is used.

Examples

Opening the Database Using the Default Parameter File This example starts and opens the database:

```
startup;
```

Mounting the Database While Specifying the Parameter File This example forces a **shutdown abort** and then mounts the database with restricted access, specifying a non-default parameter file location:

```
startup force mount dba pfile=t_init1.ora;
```

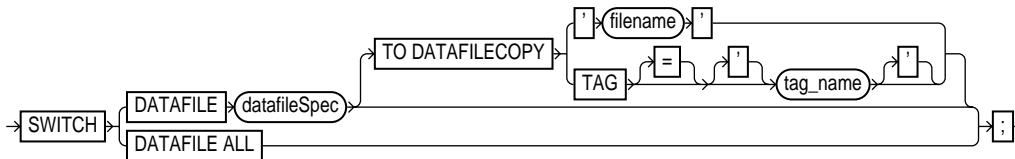
Related Topics

["alterDatabase"](#) on page 11-15

["shutdown"](#) on page 11-137

switch

Syntax



Purpose

To specify that a datafile copy is now the *current datafile*, i.e., the datafile pointed to by the control file. A switch is equivalent to using the ALTER DATABASE RENAME DATAFILE statement: Oracle renames the files in the control file but does not actually rename them on your operating system. Note that switching deletes the records for the datafile copy in the recovery catalog and the control file.

Requirements

- Execute **switch** within the braces of a **run** command.
- If your control file is a restored backup control file, then **switch** adds the datafile to the control file if it is not there already. You can only add datafiles via **switch** that were created *after* the backup control file was created.

Keywords and Parameters

datafile *datafileSpec*

specifies the datafile that you wish to rename. After the switch, the control file no longer views the specified file as current. For example, this command points the control file from tbs_1.f to cp1.f:

```
switch datafile '$/dbs/tbs_1.f' to datafile copy '$/dbs/copies/cp1.f';
```

If you do not specify a **to** option, then RMAN uses the filename specified on a prior **set newname** command (see "[set_run_option](#)" on page 11-133) for this file number as the switch target.

to datafilecopy ' specifies the input copy file for the switch, i.e., the datafile copy that you wish to rename. For example, if you issue:

```
switch datafile 2 to datafilecopy '/oracle/dbs/df2.copy';
```

The control file will list `df2.copy` as the filename for `datafile 2`.

You can also specify the datafile copy by tag. If the tag is ambiguous, then the most current copy is used, i.e., the one that requires the least media recovery.

When the tag is not unique, then RMAN understands the tag to refer to the most recently created copy. Thus, the command:

```
switch datafile 3 to datafilecopy tag mondayPMcopy;
```

switches `datafile 3` to the most recently created Monday evening copy.

datafile all specifies that all datafiles for which a **set newname for datafile** command (see "[set_run_option](#)" on page 11-133) has been issued in this job are switched to their new name.

Examples

Switching After a Restore This example allocates one disk device and one tape device to allow RMAN to restore both from disk and tape.

```
run {
  allocate channel dev1 type disk;
  allocate channel dev2 type 'sbt_tape';
  sql "ALTER TABLESPACE tbs_1 OFFLINE IMMEDIATE";
  set newname for datafile 'disk7/oracle/tbs11.f'
    to 'disk9/oracle/tbs11.f';
  restore tablespace tbs_1;
  switch datafile all;
  recover tablespace tbs_1;
  sql "ALTER TABLESPACE tbs_1 ONLINE";
}
```

Related Topics

["printScript"](#) on page 11-86

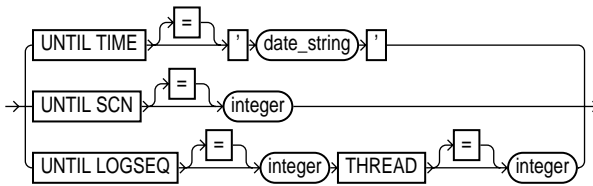
["restore"](#) on page 11-112

["run"](#) on page 11-124

["set"](#) on page 11-129

untilClause

Syntax



Purpose

A sub-clause that specifies an upper limit by time, SCN, or log sequence number for various RMAN operations.

Requirements

All date strings must be either:

- Formatted according to the NLS date format specification currently in effect.
- Created by a SQL expression that returns a DATE value, e.g., 'SYSDATE-30'.

Use this sub-clause in conjunction with the following commands:

- **recover**
- **report**
- **restore**
- **set_run_option**

Keywords and Parameters

until time <i>'date_string'</i>	specifies a time as an upper limit.
until scn <i>integer</i>	specifies an SCN as an upper limit.
until logseq <i>integer</i>	specifies a redo log sequence number as an upper limit.
thread <i>integer</i>	indicates the thread number for the redo log in question.

Examples

Performing Incomplete Recovery Until a Log Sequence Number This example assumes that log sequence 1234 was lost due to a disk crash and the database needs to be recovered using available archived redo logs.

```
run {
  allocate channel ch1 type disk;
  allocate channel ch2 type 'sbt_tape';
  set until logseq 1234 thread 1;
  restore controlfile to '$ORACLE_HOME/dbs/cf1.f' ;
  replicate controlfile from '$ORACLE_HOME/dbs/cf1.f';
  alter database mount;
  restore database;
  recover database;
  sql "ALTER DATABASE OPEN RESESTLOGS";
}
```

Performing DBPITR to a Specified SCN This example recovers the database until a specified SCN:

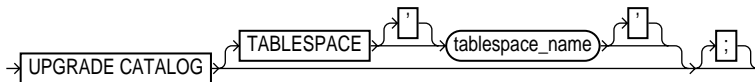
```
startup mount;
run{
  allocate channel ch1 type disk;
  restore database;
  recover database until scn 1000;
  sql "ALTER DATABASE OPEN RESESTLOGS";
}
```

Reporting Obsolete Backups This example assumes that you want to be able to recover to any point within the last seven days. It considers backups made more than a week ago as obsolete:

```
report obsolete until time 'SYSDATE-7';
```

upgradeCatalog

Syntax



Purpose

To upgrade the recovery catalog schema from an older version to the version required by the RMAN executable. For example, if you use an 8.0 recovery catalog with an 8.1 version of RMAN, then you must upgrade the catalog.

Requirements

- RMAN must be connected to the recovery catalog.
- You must enter the **upgrade** command twice in a row to confirm the upgrade.
- You will receive an error if the recovery catalog is already at a version greater than needed by the RMAN executable. RMAN permits the command to be run if the recovery catalog is already current, however, so that the packages can be re-created if necessary. RMAN displays all error messages generated during the upgrade in the message log.

Keywords and Parameters

tablespace <i>tablespace_name</i>	specifies the tablespace in which to store the recovery catalog. If not specified, then no tablespace parameter will be used in the CREATE TABLE statements used to upgrade the recovery catalog, which means that the catalog will be stored in the default tablespace.
---	---

Examples

Upgrading a Catalog This example connects to recovery catalog database RECDB and then upgrades it to a more current version:

```
rman catalog rcat/rcat@recdb
```

```
RMAN-06008: connected to recovery catalog database  
RMAN-06186: PL/SQL package rcat.DBMS_RCVCAT version 08.00.04 in RCVCAT
```

database is too old

```
RMAN> upgrade catalog
```

```
RMAN-06435: recovery catalog owner is rcat
```

```
RMAN-06442: enter UPGRADE CATALOG command again to confirm catalog upgrade
```

```
RMAN> upgrade catalog
```

```
RMAN-06408: recovery catalog upgraded to version 08.01.03
```

```
RMAN-06452: DBMS_RCVMAN package upgraded to version 08.01.05
```

```
RMAN-06452: DBMS_RCVCAT package upgraded to version 08.01.03
```

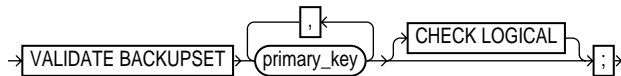
Related Topics

["createCatalog" on page 11-52](#)

["dropCatalog" on page 11-68](#)

validate

Syntax



Purpose

To examine a backup set and report whether it can be restored. RMAN scans all of the backup pieces in the specified backup sets and looks at the checksums to verify that the contents are intact so that the backup can be successfully restored if necessary. Note that **validate backupset** actually tests whether the backup sets can be restored, whereas **change ... crosscheck** merely examines the headers of the specified files.

Use this command when you suspect that one or more backup pieces in a backup set are missing or have been damaged. Use **validate backupset** to specify which backups to test; use the **validate** option of the **restore** command to let RMAN choose which backups to validate.

Requirements

- Use this command only within the braces of a **run** command.
- Allocate at least one channel before executing a **validate backupset** statement.

Keywords and Parameters

<i>primary_key</i>	specifies the backup sets to be validated by <i>primary_key</i> . Obtain the primary keys of backup sets by executing a list statement or, if you use a recovery catalog, by querying the RC_BACKUP_SET fixed view.
check logical	tests data and index blocks that pass physical corruption checks for logical corruption, e.g., corruption of a row piece or index entry. If RMAN finds logical corruption, it logs the block in the <code>alert.log</code> and server session trace file. The RMAN command completes and Oracle populates V\$BACKUP_CORRUPTION and V\$COPY_CORRUPTION with corrupt block ranges.

Note: **validate** does not use **maxcorrupt**.

Examples

Validating a Backup Set This example validates the status of the backup set whose primary key is 12:

```
run{
    allocate channel ch1 type disk;
    validate backupset 12;
}
# As the output indicates, RMAN determines whether it is possible to restore the
# specified backup set.
RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel ch1: sid=10 devtype=DISK

RMAN-03022: compiling command: validate
RMAN-03023: executing command: validate
RMAN-08016: channel ch1: starting datafile backupset restore
RMAN-08502: set_count=16 set_stamp=341344502 creation_time=14-AUG-98
RMAN-08023: channel ch1: restored backup piece 1
RMAN-08511: piece handle=/oracle/dbs/0ga5h07m_1_1 params=NULL
RMAN-08024: channel ch1: restore complete
RMAN-08031: released channel: ch1
```

Related Topics

["restore" on page 11-112](#)

validate

Recovery Catalog Views

This chapter contains descriptions of recovery catalog views. You can only access these views if you have created a recovery catalog.

Note: These views are not normalized, but are optimized for RMAN usage. Hence, most catalog views have redundant values that proceed from the join of several tables.

RC_ARCHIVED_LOG

This view lists historical information about all archived redo logs. It corresponds to the VSARCHIVED_LOG dynamic performance view in the control file.

An archived redo log record is inserted after the online redo log is successfully archived or cleared (NAME column is NULL if the log was cleared). If the log is archived multiple times (the maximum is 5), there will be multiple archived log records with the same THREAD#, SEQUENCE#, and FIRST_CHANGE#, but with a different name. An archived log record is also inserted when an archived log is restored from a backup set or a copy.

Note that an archived redo log can have no log history record if its log history record is written over in the control file or after a RESETLOGS operation.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
AL_KEY	NUMBER	NOT NULL	The primary key of the archived redo log in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The archived redo log RECID from VSARCHIVED_LOG. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The archived redo log stamp from VSARCHIVED_LOG. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
THREAD#	NUMBER	NOT NULL	The number of the redo thread.
SEQUENCE#	NUMBER	NOT NULL	The log sequence number.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS when the record was created.
FIRST_CHANGE#	NUMBER	NOT NULL	The SCN generated when Oracle switched into the redo log.

Column	Datatype	NULL	Description
FIRST_TIME	DATE	NOT NULL	The time when Oracle switched into the redo log.
NEXT_CHANGE#	NUMBER	NOT NULL	The first SCN of the next redo log in the thread.
NEXT_TIME	DATE		The first timestamp of the next redo log in the thread.
BLOCKS	NUMBER	NOT NULL	The number of blocks written (also the size of the archived log when the copy was made).
BLOCK_SIZE	NUMBER	NOT NULL	The size of the block in bytes.
COMPLETION_TIME	DATE	NOT NULL	The time when the redo log was archived or copied.
ARCHIVED	VARCHAR2(3)		Indicates whether the log was archived: YES (archived redo log) or NO (inspected online redo log). When Oracle inspects online logs, it looks at the file headers and adds an archived log record for each online log. Oracle sets ARCHIVED to NO to prevent online logs from being backed up.
STATUS	VARCHAR2(1)	NOT NULL	The status of the archived redo log: A (available), U (unavailable), or D (deleted).

RC_BACKUP_CONTROLFILE

This view lists information about control files in backup sets. Note that a backup datafile record with file number 0 represents the backup control file in the V\$BACKUP_DATAFILE dynamic performance view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
BCF_KEY	NUMBER	NOT NULL	The primary key of the control file backup in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The RECID from V\$BACKUP_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The STAMP from V\$BACKUP_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.

RC_BACKUP_CORRUPTION

Column	Datatype	NULL	Description
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set to which this record belongs in the recovery catalog. Use this column to form a join with RC_BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS when the record was created.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	The control file checkpoint SCN.
CHECKPOINT_TIME	DATE	NOT NULL	The control file checkpoint time.
CREATION_TIME#	DATE	NOT NULL	The control file creation time.
BLOCK_SIZE	NUMBER	NOT NULL	The size of the blocks in bytes.
OLDEST_OFFLINE_RANGE	NUMBER	NOT NULL	Internal use only.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (available), U (unavailable), or D (deleted).

RC_BACKUP_CORRUPTION

This view lists corrupt block ranges in datafile backups. It corresponds to the V\$BACKUP_CORRUPTION dynamic performance view in the control file. Note that corruptions are not tolerated in control file and archived redo log backups.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.

Column	Datatype	NULL	Description
RECID	NUMBER	NOT NULL	The record identifier from V\$BACKUP_CORRUPTION. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The stamp propagated from V\$BACKUP_CORRUPTION. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set to which this record belongs in the recovery catalog. Use this column to form a join with RC_BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
PIECE#	NUMBER	NOT NULL	The backup piece to which the block belongs.
BDF_KEY	NUMBER	NOT NULL	The primary key for the datafile backup or copy in the recovery catalog. Use this key to join with RC_BACKUP_DATAFILE. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
BDF_RECID	NUMBER	NOT NULL	The RECID value from V\$BACKUP_DATAFILE.
BDF_STAMP	NUMBER	NOT NULL	The STAMP value from V\$BACKUP_DATAFILE.
FILE#	NUMBER	NOT NULL	The absolute file number for the datafile.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN at backup creation.
BLOCK#	NUMBER	NOT NULL	The block number of the first corrupted block in the file.
BLOCKS	NUMBER	NOT NULL	The number of corrupted blocks found beginning with BLOCK#.
CORRUPTION_CHANGE#	NUMBER		The SCN at which corruption was detected.
MARKED_CORRUPT	VARCHAR2(3)		YES if this corruption was not previously detected by Oracle or NO if it was already known by Oracle.

RC_BACKUP_DATAFILE

This view lists information about datafiles in backup sets. It corresponds to the V\$BACKUP_DATAFILE dynamic performance view. A backup datafile is uniquely identified by BDF_KEY.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
BDF_KEY	NUMBER	NOT NULL	The primary key of the datafile backup in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The backup datafile RECID from V\$BACKUP_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The backup datafile stamp from V\$BACKUP_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set to which this record belongs in the recovery catalog. Use this column to form a join with RC_BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
BS_RECID	NUMBER	NOT NULL	The RECID from V\$BACKUP_SET.
BS_STAMP	NUMBER	NOT NULL	The STAMP from V\$BACKUP_SET.
BACKUP_TYPE	VARCHAR2(1)	NOT NULL	The type of the backup: D (full or level 0 incremental) or I (incremental level 1 or higher).
INCREMENTAL_LEVEL	NUMBER		The level of the incremental backup: NULL or 0 - 4.
COMPLETION_TIME	DATE		The completion time of the backup.

Column	Datatype	NULL	Description
FILE#	NUMBER	NOT NULL	The absolute file number of the datafile.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN at the creation of the backup.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS in the datafile header.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS in the datafile header.
INCREMENTAL_CHANGE#	NUMBER	NOT NULL	The SCN that determines whether a block will be included in the incremental backup. A block is only included if the SCN in the block header is greater than or equal to INCREMENTAL_CHANGE#. The range of redo covered by the incremental backup begins with INCREMENTAL_CHANGE# and ends with CHECKPOINT_CHANGE#.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent datafile checkpoint.
CHECKPOINT_TIME	DATE	NOT NULL	The time of the last datafile checkpoint.
ABSOLUTE_FUZZY_CHANGE#	NUMBER		The absolute fuzzy SCN.
DATAFILE_BLOCKS	NUMBER	NOT NULL	The number of data blocks in the datafile.
BLOCKS	NUMBER	NOT NULL	The number of data blocks written to the backup.
BLOCK_SIZE	NUMBER	NOT NULL	The size of the data blocks in bytes.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (all pieces available), D (all pieces deleted), O (some pieces are available but others are not, so the backup set is unusable).

RC_BACKUP_PIECE

This view lists information about backup pieces. This view corresponds to the V\$BACKUP_PIECE dynamic performance view. Each backup set contains one or more backup pieces.

Multiple copies of the same backup piece can exist, but each copy will have its own record in the control file and its own row in the view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DB_ID	NUMBER	NOT NULL	The database identifier.
BP_KEY	NUMBER	NOT NULL	The primary key for the backup piece in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The backup piece RECID from V\$BACKUP_PIECE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The backup piece stamp propagated from V\$BACKUP_PIECE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set to which this record belongs in the recovery catalog. Use this column to form a join with RC_BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
BACKUP_TYPE	VARCHAR2(1)	NOT NULL	The type of the backup: D (full or level 0 incremental), I (incremental level 1 or higher), L (archived redo log).
INCREMENTAL_LEVEL	NUMBER		The level of the incremental backup: NULL or 0 - 4.
PIECE#	NUMBER	NOT NULL	The number of the backup piece. The first piece has the value of 1.
COPY#	NUMBER	NOT NULL	The copy number of the backup piece.

Column	Datatype	NULL	Description
DEVICE_TYPE	VARCHAR2(255)	NOT NULL	The type of backup device: DISK or SBT_TAPE (sequential media).
HANDLE	VARCHAR2(1024)	NOT NULL	The filename of the backup piece. It is the value that RMAN passes to the OSD layer that identifies the file.
COMMENTS	VARCHAR2(255)		Comments about the backup piece.
MEDIA	VARCHAR2(80)		A comment that contains further information about the media manager that created this backup.
MEDIA_POOL	NUMBER		The number of the media pool in which the backup is stored.
CONCUR	VARCHAR2(3)		Specifies whether backup media supports concurrent access: YES or NO.
TAG	VARCHAR2(32)		The user-specified tag for the backup piece.
START_TIME	DATE	NOT NULL	The time when RMAN started to write the backup piece.
COMPLETION_TIME	DATE	NOT NULL	The time when the backup piece was completed.
ELAPSED_SECONDS	NUMBER		The duration of the creation of the backup piece.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup piece: A (available), U (unavailable), D (deleted), or X (expired).

RC_BACKUP_REDOLOG

This view lists information about archived redo logs in backup sets. It corresponds to the V\$BACKUP_REDOLOG dynamic performance view in the control file.

You cannot back up online logs directly; you must first archive them to disk and then back them up. An archived log backup set contains one or more archived logs.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
BRL_KEY	NUMBER	NOT NULL	The primary key of the archived redo log in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.

Column	Datatype	NULL	Description
RECID	NUMBER	NOT NULL	The record identifier propagated from V\$BACKUP_REDOLOG. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The stamp from V\$BACKUP_REDOLOG. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set to which this record belongs in the recovery catalog. Use this column to form a join with RC_BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies the backup set to which this record belongs in the target database control file.
BACKUP_TYPE	VARCHAR2(1)	NOT NULL	The type of the backup: L (archived redo log).
COMPLETION_TIME	DATE	NOT NULL	The time when the backup completed.
THREAD#	NUMBER	NOT NULL	The thread number of the redo log.
SEQUENCE#	NUMBER	NOT NULL	The log sequence number.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS when the record was created.
FIRST_CHANGE#	NUMBER	NOT NULL	The SCN generated when Oracle switched into the redo log.
FIRST_TIME	DATE	NOT NULL	The time when Oracle switched into the redo log.
NEXT_CHANGE#	NUMBER	NOT NULL	The first SCN of the next redo log in the thread.
NEXT_TIME	DATE	NOT NULL	The first timestamp of the next redo log in the thread.
BLOCKS	NUMBER	NOT NULL	The number of blocks written to the backup.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (all pieces available), D (all pieces deleted), O (some pieces are available but others are not, so the backup set is unusable).

RC_BACKUP_SET

This view lists information about backup sets for all incarnations of the database. It corresponds to the V\$BACKUP_SET dynamic performance view. A backup set record is inserted after the backup has successfully completed.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DB_ID	NUMBER	NOT NULL	The unique database identifier.
BS_KEY	NUMBER	NOT NULL	The primary key of the backup set in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The backup set RECID from V\$BACKUP_SET. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file. Use either RECID and STAMP or SET_STAMP and SET_COUNT to access V\$BACKUP_SET.
STAMP	NUMBER	NOT NULL	The backup set STAMP from V\$BACKUP_SET. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file. Use either RECID and STAMP or SET_STAMP and SET_COUNT to access V\$BACKUP_SET.
SET_STAMP	NUMBER	NOT NULL	The SET_STAMP value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies this record in the target database control file. Use either RECID and STAMP or SET_STAMP and SET_COUNT to access V\$BACKUP_SET.
SET_COUNT	NUMBER	NOT NULL	The SET_COUNT value from V\$BACKUP_SET. SET_STAMP and SET_COUNT form a concatenated key that uniquely identifies this record in the target database control file. Use either RECID and STAMP or SET_STAMP and SET_COUNT to access V\$BACKUP_SET.
BACKUP_TYPE	VARCHAR2(1)	NOT NULL	The type of the backup: D (full or level 0 incremental), I (incremental level 1 or higher), L (archived redo log).
INCREMENTAL_LEVEL	NUMBER	NOT NULL	The level of the incremental backup: NULL or 0 - 4.
PIECES	NUMBER	NOT NULL	The number of backup pieces in the backup set.
START_TIME	DATE	NOT NULL	The time when the backup began.
COMPLETION_TIME	DATE	NOT NULL	The time when the backup completed.
ELAPSED_SECONDS	NUMBER		The duration of the backup in seconds.

RC_CHECKPOINT

Column	Datatype	NULL	Description
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (all pieces available), D (all pieces deleted), O (some pieces are available but others are not, so the backup set is unusable).

RC_CHECKPOINT

This view lists recovery catalog resynchronization information. It does not give information about database checkpoints. In most cases, you should use RC_RESYNC instead.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
CKP_KEY	NUMBER	NOT NULL	The primary key for the checkpoint.
CKP_SCN	NUMBER	NOT NULL	The control file checkpoint SCN.
CKP_CF_SEQ	NUMBER	NOT NULL	The control file sequence number.
CKP_TIME	DATE		The time at which the catalog was checkpointed.
CKP_TYPE	VARCHAR2(7)	NOT NULL	The type of resynchronization: FULL or PARTIAL.

RC_CONTROLFILE_COPY

This view lists information about control file copies on disk. Note that a datafile copy record with a file number of 0 is used to represent the control file copy in the V\$DATAFILE_COPY dynamic performance view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.

Column	Datatype	NULL	Description
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
CCF_KEY	NUMBER	NOT NULL	The primary key of the control file copy in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The record identifier from V\$DATAFILE_COPY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The stamp from V\$DATAFILE_COPY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
NAME	VARCHAR2(1024)	NOT NULL	The control file copy filename.
TAG	VARCHAR2(32)		The tag of the control file copy. NULL if no tag used.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS when the record was created.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	The control file checkpoint SCN.
CHECKPOINT_TIME	DATE	NOT NULL	The control file checkpoint time.
CREATION_TIME	DATE	NOT NULL	The control file creation time.
BLOCK_SIZE	NUMBER	NOT NULL	The block size in bytes.
MIN_OFFR_RECID	NUMBER	NOT NULL	Internal use only.
OLDEST_OFFLINE_RANGE	NUMBER	NOT NULL	Internal use only.
COMPLETION_TIME	DATE	NOT NULL	The time when the copy was generated.
STATUS	VARCHAR2(1)	NOT NULL	The status of the copy: A (available), U (unavailable), or D (deleted).

RC_COPY_CORRUPTION

This view lists corrupt block ranges in datafile copies. It corresponds to the V\$COPY_CORRUPTION dynamic performance view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
RECID	NUMBER	NOT NULL	The record identifier from V\$COPY_CORRUPTION. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The stamp from V\$COPY_CORRUPTION. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
CDF_KEY	NUMBER	NOT NULL	The primary key of the datafile copy in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output. Use this column to form a join with RC_DATAFILE_COPY.
COPY_RECID	NUMBER	NOT NULL	The RECID from RC_DATAFILE_COPY. This value is propagated from the control file.
COPY_STAMP	NUMBER	NOT NULL	The STAMP from RC_DATAFILE_COPY. This value is propagated from the control file.
FILE#	NUMBER	NOT NULL	The absolute file number of the datafile.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN recorded at the creation of the copy.
BLOCK#	NUMBER	NOT NULL	The block number of the first corrupted block in the file.
BLOCKS	NUMBER	NOT NULL	The number of corrupted blocks found beginning with BLOCK#.
CORRUPTION_CHANGE#	NUMBER		The SCN at which corruption was detected.
MARKED_CORRUPT	VARCHAR2(3)		YES if this corruption was not previously detected by Oracle or NO if it was already known by Oracle.

RC_DATABASE

This view gives information about the databases registered in the recovery catalog.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER		The primary key for the current incarnation. Use this column to form a join with RC_DATABASE_INCARNATION.
DBID	NUMBER	NOT NULL	Unique identifier for the database obtained from V\$DATABASE.
NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database for the current incarnation.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS operation when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS operation when the record was created.

RC_DATABASE_INCARNATION

This view lists information about all database incarnations registered in the recovery catalog. Oracle creates a new incarnation whenever you open a database with the RESETLOGS option.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the database. Use this column to form a join with almost any other catalog view.
DBID	NUMBER	NOT NULL	Unique identifier for the database.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation.
NAME	VARCHAR2(8)	NOT NULL	The DB_NAME for the database at the time of the RESETLOGS operation.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the RESETLOGS operation that created this incarnation.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the RESETLOGS operation that created this incarnation.
CURRENT_INCARNATION	VARCHAR2(3)		YES if it is the current incarnation; NO if it is not.
PARENT_DBINC_KEY	NUMBER		The DBINC_KEY of the previous incarnation for this database. The value is NULL if it is the first incarnation recorded for the database.

RC_DATAFILE

This view lists information about all datafiles registered in the recovery catalog. It corresponds to the V\$DATAFILE dynamic performance view. A datafile is shown as dropped if its tablespace was dropped.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
TS#	NUMBER	NOT NULL	The tablespace identifier in the target database. The TS# may exist multiple times in the same incarnation if the tablespace is dropped and re-created.
TABLESPACE_NAME	VARCHAR2(30)	NOT NULL	The tablespace name. The name may exist multiple times in the same incarnation if the tablespace is dropped and re-created.
FILE#	NUMBER	NOT NULL	The absolute file number of the datafile. The same datafile number may exist multiple times in the same incarnation if the datafile is dropped and re-created.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN at datafile creation.
CREATION_TIME	DATE		The time of datafile creation.
DROP_CHANGE#	NUMBER		The SCN recorded when the datafile was dropped. If a new datafile with the same file number is discovered then the DROP_CHANGE# is set to CREATION_CHANGE# for the datafile; otherwise the value is set to RC_CHECKPOINT.CKP_SCN.
DROP_TIME	DATE		The time when the datafile was dropped. If a new datafile with the same file number is discovered then the DROP_TIME is set to CREATION_TIME for the datafile; otherwise the value is set to RC_CHECKPOINT.CKP_TIME.
BYTES	NUMBER		The size of the datafile in bytes.
BLOCKS	NUMBER		The number of blocks in the datafile.
BLOCK_SIZE	NUMBER	NOT NULL	The size of the data blocks.
NAME	VARCHAR2(1024)		The datafile filename.
STOP_CHANGE#	NUMBER		SCN for datafile if offline normal or read-only.
READ_ONLY	NUMBER	NOT NULL	1 if STOP_CHANGE# is read-only; otherwise 0.

RC_DATAFILE_COPY

This view lists information about datafile copies on disk. It corresponds to the V\$DATAFILE_COPY dynamic performance view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
CDF_KEY	NUMBER	NOT NULL	The primary key of the datafile copy in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The datafile copy record from V\$DATAFILE_COPY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The datafile copy stamp from V\$DATAFILE_COPY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
NAME	VARCHAR2(1024)	NOT NULL	The filename of the datafile copy.
TAG	VARCHAR2(32)		The tag for the datafile copy.
FILE#	NUMBER	NOT NULL	The absolute file number for the datafile.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN for the datafile copy creation.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS in the datafile header.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS in the datafile header.
INCREMENTAL_LEVEL	NUMBER		The incremental level of the copy: 0 or NULL.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent datafile checkpoint.
CHECKPOINT_TIME	DATE	NOT NULL	The time of the most recent datafile checkpoint.
ABSOLUTE_FUZZY_CHANGE#	NUMBER		The highest SCN in any block of the file, if known.

RC_LOG_HISTORY

Column	Datatype	NULL	Description
RECOVERY_FUZZY_CHANGE#	NUMBER		The SCN to which recovery must proceed for the file to become not fuzzy. If not NULL, this file must be recovered at least to the specified SCN before the database can be opened with this file.
RECOVERY_FUZZY_TIME	DATE		The time to which recovery must proceed for the file to become not fuzzy. If not NULL, this file must be recovered at least to the specified time before the database can be opened with this file.
ONLINE_FUZZY	VARCHAR2(3)		YES/NO. If set to YES, this copy was made after a crash or OFFLINE IMMEDIATE (or is a copy of a copy that was taken improperly while the database was open). Recovery will need to apply all redo up to the next crash recovery marker to make the file consistent.
BACKUP_FUZZY	VARCHAR2(3)		YES/NO. If set to YES, this is a copy taken using the BEGIN BACKUP/END BACKUP technique. To make this copy consistent, Recovery will need to apply all redo up to the marker that is placed in the redo stream when the ALTER TABLESPACE END BACKUP command is used.
BLOCKS	NUMBER	NOT NULL	The number of blocks in the datafile copy (also the size of the datafile when the copy was made).
BLOCK_SIZE	NUMBER	NOT NULL	The size of the blocks in bytes.
COMPLETION_TIME			The time when the copy completed.
STATUS	VARCHAR2(1)	NOT NULL	The status of the copy: A (available), U (unavailable), or D (deleted).

RC_LOG_HISTORY

This view lists historical information about the online redo logs. It corresponds to the VSLOG_HISTORY dynamic performance view.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.

Column	Datatype	NULL	Description
RECID	NUMBER	NOT NULL	The redo log history RECID from VSLOG_HISTORY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The redo log history stamp from VSLOG_HISTORY. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
THREAD#	NUMBER	NOT NULL	The thread number of the online redo log.
SEQUENCE#	NUMBER	NOT NULL	The log sequence number of the redo log.
FIRST_CHANGE#	NUMBER	NOT NULL	The SCN generated when switching into the redo log.
FIRST_TIME	DATE	NOT NULL	The timestamp when switching into the redo log.
NEXT_CHANGE#	NUMBER	NOT NULL	The first SCN of the next redo log in the thread.
CLEARED	VARCHAR2(3)		'?' if the redo log was cleared with the ALTER DATABASE CLEAR LOGFILE statement; otherwise, NULL. This statement allows a log to be dropped without archiving it first.

RC_OFFLINE_RANGE

This view lists the offline ranges for datafiles. It corresponds to the V\$OFFLINE_RANGE dynamic performance view.

An offline range is created for a datafile when its tablespace is first altered to be offline normal or read-only, and then subsequently altered to be online or read-write. Note that no offline range is created if the datafile itself is altered to be offline or if the tablespace is altered to be offline immediate.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
RECID	NUMBER	NOT NULL	The record identifier for the offline range from V\$OFFLINE_RANGE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.

Column	Datatype	NULL	Description
STAMP	NUMBER	NOT NULL	The stamp for the offline range from V\$OFFLINE_RANGE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
FILE#	NUMBER	NOT NULL	The absolute file number of the datafile.
CREATION_CHANGE#	NUMBER	NOT NULL	The SCN at datafile creation.
OFFLINE_CHANGE#	NUMBER	NOT NULL	The SCN taken when the datafile was taken offline.
ONLINE_CHANGE#	NUMBER	NOT NULL	The online checkpoint SCN.
ONLINE_TIME	DATE	NOT NULL	The online checkpoint time.
CF_CREATE_TIME	DATE		The time of control file creation.

RC_PROXY_CONTROLFILE

This view contains descriptions of control file backups that were taken with proxy copy. In a proxy copy, the media manager takes over the operations of backing up and restoring data. Each row represents a backup of one control file. This view corresponds to V\$PROXY_DATAFILE.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
XDF_KEY	NUMBER	NOT NULL	The proxy copy primary key in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The proxy copy record identifier from V\$PROXY_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The proxy copy stamp from V\$PROXY_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
TAG	VARCHAR2(32)		The tag for the proxy copy.

Column	Datatype	NULL	Description
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS when the record was created.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS when the record was created.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	Checkpoint SCN when the copy was made.
CHECKPOINT_TIME	DATE	NOT NULL	Checkpoint time when the copy was made.
CREATION_TIME	DATE	NOT NULL	The control file creation time.
BLOCK_SIZE	NUMBER	NOT NULL	The block size for the copy in bytes.
MIN_OFFR_RECID	NUMBER	NOT NULL	Internal use only.
OLDEST_OFFLINE_RANGE	NUMBER	NOT NULL	Internal use only.
DEVICE_TYPE	VARCHAR2(255)	NOT NULL	The type of sequential media device.
HANDLE	VARCHAR2(1024)	NOT NULL	The filename for the proxy copy. It is the value that RMAN passes to the OSD layer that identifies the file.
COMMENTS	VARCHAR2(255)		Comments about the proxy copy.
MEDIA	VARCHAR2(80)		A comment that contains further information about the media manager that created this backup.
MEDIA_POOL	NUMBER		The number of the media pool in which the proxy copy is stored.
START_TIME	DATE	NOT NULL	The time when proxy copy was initiated.
COMPLETION_TIME	DATE	NOT NULL	The time when the proxy copy was completed.
ELAPSED_SECONDS	NUMBER		The duration of the proxy copy.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (available), U (unavailable), X (expired), or D (deleted).

RC_PROXY_DATAFILE

This view contains descriptions of datafile backups that were taken with proxy copy. It corresponds to the VSPROXY_DATAFILE dynamic performance view. In a proxy copy, the media manager takes over the operations of backing up and restoring data. Each row represents a backup of one database file.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
XDF_KEY	NUMBER	NOT NULL	The proxy copy primary key in the recovery catalog. If you issue the list command while connected to the recovery catalog, this value appears in the KEY column of the output.
RECID	NUMBER	NOT NULL	The proxy copy record identifier from VSPROXY_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
STAMP	NUMBER	NOT NULL	The proxy copy stamp from VSPROXY_DATAFILE. RECID and STAMP form a concatenated primary key that uniquely identifies this record in the target database control file.
TAG	VARCHAR2(32)		The tag for the proxy copy.
FILE#	NUMBER	NOT NULL	The absolute file number of the datafile that is proxy copied.
CREATION_CHANGE#	NUMBER	NOT NULL	The datafile creation SCN.
RESETLOGS_CHANGE#	NUMBER	NOT NULL	The SCN of the most recent RESETLOGS in the datafile header.
RESETLOGS_TIME	DATE	NOT NULL	The timestamp of the most recent RESETLOGS in the datafile header.
INCREMENTAL_LEVEL	NUMBER		0 if this copy is part of an incremental backup strategy, otherwise NULL.
CHECKPOINT_CHANGE#	NUMBER	NOT NULL	Checkpoint SCN when the copy was made.
CHECKPOINT_TIME	DATE	NOT NULL	Checkpoint time when the copy was made.
ABSOLUTE_FUZZY_CHANGE#	NUMBER		The highest SCN in any block of the file, if known.

Column	Datatype	NULL	Description
RECOVERY_FUZZY_CHANGE#	NUMBER		The SCN to which recovery must proceed for the file to become not fuzzy. If not NULL, this file must be recovered at least to the specified SCN before the database can be opened with this file.
RECOVERY_FUZZY_TIME	DATE		The timestamp to which recovery must proceed for the file to become not fuzzy. If not NULL, this file must be recovered at least to the specified time before the database can be opened with this file.
ONLINE_FUZZY	VARCHAR2(3)		YES/NO. If set to YES, this copy was made after a crash or offline immediate (or is a copy of a copy which was taken improperly while the database was open). Recovery will need to apply all redo up to the next crash recovery marker to make the file consistent.
BACKUP_FUZZY	VARCHAR2(3)		YES/NO. If set to YES, this is a copy taken using the BEGIN BACKUP/END BACKUP technique. To make this copy consistent, Recovery will need to apply all redo up to the marker that is placed in the redo stream when the ALTER TABLESPACE END BACKUP command is used.
BLOCKS	NUMBER	NOT NULL	Size of the datafile copy in blocks (also the size of the datafile when the copy was made).
BLOCK_SIZE	NUMBER	NOT NULL	The block size for the copy in bytes.
DEVICE_TYPE	VARCHAR2(255)	NOT NULL	The type of sequential media device.
HANDLE	VARCHAR2(1024)	NOT NULL	The filename for the proxy copy. It is the value that RMAN passes to the OSD layer that identifies the file.
COMMENTS	VARCHAR2(255)		Comments about the proxy copy.
MEDIA	VARCHAR2(80)		A comment that contains further information about the media manager that created this backup.
MEDIA_POOL	NUMBER		The number of the media pool in which the proxy copy is stored.
START_TIME	DATE	NOT NULL	The time when proxy copy was initiated.
COMPLETION_TIME	DATE	NOT NULL	The time when the proxy copy was completed.
ELAPSED_SECONDS	NUMBER		The duration of the proxy copy.
STATUS	VARCHAR2(1)	NOT NULL	The status of the backup set: A (available), U (unavailable), X (expired), or D (deleted).

RC_REDO_LOG

This view lists information about the online redo logs for all incarnations of the database.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
THREAD#	NUMBER	NOT NULL	The number of the redo thread.
GROUP#	NUMBER	NOT NULL	The number of the online redo log group.
NAME	VARCHAR2(1024)	NOT NULL	The name of the online redo log file.

RC_REDO_THREAD

This view lists data about all redo threads for all incarnations of the database.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
THREAD#	NUMBER	NOT NULL	The redo thread number for the database incarnation.
STATUS	VARCHAR2(1)	NOT NULL	The status of the redo thread: D (disabled), E (enabled), or O (open).
SEQUENCE#	NUMBER	NOT NULL	The last allocated log sequence number.
ENABLE_CHANGE#	NUMBER		The SCN at which this thread was enabled.
ENABLE_TIME	DATE		The time at which this thread was enabled.
DISABLE_CHANGE#	NUMBER		The SCN of the last disabled thread.
DISABLE_TIME	DATE		The time of the last disabled thread.

RC_RESYNC

This view lists information about recovery catalog resynchronizations. Every full resynchronization takes a snapshot of the target database control file and resynchronizes the recovery catalog from the snapshot.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
RESYNC_KEY	NUMBER	NOT NULL	The primary key for the resynchronization.
CONTROLFILE_CHANGE#	NUMBER	NOT NULL	The control file checkpoint SCN from which the catalog was resynchronized.
CONTROLFILE_TIME	DATE		The control file checkpoint timestamp from which the catalog was resynchronized.
CONTROLFILE_SEQUENCE#	NUMBER	NOT NULL	The control file sequence number.
CONTROLFILE_VERSION	DATE	NOT NULL	The creation time for the version of the control file from which the catalog was resynchronized.
RESYNC_TYPE	VARCHAR2(7)	NOT NULL	The type of resynchronization: FULL or PARTIAL.
DB_STATUS	VARCHAR2(7)		The status of the target database: OPEN or MOUNTED.
RESYNC_TIME	DATE	NOT NULL	The time of the resynchronization.

RC_STORED_SCRIPT

This view lists information about scripts stored in the recovery catalog. The view contains one row for each stored script.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the database that owns this script. Use this column to form a join with almost any other catalog view.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
SCRIPT_NAME	VARCHAR2(100)	NOT NULL	The name of the script.

RC_STORED_SCRIPT_LINE

This view lists information about lines of the scripts stored in the recovery catalog. The view contains one row for each line of each stored script.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the database that owns this script. Use this column to form a join with almost any other catalog view.
SCRIPT_NAME	VARCHAR2(100)	NOT NULL	The name of the stored script.
LINE	NUMBER	NOT NULL	The number of the line in the script. The line of a script is uniquely identified by SCRIPT_NAME and LINE.
TEXT	VARCHAR2(1024)	NOT NULL	The text of the line of the script.

RC_TABLESPACE

This view lists information about all tablespaces registered in the recovery catalog, all dropped tablespaces, and tablespaces that belong to old database incarnations. It corresponds to the V\$TABLESPACE dynamic performance view. The current value is shown for tablespace attributes.

Column	Datatype	NULL	Description
DB_KEY	NUMBER	NOT NULL	The primary key for the target database. Use this column to form a join with almost any other catalog view.
DBINC_KEY	NUMBER	NOT NULL	The primary key for the incarnation of the target database. Use this column to form a join with RC_DATABASE_INCARNATION.
DB_NAME	VARCHAR2(8)	NOT NULL	The DB_NAME of the database incarnation to which this record belongs.
TS#	NUMBER	NOT NULL	The tablespace identifier in the target database. The TS# may exist multiple times in the same incarnation if the tablespace is dropped and re-created.
NAME	VARCHAR2(30)	NOT NULL	The tablespace name. The name may exist multiple times in the same incarnation if the tablespace is dropped and re-created.
CREATION_CHANGE#	NUMBER	NOT NULL	The creation SCN (from the first datafile).
CREATION_TIME	DATE		The creation time of the tablespace. NULL for offline tablespaces after creating the control file.

Column	Datatype	NULL	Description
DROP_CHANGE#	NUMBER		The SCN recorded when the tablespace was dropped. If a new tablespace with the same TS# is discovered then the DROP_CHANGE# is set to CREATION_CHANGE# for the tablespace; otherwise, the value is set to RC_CHECKPOINT.CKP_SCN.
DROP_TIME	DATE		The date when the tablespace was dropped.

Part IV

Performing Operating System Backup and Recovery

Performing Operating System Backups

If you do not use Recovery Manager, you can make backups of your database using operating system utilities and recover datafiles using SQL*Plus. This chapter explains how to use O/S methods to back up an Oracle database, and includes the following topics:

- [Listing Database Files Before Performing a Backup](#)
- [Performing O/S Backups](#)
- [Recovering From a Failed Online Tablespace Backup](#)
- [Using Export and Import for Supplemental Protection](#)

Listing Database Files Before Performing a Backup

Before taking a backup, identify all the files in your database. Then ascertain what you need to back up.

To list datafiles, online redo logs, and control files:

1. Start SQL*Plus and issue a SELECT on V\$DATAFILE to obtain a list of datafiles:

```
SQL> SELECT name FROM v$datafile;
NAME
-----
/vobs/oracle/dbs/tbs_01.f
/vobs/oracle/dbs/tbs_02.f
/vobs/oracle/dbs/tbs_03.f
/vobs/oracle/dbs/tbs_11.f
/vobs/oracle/dbs/tbs_12.f
/vobs/oracle/dbs/tbs_21.f
/vobs/oracle/dbs/tbs_22.f
/vobs/oracle/dbs/tbs_23.f
/vobs/oracle/dbs/tbs_24.f
9 rows selected.
```

You can also join the V\$TABLESPACE and V\$DATAFILE views to obtain a listing of datafiles along with their associated tablespaces:

```
SQL> SELECT t.name "Tablespace", f.name "Datafile"
2> FROM v$tablespace t, v$datafile f
3> WHERE t.ts# = f.ts#
4> ORDER BY t.name;
```

Tablespace	Datafile
-----	-----
SYSTEM	/vobs/oracle/dbs/tbs_01.f
SYSTEM	/vobs/oracle/dbs/tbs_02.f
SYSTEM	/vobs/oracle/dbs/tbs_03.f
TBS_1	/vobs/oracle/dbs/tbs_11.f
TBS_1	/vobs/oracle/dbs/tbs_12.f
TBS_2	/vobs/oracle/dbs/tbs_21.f
TBS_2	/vobs/oracle/dbs/tbs_22.f
TBS_2	/vobs/oracle/dbs/tbs_23.f
TBS_2	/vobs/oracle/dbs/tbs_24.f

2. Obtain a list of online redo log files by using the V\$LOGFILE view. For example, issue this query:

```
SELECT member FROM v$logfile;
MEMBER
-----
```

```

/vobs/oracle/dbs/t1_log1.f
/vobs/oracle/dbs/t1_log2.f
2 rows selected.

```

3. Obtain the names of the current control files using the `CONTROL_FILES` parameter. For example, issue this query:

```

SHOW PARAMETERS control_files;
NAME                                TYPE      VALUE
-----                                -
control_files                        string    /vobs/oracle/dbs/cf1.f

```

4. If you plan to take a control file backup using the `ALTER DATABASE` command with the `BACKUP CONTROLFILE TO 'filename'` option, save a list of all datafiles and online redo log files with the control file backup.

Performing O/S Backups

While Recovery Manager is the recommended tool for backing up an Oracle database, you can also make backups using O/S utilities. The utility you choose is dependent on your operating system.

This section describes the various aspects of making O/S backups, and includes the following topics:

- [Performing Whole Database Backups](#)
- [Performing Tablespace and Datafile Backups](#)
- [Performing Control File Backups](#)

Performing Whole Database Backups

Take a whole database backup of all files that constitute a database after the database is shut down to system-wide use in normal priority. *A whole database backup taken while the database is open or after an instance crash or SHUTDOWN ABORT is inconsistent.* In such cases, the files are inconsistent with respect to the checkpoint SCN.

You can take a whole database backup if a database is operating in either ARCHIVELOG or NOARCHIVELOG mode. If you run the database in NOARCHIVELOG mode, however, the backup must be consistent, i.e., you must shut down the database cleanly before the backup.

The set of backup files that result from a consistent whole database backup are consistent because all files correspond to the same SCN. You can restore the

database without performing recovery. After restoring the backup files, you can perform additional recovery steps to recover the database to a more current time if the database is operated in ARCHIVELOG mode. Also, you can take inconsistent whole database backups if your database is in ARCHIVELOG mode.

Only use a backup control file created during a whole database backup to restore the other files taken in that backup, not for complete or incomplete database recovery. The reason is that Oracle recognizes backup control files created with the ALTER DATABASE BACKUP CONTROLFILE statement as backup control files; O/S copies of control files look like current control files to Oracle. Unless you are making a whole database backup, *always* back up the control file using a SQL statement.

See also: For more information about backing up control files, see "[Performing Control File Backups](#)" on page 13-11.

Making Consistent Whole Database Backups

To guarantee that a database's datafiles are consistent, shut down the database with the NORMAL, IMMEDIATE, or TRANSACTIONAL options before making a whole database backup. Never perform a whole database backup after an instance failure or after the database is shut down using a SHUTDOWN ABORT statement unless your database is in ARCHIVELOG mode.

To make a consistent whole database backup:

1. If the database is open, use SQL*Plus to shut down the database with the NORMAL, IMMEDIATE, or TRANSACTIONAL options:

```
SHUTDOWN NORMAL;  
SHUTDOWN IMMEDIATE;  
SHUTDOWN TRANSACTIONAL;
```

Do not make a whole database backup when the instance is aborted or stopped because of a failure. Reopen the database and shut it down cleanly first.

2. Use O/S commands or a backup utility to make backups of all datafiles, and a single control file of the database. Also back up the parameter files associated with the database.

For example, if all datafiles, control files, and parameter files are contained in /disk1/oracle/dbs on UNIX, you might back up the directory to /disk2/backup as follows:

```
% cp -r /disk1/oracle/dbs /disk2/backup
```


3. Restart the database

```
STARTUP;
```

See Also: For more information on starting up and shutting down a database, see the *Oracle8i Administrator's Guide*.

Verifying Backups

DBVERIFY is an external command-line utility that performs a physical data structure integrity check on an offline database. Use DBVERIFY primarily when you need to ensure that a backup database or datafile is valid before it is restored or as a diagnostic aid when you have encountered data corruption problems.

The name and location of DBVERIFY is dependent on your operating system. For example, to perform an integrity check on datafile `tbs_52.f` on UNIX, you can execute the **dbv** command as follows:

```
% dbv file=tbs_52.f
```

```
DBVERIFY: Release 8.1.5.0.0
```

```
(c) Copyright 1998 Oracle Corporation. All rights reserved.
```

```
DBVERIFY - Verification starting : FILE = tbs_52.f
```

```
DBVERIFY - Verification complete
```

```
Total Pages Examined          : 250
Total Pages Processed (Data)   : 4
Total Pages Failing (Data)     : 0
Total Pages Processed (Index)  : 15
Total Pages Failing (Index)    : 0
Total Pages Processed (Other)  : 29
Total Pages Empty              : 202
Total Pages Marked Corrupt     : 0
Total Pages Influx             : 0
```

See Also: For more information about using DBVERIFY, see *Oracle8i Utilities*.

Performing Tablespace and Datafile Backups

Only make tablespace and datafile backups when operating in ARCHIVELOG mode. You cannot use individual datafile backups to restore a database operating in NOARCHIVELOG mode because you do not have archived redo logs to recover the datafiles to the same point in time.

This section contains the topics:

- [Backing Up Online Tablespaces and Datafiles](#)
- [Backing Up Offline Tablespaces and Datafiles](#)

Backing Up Online Tablespaces and Datafiles

You can back up all or specified datafiles of an online tablespace while the database is open. When you back up an individual datafile or online tablespace, Oracle stops recording checkpoints in the headers of the online datafiles being backed up.

Oracle stops recording checkpoints as a direct result of issuing the ALTER TABLESPACE BEGIN BACKUP statement, which puts the database in *hot backup mode*. You must put the database in hot backup mode to make O/S backups when the database is open. After the backup is completed, Oracle advances the file header to the current database checkpoint, but only after you execute the ALTER TABLESPACE END BACKUP statement to take the tablespace out of hot backup mode.

When you restore a datafile, the header has a record of the most recent datafile checkpoint that occurred *before* the online tablespace backup, not any that occurred *during* it. As a result, Oracle asks for the appropriate set of redo log files to apply should recovery be needed.

To back up online tablespaces in an open database:

1. Before beginning a backup of a tablespace, identify all of the tablespace's datafiles using the DBA_DATA_FILES data dictionary view. For example, assume that you want to back up the USERS tablespace. Enter the following:

```
SELECT tablespace_name, file_name
FROM sys.dba_data_files
WHERE tablespace_name = 'USERS';
```

TABLESPACE_NAME	FILE_NAME
-----	-----
USERS	/oracle/dbs/tbs_21.f
USERS	/oracle/dbs/tbs_22.f

In this example, `/oracle/dbs/tbs_21.f` and `/oracle/dbs/tbs_22.f` are fully specified filenames corresponding to the datafiles of the USERS tablespace.

2. Mark the beginning of the online tablespace backup. For example, the following statement marks the start of an online backup for the tablespace USERS:

```
ALTER TABLESPACE users BEGIN BACKUP;
```

WARNING: If you forget to mark the beginning of an online tablespace backup, or neglect to assure that the `BEGIN BACKUP` command has completed before backing up an online tablespace, the backup datafiles are not useful for subsequent recovery operations. Attempting to recover such a backup is risky and can return errors that result in inconsistent data. For example, the attempted recovery operation will issue a "fuzzy files" warning, and lead to an inconsistent database that will not open.

3. Back up the online datafiles of the online tablespace using O/S commands. For example, UNIX users might enter:

```
% cp /oracle/dbs/tbs_21.f /oracle/backup/tbs_21.backup
% cp /oracle/dbs/tbs_22.f /oracle/backup/tbs_22.backup
```

4. After backing up the datafiles of the online tablespace, indicate the end of the online backup using the SQL command `ALTER TABLESPACE` with the `END BACKUP` option. For example, the following statement ends the online backup of the tablespace USERS:

```
ALTER TABLESPACE users END BACKUP;
```

If you forget to mark the end of an online tablespace backup, and an instance failure or `SHUTDOWN ABORT` occurs, Oracle assumes that media recovery (possibly requiring archived redo logs) is necessary at the next instance startup. To avoid performing media recovery in this case, use the `ALTER DATABASE` datafile *filename* `END BACKUP` statement.

See Also: See the *Oracle8i Reference* for more information about the `DBA_DATA_FILES` data dictionary view. For more information about startup and shutdown commands, see the *SQL*Plus User's Guide and Reference*.

Determining Datafile Backup Status To check the backup status of a datafile, query the `V$BACKUP` view. This view lists all online files and gives their backup status. It is most useful when the database is open. It is also useful immediately after a crash,

because it shows the backup status of the files at the time of the crash. Use this information to determine whether you have left tablespaces in hot backup mode.

V\$BACKUP is not useful if the control file currently in use is a restored backup or a new control file created after the media failure occurred. A restored or re-created control file does not contain the information Oracle needs to fill V\$BACKUP accurately. Also, if you have restored a backup of a file, that file's STATUS in V\$BACKUP reflects the backup status of the older version of the file, not the most current version. Thus, this view can contain misleading information on restored files.

For example, the following query displays the current backup status of datafiles:

```
SELECT file#, status FROM v$backup;
```

```
FILE#          STATUS
-----
0011           INACTIVE
0012           INACTIVE
0013           ACTIVE
...
```

In the STATUS column, INACTIVE indicates that the file is not currently being backed up, whereas ACTIVE indicates that the file is currently being backed up.

Backing Up Multiple Online Tablespaces When backing up several online tablespaces, use either of the following procedures:

To back up online tablespaces in parallel:

1. Prepare all online tablespaces for backup by issuing all necessary ALTER TABLESPACE statements at once. For example, put tablespaces TS1, TS2, and TS3 in hot backup mode:

```
ALTER TABLESPACE ts1 BEGIN BACKUP;
ALTER TABLESPACE ts2 BEGIN BACKUP;
ALTER TABLESPACE ts3 BEGIN BACKUP;
```

2. Back up all files of the online tablespaces. For example, a UNIX user might back up tbs_1.f, tbs_2.f, and tbs_3.f as follows:

```
% cp /oracle/dbs/tbs_1.f /oracle/backup/tbs_1.bak
% cp /oracle/dbs/tbs_2.f /oracle/backup/tbs_2.bak
% cp /oracle/dbs/tbs_3.f /oracle/backup/tbs_3.bak
```

3. Indicate that the online backups have been completed:

```
ALTER TABLESPACE ts1 END BACKUP;
ALTER TABLESPACE ts2 END BACKUP;
ALTER TABLESPACE ts3 END BACKUP;
```

To back up online tablespaces serially:

1. Prepare a tablespace for online backup. For example, to put tablespace TBS_1 in hot backup mode enter:

```
SQL> ALTER TABLESPACE tbs_1 BEGIN BACKUP;
```

2. Back up the datafiles in the tablespace. For example, enter:

```
% cp /oracle/dbs/tbs_1.f /oracle/backup/tbs_1.bak
```

3. Take the tablespace out of hot backup mode. For example, enter:

```
SQL> ALTER TABLESPACE tbs_1 END BACKUP;
```

4. Repeat this procedure for each remaining tablespace until you have backed up all the desired tablespaces.

Oracle recommends the serial option because it minimizes the time between ALTER TABLESPACE ... BEGIN/END BACKUP statements. During online backups, more redo information is generated for the tablespace.

Splitting Mirrors in Suspend Mode Sometimes systems allow you to mirror a disk or file, i.e., maintain an exact duplicate of the primary data in another location, and then *split the mirror*. Splitting the mirror involves separating the copies so that you can use them independently of one another.

Some RAID devices benefit from suspending writes while the split operation is occurring; your RAID vendor can advise you on whether your system would benefit from this feature.

The basic steps for using SUSPEND and RESUME are as follows:

1. Place the database tablespaces in hot backup mode using the ALTER TABLESPACE BEGIN BACKUP statement. For example, to place tablespace USERS in hot backup mode enter:

```
ALTER TABLESPACE users BEGIN BACKUP;
```

2. If your mirror system has problems with splitting a mirror while disk writes are occurring, issue the following:

```
ALTER SYSTEM SUSPEND;
```

3. Split your mirrors at the O/S or hardware level.
4. Issue the following:

```
ALTER SYSTEM RESUME;
```
5. Take the specified tablespaces out of hot backup mode. For example, to take tablespace USERS out of hot backup mode enter:

```
ALTER TABLESPACE users END BACKUP;
```
6. Copy the control file and archive the online redo logs as usual for a backup.

WARNING: Do not use the SUSPEND command as a substitute for placing a tablespace in hot backup mode.

See Also: For more information about the SUSPEND/RESUME feature, see the *Oracle8i Administrator's Guide*. For more information about the ALTER SYSTEM command with the RESUME and SUSPEND options, see the *Oracle8i SQL Reference*.

Backing Up Offline Tablespaces and Datafiles

You can back up all or some of the datafiles of an individual tablespace while the tablespace is offline. All other tablespaces of the database can remain open and available for system-wide use. You must have the MANAGE TABLESPACE system privilege to take tablespaces offline and online.

Note: You cannot take the SYSTEM tablespace or any tablespace with active rollback segments offline. The following procedure cannot be used for such tablespaces.

To back up offline tablespaces:

1. Before beginning a backup of a tablespace, identify the tablespace's datafiles using the DBA_DATA_FILES table. For example, assume that you want to back up the USERS tablespace. Enter the following:

```
SELECT tablespace_name, file_name  
FROM sys.dba_data_files  
WHERE tablespace_name = 'USERS';
```

TABLESPACE_NAME	FILE_NAME
-----	-----
USERS	/oracle/dbs/users.f

In this example, `/oracle/dbs/users.f` is a fully specified filename corresponding to the datafile in the USERS tablespace.

2. Take the tablespace offline using normal priority if possible. Normal priority is recommended because it guarantees that you can subsequently bring the tablespace online without the requirement for tablespace recovery. For example, the following statement takes a tablespace named USERS offline normally:

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

After you take a tablespace offline with normal priority, all datafiles of the tablespace are closed.

3. Back up the offline datafiles. For example, a UNIX user might enter the following to back up datafile `users.f`:

```
% cp /disk1/oracle/dbs/users.f /disk2/backup/users.backup
```

4. Bring the tablespace online. For example, the following statement brings tablespace USERS back online:

```
ALTER TABLESPACE users ONLINE;
```

Note: If you took the tablespace offline using temporary or immediate priority, you must not bring the tablespace online unless you perform tablespace recovery.

After you bring a tablespace online, it is open and available for use.

Performing Control File Backups

Back up the control file of a database after making a structural modification to a database operating in ARCHIVELOG mode. To back up a database's control file, you must have the ALTER DATABASE system privilege.

You have three options when backing up the control file:

- [Backing Up the Control File to a Physical File](#)
- [Backing Up the Control File to a Trace File](#)

Backing Up the Control File to a Physical File

The primary method for backing up the control file is to use a SQL statement to generate a binary file.

To back up the control file after a structural change:

1. Make the desired change to the database. For example, you might create a new datafile:

```
ALTER DATABASE CREATE DATAFILE '/oracle/dbs/tbs_20.f' AS '/oracle/dbs/tbs_4.f';
```

2. Back up the database's control file. The following SQL statement backs up a database's control file to `/oracle/backup/cf.bak`:

```
ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/cf.bak' REUSE;
```

The REUSE option allows you to have the new control file overwrite a control file that currently exists.

Backing Up the Control File to a Trace File

The TRACE option of the ALTER DATABASE BACKUP CONTROLFILE statement helps you manage and recover your control file. The TRACE option prompts Oracle to write SQL commands to the database's trace file rather than generate a physical backup. The statements in the trace file start the database, re-create the control file, and recover and open the database appropriately.

Each SQL statement in the trace file is commented. Thus, you can copy the commands from the trace file into a script file, edit them as necessary, and use the script to recover the database if all copies of the control file are lost (or to change the size of the control file). The trace file is located in the location specified by the USER_DUMP_DEST initialization parameter.

To back up the control file to a trace file, mount the database and issue the following statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

Creating a Trace File: Scenario Assume that you want to generate a script that will re-create the control file for the SALES database. The database has these characteristics:

- Three threads are enabled, of which thread 2 is public and thread 3 is private.
- The redo logs are multiplexed into three groups of two members each.
- The database has these datafiles:

- /diska/prod/sales/db/filea.dbf (offline datafile in online tablespace)
- /diska/prod/sales/db/database1.dbf (online)
- /diska/prod/sales/db/fileb.dbf (only file in read-only tablespace)

You issue the following statement to create the trace file:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS;
```

You then edit the trace file to create a script that creates a new control file based on the control file that was current when you generated the trace file. To avoid recovering offline normal or read-only tablespaces, omit them from the CREATE CONTROLFILE statement. At database open time, the dictionary check code will mark these files as MISSING. The RENAME command will rename them back to their filenames.

For example, the script might read as follows:

```
# The following statements will create a new control file and use it to open the database.
# No data other than log history will be lost. Additional logs may be required for media
# recovery of offline datafiles. Use this only if the current version of all online logs
# are available.

STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE SALES NORESETLOGS ARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 16
    MAXLOGHISTORY 1600
LOGFILE
    GROUP 1
        '/diska/prod/sales/db/log1t1.dbf',
        '/diskb/prod/sales/db/log1t2.dbf'
    ) SIZE 100K
    GROUP 2
        '/diska/prod/sales/db/log2t1.dbf',
        '/diskb/prod/sales/db/log2t2.dbf'
    ) SIZE 100K,
    GROUP 3
        '/diska/prod/sales/db/log3t1.dbf',
        '/diskb/prod/sales/db/log3t2.dbf'
    ) SIZE 100K
DATAFILE
    '/diska/prod/sales/db/database1.dbf',
    '/diskb/prod/sales/db/filea.dbf'
;
```

```
# This datafile is offline, but its tablespace is online. Take the datafile offline
# manually.
ALTER DATABASE DATAFILE '/diska/prod/sales/db/filea.dbf' OFFLINE;

# Recovery is required if any datafiles are restored backups,
# or if the most recent shutdown was not normal or immediate.
RECOVER DATABASE;

# All redo logs need archiving and a log switch is needed.
ALTER SYSTEM ARCHIVE LOG ALL;

# The database can now be opened normally.
ALTER DATABASE OPEN;

# The backup control file does not list read-only and normal offline tablespaces so that
# Oracle can avoid performing recovery on them. Oracle checks the data dictionary and
# finds information on these absent files and marks them 'MISSINGxxxx'. It then renames
# the missing files to acknowledge them without having to recover them.
ALTER DATABASE RENAME FILE 'MISSING0002'
    TO '/diska/prod/sales/db/fileb.dbf';
```

Using the command without `NORESETLOGS` produces the same output. Using the command with `RESETLOGS` produces a similar script that includes statements that recover and open the database, but resets the redo logs upon startup.

Recovering From a Failed Online Tablespace Backup

The following situations can cause a tablespace backup to fail and be incomplete:

- You did not indicate the end of the online tablespace backup operation using the `ALTER TABLESPACE` command with the `END BACKUP` option, and the database was subsequently shut down with the `ABORT` option.
- An instance or `SHUTDOWN ABORT` interrupted the backup.

Upon detecting an incomplete online tablespace backup at startup, Oracle assumes that media recovery is necessary for startup to proceed.

For example, Oracle may display:

```
SQL> startup
ORACLE instance started.
Total System Global Area                19839308 bytes
Fixed Size                               63820 bytes
Variable Size                           11042816 bytes
Database Buffers                        8192000 bytes
Redo Buffers                             540672 bytes
```

```
Database mounted.
ORA-01113: file 12 needs media recovery
ORA-01110: data file 12: '/oracle/dbs/tbs_41.f'
```

To avoid performing media recovery on a tablespace:

1. Use the V\$BACKUP view to list the datafiles of the tablespaces that were being backed up before the database was restarted:

```
SQL> SELECT * FROM v$backup WHERE status = 'ACTIVE';
FILE#          STATUS          CHANGE#          TIME
-----
          12 ACTIVE                20863 25-NOV-98
          13 ACTIVE                20863 25-NOV-98
          20 ACTIVE                20863 25-NOV-98
3 rows selected.
```

2. Issue the ALTER DATABASE DATAFILE ... END BACKUP statement to end the hot backup. For example, to take datafiles 12, 13, and 20 out of hot backup mode enter:

```
ALTER DATABASE DATAFILE 12,13,20 END BACKUP;
```

WARNING: Do not use ALTER DATABASE DATAFILE ... END BACKUP if you have restored any of the affected files from a backup.

3. Open the database:

```
ALTER DATABASE OPEN;
```

To recover the database without using the END BACKUP statement:

1. Mount the database:

```
STARTUP MOUNT;
```

2. Recover the database:

```
RECOVER DATABASE;
```

3. Use the V\$BACKUP view to confirm that there are no active datafiles:

```
SQL> SELECT * FROM v$backup WHERE status = 'ACTIVE';
FILE#          STATUS          CHANGE#          TIME
-----
0 rows selected.
```

See Also: For information on recovering a database, see [Chapter 14, "Performing Operating System Recovery"](#).

Using Export and Import for Supplemental Protection

Export and Import are utilities that move Oracle data in and out of Oracle databases. Export writes data from an Oracle database to an operating system file in a special binary format. Import reads Export files and restores the corresponding information into an existing database. Although Export and Import are designed for moving Oracle data, you can also use them to supplement backups of data.

This section describes the Import and Export utilities, and includes the following topics:

- [Using Export](#)
- [Using Import](#)

See Also: The Export and Import utilities are described in detail in *Oracle8i Utilities*.

Using Export

The Export utility allows you to back up your database while it is open and available for use. It writes a read-consistent view of the database's objects to an operating system file. System audit options are not exported.

WARNING: If you use Export to perform a backup, you must export all data in a logically consistent way so that the backup reflects a single point in time. No one should make changes to the database while the Export takes place. Ideally, you should run the database in restricted mode while you export the data, so no regular users can access the data.

[Table 13-1](#) lists available export modes.

Table 13–1 Export Modes

Mode	Description
User	Exports all objects owned by a user.
Table	Exports all or specific tables owned by a user.
Full Database	Exports all objects of the database.

Following are descriptions of Export types:

Incremental Export	<p>Only database data that has changed since the last incremental, cumulative, or complete export is exported. An incremental export exports the object's definition and all its data. Incremental exports are typically performed more often than cumulative or complete reports.</p> <p>For example, if tables A, B, and C exist, and only table A's information has been modified since the last incremental export, only table A is exported.</p>
Cumulative Exports	<p>Only database data that has been changed since the last cumulative or complete export is exported.</p> <p>Perform this type of export on a limited basis, such as once a week, to condense the information contained in numerous incremental exports.</p> <p>For example, if tables A, B, and C exist, and only table A's and table B's information has been modified since the last cumulative export, only the changes to tables A and B are exported.</p>
Complete Exports	<p>All database data is exported.</p> <p>Perform this type of export on a limited basis, such as once a month, to export all data contained in a database.</p>

Using Import

The Import utility allows you to restore the database information held in previously created Export files. It is the complement utility to Export.

To recover a database using Export files and the Import utility:

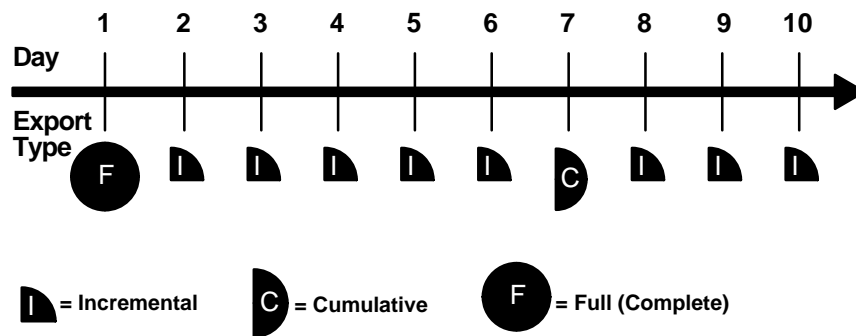
1. Re-create the database structure, including all tablespaces and users.

Note: These re-created structures should not have objects in them.

2. Import the appropriate Export files to restore the database to the most current state possible. Depending on how your Export schedule is performed, imports of varying degrees will be necessary to restore a database.

Assume that the schedule illustrated in [Figure 13-1](#) is used in exporting data from an Oracle database

Figure 13-1 A Typical Export Schedule



A complete export was taken on Day 1, a cumulative export was taken every week, and incremental exports were taken daily. Follow these steps to recover:

1. Recreate the database, including all tablespaces and users.
2. Import the complete database export taken on Day 1.
3. Import the cumulative database export taken on Day 7.
4. Import the incremental database exports taken on Days 8, 9, and 10.

Performing Operating System Recovery

This chapter describes how to recover a database, and includes the following topics:

- [What Is Media Recovery?](#)
- [Determining Which Files to Recover](#)
- [Restoring Files](#)
- [Understanding Basic Media Recovery Procedures](#)
- [Performing Complete Media Recovery](#)
- [Performing Incomplete Media Recovery](#)
- [Opening the Database after Media Recovery](#)

What Is Media Recovery?

Media recovery is the recovery of a database to a specific time. It involves these basic stages:

- [Restoring Files](#)
- [Recovering Datafiles](#)

Restoring Files

The first step in media recovery is to restore database files. Restoring involves reconstructing an original copy of a files or files from a backup. For example, assume that you query V\$DATAFILE to see your datafiles:

```
SQL> SELECT name FROM v$datafile;  
NAME
```

```
-----  
/oracle/dbs/tbs_01.f  
/oracle/dbs/tbs_02.f  
/oracle/dbs/tbs_03.f  
/oracle/dbs/tbs_11.f  
/oracle/dbs/tbs_12.f  
/oracle/dbs/tbs_21.f  
/oracle/dbs/tbs_22.f  
7 rows selected.
```

Restoring these datafiles involves using O/S commands to copy backups of the datafiles to their default locations, overwriting the files with the same names currently there, or to new locations. For example, to restore datafile `tbs_12.f` from backup `tbs_12.bak`, you might enter:

```
% cp /oracle/backup/tbs_12.bak /oracle/dbs/tbs_12.f
```

Recovering Datafiles

The second step is to perform media recovery on the specified datafiles. Media recovery is the application of online and archived redo logs to restored datafiles in order to update them to a current or non-current time. Media recovery is *complete* when you use all available redo to recover the database to the most current SCN; it is *incomplete* when you do not.

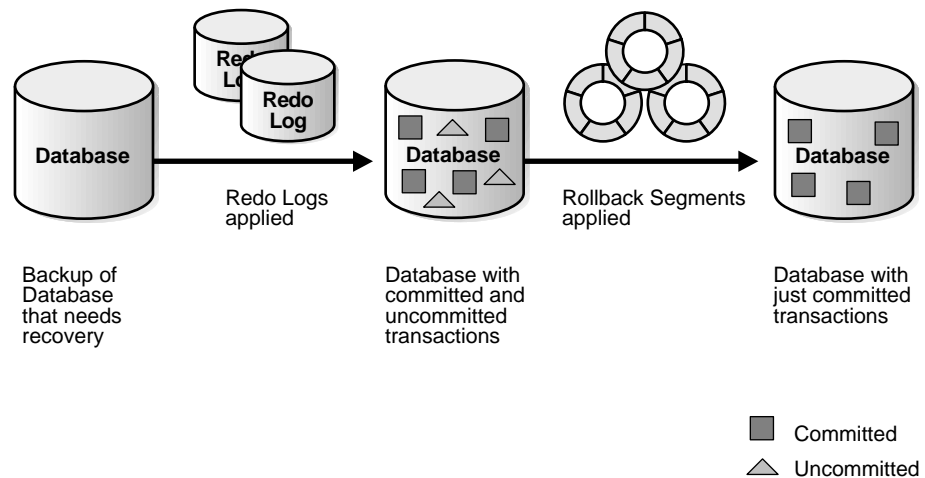
Oracle recovery occurs in two phases:

- Rolling forward
- Rolling back

During the roll forward phase of recovery, Oracle applies the changes recorded in the redo log records—either online, archived, or both—to the datafiles. Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time. After rolling forward, the data blocks contain committed as well as uncommitted changes.

During the rolling back phase, Oracle uses rollback segments to undo the effects of uncommitted transactions applied during the rolling forward phase. Oracle uses rollback segments to identify those transactions that were never committed, yet were recorded in the redo log. [Figure 14-1](#) shows the two stages of rolling forward and rolling back.

Figure 14-1 Basic Recovery Procedure



Determining Which Files to Recover

You can often use the table `V$RECOVER_FILE` to determine which files to recover. This view lists all files that need to be recovered, and explains why they need to be recovered.

The following query displays the file ID numbers of datafiles that require media recovery as well as the reason for recovery (if known) and the SCN/time when recovery needs to begin:

```
SQL> SELECT * FROM v$recover_file;
```

FILE#	ONLINE	ERROR	CHANGE#	TIME
14	ONLINE			0
15	ONLINE	FILE NOT FOUND		0
21	OFFLINE	OFFLINE NORMAL		0

Note: The view is not useful if the control file currently in use is a restored backup or a new control file created since the media failure occurred. A restored or re-created control file does not contain the information Oracle needs to fill VSRECOVER_FILE accurately.

Query V\$DATAFILE and V\$TABLESPACE to obtain filenames and tablespace names for datafiles requiring recovery. For example, enter:

```
SQL> SELECT d.name, t.name
2 FROM v$datafile d, v$tablespace t
3 WHERE t.ts# = d.ts#
4 AND d.file# in (14,15,21); # use values obtained from V$RECOVER_FILE query
```

NAME	TABLESPACE_NAME
/oracle/dbs/tbs_14.f	TBS_1
/oracle/dbs/tbs_15.f	TBS_2
/oracle/dbs/tbs_21.f	TBS_3

Besides determining which files to recover, you must also know which files you should *not* recover. The following have special implications for media recovery:

- [Unrecoverable Tables and Indexes](#)
- [Read-Only Tablespaces](#)

Unrecoverable Tables and Indexes

You can create tables and indexes using the CREATE TABLE AS SELECT statement. You can also specify that Oracle create them as *unrecoverable*. When you create a table or index as unrecoverable, Oracle does not generate redo log records for the operation. Thus, you cannot recover objects created unrecoverable, even if you are running in ARCHIVELOG mode.

Note: If you cannot afford to lose tables or indexes created unrecoverable, take a backup after the unrecoverable table or index is created.

Be aware that when you perform media recovery, and some tables or indexes are created as recoverable while others are unrecoverable, the unrecoverable objects will be marked logically corrupt by the RECOVER operation. Any attempt to access the unrecoverable objects returns an ORA-01578 error message. Drop the unrecoverable objects and re-create them if needed.

Because it is possible to create a table unrecoverable and then create a recoverable index on that table, the index is not marked as logically corrupt after you perform media recovery. The table was unrecoverable (and thus marked as corrupt after recovery), however, so the index points to corrupt blocks. The index must be dropped, and the table and index must be re-created if necessary.

See Also: For information about the impact of unrecoverable operations on a standby database, see ["Determining Whether a Backup is Required After UNRECOVERABLE Operations"](#) on page 16-28.

Read-Only Tablespaces

Media recovery with the USING BACKUP CONTROLFILE option checks for read-only files. You cannot recover a read-only file. To avoid this error, take datafiles from read-only tablespaces offline before doing recovery with a backup control file.

Use the correct version of the control file for the recovery. If the tablespace will be read-only when the recovery is complete, then the control file must be from a time when the tablespace was read-only. Similarly, if the tablespace will be read-write at the end of recovery, it should be read-write in the control file.

If the appropriate control file is unavailable, execute a CREATE CONTROLFILE statement as described in ["Losing All Copies of the Current Control File"](#) on page 15-13. If you need to re-create a control file for a database with read-only tablespaces, issue the following to obtain the procedure that you need to follow:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

The procedure is similar to the procedure for offline normal tablespaces, except that you need to bring the tablespace online after the database is open.

See Also: To learn about taking trace backups of the control file, see ["Backing Up the Control File to a Trace File"](#) on page 13-12.

Restoring Files

If you determine that media recovery is necessary, restore the files necessary to perform it. Learn how to execute the following tasks:

- [Restoring Backup Datafiles](#)
- [Re-Creating Datafiles when Backups Are Unavailable](#)
- [Restoring Necessary Archived Redo Log Files](#)

Restoring Backup Datafiles

If a media failure permanently damages one or more datafiles of a database, you must restore backups of the damaged datafiles before you can recover the damaged files. If you cannot restore a damaged datafile to its original location (for example, you must replace a disk, so you restore the files to an alternate disk), then you must indicate the new locations of these files to the control file of the associated database.

To restore backup datafiles to their default location:

1. Determine which datafiles to recover using the techniques described in "[Determining Which Files to Recover](#)" on page 14-3.
2. Copy backups of the damaged datafiles to their default location using O/S commands. For example, to restore `tbs_14.f` on UNIX you might issue:

```
% cp /disk2/backup/tbs_14.bak /disk1/oracle/dbs/tbs_14.f
```

Re-Creating Datafiles when Backups Are Unavailable

If a datafile is damaged and no backup of the file is available, you can still recover the datafile if:

- All archived log files written since the creation of the original datafile are available.
- The control file contains the name of the damaged file (that is, the control file is current, or is a backup taken after the damaged datafile was added to the database).

To re-create a datafile for recovery:

1. Create a new, empty datafile to replace a damaged datafile that has no corresponding backup. For example, assume that the datafile `disk1:users1` has been damaged, and no backup is available. The following statement re-creates the original datafile (same size) on `disk2`:

```
ALTER DATABASE CREATE DATAFILE 'disk1:users1' AS 'disk2:users1';
```

This statement creates an empty file that matches the lost file. Oracle looks at information in the control file and the data dictionary to obtain size information. The old datafile is renamed as the new datafile.

2. Perform media recovery on the empty datafile. For example, enter:

```
RECOVER DATAFILE 'disk2:users1'
```

3. All archived redo logs written since the original datafile was created must be mounted and reapplied to the new, empty version of the lost datafile during recovery.

Note: You cannot re-create any of the datafiles for the SYSTEM tablespace by using the CREATE DATAFILE clause of the ALTER DATABASE statement, since the necessary redo data is not available.

Restoring Necessary Archived Redo Log Files

All archived redo log files required for the pending media recovery eventually need to be on disk so that they are readily available to Oracle.

To restore necessary archived redo logs:

1. To determine which archived redo log files you need, query `V$LOG_HISTORY` and `V$RECOVERY_LOG`. You will need all redo information from the time the datafile was added to the database if no backup of the datafile is available.

View	Description
<code>V\$LOG_HISTORY</code>	Lists all of the archived logs, including their probable names, given the current archived log file naming scheme as set by the initialization parameter <code>LOG_ARCHIVE_FORMAT</code> .

View	Description
V\$RECOVERY_LOG	Lists only the archived redo logs that Oracle needs to perform recovery. It also includes the probable names of the files, using LOG_ARCHIVE_FORMAT.

2. If space is available, restore the required archived redo log files to the location specified by LOG_ARCHIVE_DEST_1 or LOG_ARCHIVE_DEST. Oracle locates the correct log automatically when required during media recovery.

For example, enter:

```
% cp /disk2/arc_backup/*.arc /disk1/oracle/dbs/arc_dest
```

3. If sufficient space is not available at the location indicated by the destination initialization parameter, restore some or all of the required archived redo log files to an alternate location. Specify the location before or during media recovery using the LOGSOURCE parameter of the SET statement in SQL*Plus or the RECOVER ... FROM parameter of the ALTER DATABASE statement in SQL. For example, enter:

```
SET LOGSOURCE /disk2/temp # set location using SET command
ALTER DATABASE RECOVER FROM '/disk2/temp' DATABASE; # set in RECOVER statement
```

4. After an archived log is applied, and after making sure that a copy of each archived log group still exists in offline storage, delete the restored copy of the archived redo log file to free disk space. For example, after making the log directory your working directory, enter:

```
% rm *.arc
```

See Also: For more information about the data dictionary views, see the *Oracle8i Reference*.

Understanding Basic Media Recovery Procedures

Before beginning recovery, familiarize yourself with the following topics:

- [Using Media Recovery Statements](#)
- [Applying Archived Redo Logs](#)
- [Recovering a Database in NOARCHIVELOG Mode](#)
- [Recovering a Database in ARCHIVELOG Mode](#)
- [Performing Media Recovery in Parallel](#)

Using Media Recovery Statements

Oracle uses these basic media recovery SQL*Plus statements, which differ only in the way the system determines the set of files to be recovered:

- RECOVER DATABASE
- RECOVER TABLESPACE
- RECOVER DATAFILE

Each statement uses the same criteria to determine whether files are recoverable. Oracle prevents two recovery sessions from recovering the same file and prevents media recovery of a file that is in use.

You can also use the SQL statement ALTER DATABASE RECOVER, although Oracle strongly recommends you use the SQL*Plus RECOVER statement instead so that Oracle will prompt you for the names of the archived redo logs.

See Also: For more information about SQL*Plus RECOVER statements, see *SQL*Plus User's Guide and Reference*. For more information about the ALTER DATABASE RECOVER statement, see *Oracle8i SQL Reference*.

RECOVER DATABASE Statement

RECOVER DATABASE performs media recovery on all online datafiles that require redo to be applied. For example, issue the following at the SQL prompt to recover the whole database:

```
RECOVER DATABASE
```

If you shut down all instances cleanly, and did not restore any backups, issuing RECOVER DATABASE returns an error indicating that no recovery is required. It also fails if any instances have the database open, since they have the datafile locks. To perform media recovery on an entire database, the database must be mounted EXCLUSIVE and closed.

RECOVER TABLESPACE Statement

RECOVER TABLESPACE performs media recovery on all datafiles in the tablespaces listed. For example, enter the following at the SQL prompt to recover tablespace TBS_1:

```
RECOVER TABLESPACE tbs_1
```

The tablespaces must be offline to perform the recovery. Oracle indicates an error if none of the files require recovery.

RECOVER DATAFILE Statement

RECOVER DATAFILE lists the datafiles to be recovered. For example, enter the following at the SQL prompt to recover datafile `/oracle/dbs/tbs_22.f`:

```
RECOVER DATAFILE '/oracle/dbs/tbs_22.f'
```

The database can be open or closed, provided that you can acquire the media recovery locks. If the database is open in any instance, then datafile recovery can only recover offline files.

See Also: For more information about media recovery statements, see the *Oracle8i SQL Reference*.

Applying Archived Redo Logs

During complete or incomplete media recovery, Oracle applies redo log files to the datafiles during the roll forward phase of media recovery. Because rollback data is recorded in the redo log, rolling forward regenerates the corresponding rollback segments. Rolling forward proceeds through as many redo log files as necessary to bring the database forward in time.

As a log file is needed, Oracle suggests the name of the file. For example, if you are using SQL*Plus, it returns the following lines and prompts:

```
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread#
ORA-00289: Suggestion : logfile
ORA-00280: Change ##### for thread # is in sequence #
Specify log: [<RET> for suggested | AUTO | FROM logsource | CANCEL ]
```

Similar messages are returned when you use an ALTER DATABASE ... RECOVER statement. However, no prompt is displayed.

Suggested Archived Redo Log Filenames

Oracle suggests archived redo log filenames by concatenating the current values of the initialization parameters LOG_ARCHIVE_DEST_1 or LOG_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT and using information from the control file. For example, the following are possible settings for archived logs:

```
LOG_ARCHIVE_DEST_1 = /oracle/arc_dest/arc
LOG_ARCHIVE_FORMAT = r_%t_%s.arc
```



```
SQL> SELECT name FROM v$archived_log;
```

```
NAME
```

```
-----  
/oracle/arc_dest/arcr_1_467.arc  
/oracle/arc_dest/arcr_1_468.arc  
/oracle/arc_dest/arcr_1_469.arc  
/oracle/arc_dest/arcr_1_470.arc
```

Thus, if all the required archived log files are mounted at the LOG_ARCHIVE_DEST_1 or LOG_ARCHIVE_DEST destination, and the value for LOG_ARCHIVE_FORMAT is never altered, Oracle can suggest and apply log files to complete media recovery automatically.

To restore archived redo logs to a non-default location:

1. Change the value for this parameter to a new location. For example, enter:

```
LOG_ARCHIVE_DEST_1 = /oracle/new_location
```

2. Move the log files to the new location. For example, enter:

```
% cp /oracle/arc_dest/* /oracle/new_location
```

3. Start a new instance and mount the database:

```
STARTUP MOUNT
```

4. Initiate beginning media recovery as usual. For example, enter:

```
RECOVER DATABASE
```

In some cases, you may want to override the current setting for the destination parameter as a source for redo log files. For example, assume that a database is open and an offline tablespace must be recovered, but not enough space is available to mount the necessary redo log files at the location specified by the destination parameter.

To recover using logs in a non-default location:

1. Mount the archived redo logs to an alternate location. For example, enter:

```
% cp /disk1/oracle/arc_dest/* /disk2/temp
```

2. Specify the alternate location to Oracle for the recovery operation. Use the LOGSOURCE parameter of the SET statement or the RECOVER ... FROM parameter of the ALTER DATABASE statement. For example, enter:

```
SET LOGSOURCE "/disk2/temp"
```

3. Recover the offline tablespace:

```
RECOVER TABLESPACE offline_tbsp
```

Note: Overriding the redo log source does not affect the archive redo log destination for filled online groups being archived.

Consider overriding the current setting for the destination parameter when not enough space is available to mount all the required log files at any one location. In this case, you can set the log file source to an operating system variable (such as a logical or an environment variable) that acts as a search path to several locations.

See Also: Such functionality is operating system-dependent. See your operating system-specific Oracle documentation for more information.

Applying Logs Automatically Using the SQL*Plus RECOVER Statement

When using SQL*Plus, use the following command to automate the application of the default filenames of archived redo logs needed during recovery:

```
SET AUTORECOVERY ON
```

No interaction is required when you issue the RECOVER statement, provided that the necessary files are in the correct locations with the correct names.

The filenames used when you use SET AUTORECOVERY ON are derived from the values of the initialization parameters LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_1 in conjunction with LOG_ARCHIVE_FORMAT. If you execute SET AUTORECOVERY OFF, which is the default option, then you must enter the filenames manually, or accept the suggested default filename.

To automate the application of archived redo logs:

1. Restore a backup of the offline datafiles. This example restores a consistent backup of the whole database:

```
% cp /oracle/work/BACKUP/tbs* /oracle/dbs
```

2. Make sure the database is mounted. For example, if the database is shut down, enter:

```
SQL> STARTUP MOUNT
```

3. Turn on autorecovery:

```
SQL> SET AUTORECOVERY ON
Autorecovery ON
```

4. Recover the desired datafiles. This example recovers the whole database:

```
SQL> RECOVER DATABASE
```

5. Oracle automatically suggests and applies the necessary archived logs:

```
ORA-00279: change 53577 generated at 01/26/99 19:20:58 needed for thread 1
ORA-00289: suggestion : /oracle/work/arc_dest/arcr_1_802.arc
ORA-00280: change 53577 for thread 1 is in sequence #802
Log applied.
ORA-00279: change 53584 generated at 01/26/99 19:24:05 needed for thread 1
ORA-00289: suggestion : /oracle/work/arc_dest/arcr_1_803.arc
ORA-00280: change 53584 for thread 1 is in sequence #803
ORA-00278: log file '/oracle/work/arc_dest/arcr_1_802.arc' no longer needed for this
recovery
Log applied.
ORA-00279: change 53585 generated at 01/26/99 19:24:14 needed for thread 1
ORA-00289: suggestion : /oracle/work/arc_dest/arcr_1_804.arc
ORA-00280: change 53585 for thread 1 is in sequence #804
ORA-00278: log file '/oracle/work/arc_dest/arcr_1_803.arc' no longer needed for this
recovery
Log applied.
Media recovery complete.
```

If you use an OPS configuration, and you are performing incomplete recovery or using a backup control file, then Oracle can only compute the name of the first archived redo log file from the *first* thread. You may have to apply the first log file from the other threads. Once the first log file in a given thread has been supplied, Oracle can suggest the names of the subsequent logfiles in those threads.

See Also: For examples of log file application, see your operating system-specific Oracle documentation.

Applying Logs Individually Using ALTER DATABASE RECOVER

When you perform media recovery using SQL statements, Oracle does not display a prompt for log files after media recovery is started. Instead, you must provide the correct log file using an ALTER DATABASE RECOVER LOGFILE statement. For example, if a message suggests `log1.arc`, apply the suggestion using the following statement:

```
ALTER DATABASE RECOVER LOGFILE 'log1.arc';
```

As a result, recovering a tablespace requires several statements, as indicated in the following example (DBA input is boldfaced; variable information is italicized.):

```
SQL> ALTER DATABASE RECOVER TABLESPACE users;
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile1
ORA-00280: Change ##### for thread # is in sequence #
SQL> ALTER DATABASE RECOVER LOGFILE 'logfile1';
ORA-00279: Change ##### generated at DD/MM/YY HH:MM:SS needed for thread # <D%0>
ORA-00289: Suggestion : logfile2
ORA-00280: Change ##### for thread # is in sequence #
SQL> ALTER DATABASE RECOVER LOGFILE 'logfile2';
. . .
Repeat until all logs are applied.)
Statement processed.
SQL> ALTER TABLESPACE users ONLINE;
Statement processed.
```

Applying Logs Automatically Using ALTER DATABASE RECOVER

In this example, assume that the backup files have been restored, and that the user has administrator privileges. As in the method you used with SQL*Plus, automatic application of the redo logs can be started with the following statements, before and during recovery, respectively:

```
ALTER DATABASE RECOVER AUTOMATIC ...;
ALTER DATABASE RECOVER AUTOMATIC LOGFILE suggested_log_filename;
```

An example of the first statement follows:

```
SQL> ALTER DATABASE RECOVER AUTOMATIC TABLESPACE users;
Statement processed.
SQL> ALTER TABLESPACE users ONLINE;
Statement processed.
```

In this example, it is assumed that the backup files have been restored, and that the user has administrator privileges.

An example of the ALTER DATABASE RECOVER AUTOMATIC LOGFILE statement follows:

```
SQL> ALTER DATABASE RECOVER TABLESPACE users;
ORA-00279: Change #### generated at DD/MM/YY HH:MM:SS needed for thread #
ORA-00289: Suggestion : logfile1
ORA-00280: Change #### for thread # is in sequence #
SQL> ALTER DATABASE RECOVER AUTOMATIC LOGFILE 'logfile1';
Statement processed.
SQL> ALTER TABLESPACE users ONLINE;
Statement processed.
```

In this example, assume that the backup files have been restored, and that the user has administrator privileges.

Note: After issuing the ALTER DATABASE RECOVER statement, you can view all files that have been considered for recovery in the V\$RECOVERY_FILE_STATUS view. You can access status information for each file in the V\$RECOVERY_STATUS view. These views are not accessible after you terminate the recovery session.

See Also: For information about the content of all recovery-related views, see the *Oracle8i Reference*.

Successful Application of Redo Logs

If you are using SQL*Plus's recovery options (not SQL statements), each time Oracle finishes applying a redo log file, the following message is returned:

```
Log applied.
```

Oracle then prompts for the next log in the sequence or, if the most recently applied log is the last required log, terminates recovery.

Unsuccessful Application of Redo Logs

If the suggested file is incorrect or you provide an incorrect filename, Oracle returns an error message. For example, you may see something similar to the following:

```
ORA-00308: cannot open archived log '/oracle/work/arc_dest/arcr_1_811.arc'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

Recovery cannot continue until the required redo log file is applied. If Oracle returns an error message after supplying a redo log filename, the following scenarios are possible:

Error	Possible Cause	Solution
ORA-27037: unable to obtain file status	Entered wrong filename. Log is missing.	Re-enter correct filename. Restore backup archived redo log.
ORA-27047: unable to read the header block of file	The log may have been partially written or become corrupted.	If you can locate an uncorrupted or complete copy of the log, apply that copy; you do not need to restart recovery. If no copy of the log exists and you know the time of the last valid redo entry, perform incomplete recovery; in this case, restart recovery from the beginning, including restoring backups.

Interrupting the Application of Redo Logs

If you start a media recovery operation and must then interrupt it, e.g., because a recovery operation must end for the night and resume the next morning, interrupt recovery at any time by taking either of the following actions:

- Enter the word CANCEL when prompted for a redo log file.
- Use your operating system's interrupt signal if you must abort when recovering an individual datafile, or when automated recovery is in progress.

After recovery is canceled, it must be completed before opening a database for normal operation. To resume recovery, restart it. Recovery resumes where it left off when it was canceled.

There are several reasons why, after starting recovery, you may want to restart. For example, if you want to restart with a different backup or want to use the same backup but need to change the end time to an earlier point in time than you initially specified, then the entire operation must recommence by restoring a backup. Failure to do so may result in "file inconsistent" error messages when attempting to open the database.

Recovering a Database in NOARCHIVELOG Mode

If a database is in NOARCHIVELOG mode and a media failure damages some or all of the datafiles, the only option for recovering the database is usually to restore the most recent whole database backup. If you are using Export to supplement regular backups, then you can instead restore the database by importing an exported backup of the database.

The disadvantage of NOARCHIVELOG mode is that to recover your database from the time of the most recent full backup up to the time of the media failure, you have to re-enter manually all of the changes executed in that interval. If your database was in ARCHIVELOG mode, however, the redo log covering this interval would have been available as archived log files or online log files. Using archived redo logs would have enabled you to use complete or incomplete recovery to reconstruct your database, thereby minimizing the amount of lost work.

If you have a database damaged by media failure and operating in NOARCHIVELOG mode, and you want to restore from your most recent consistent whole database backup (your only option at this point), follow the steps below.

To restore the most recent whole database backup to the default location:

1. If the database is open, abort the instance:

```
SHUTDOWN ABORT
```

2. Correct the media problem so that the backup database files can be restored to their original locations.
3. Restore the most recent whole database backup using O/S commands. Restore all of the datafiles and control files of the whole database backup, not just the damaged files. This example restores a whole database backup:

```
% cp /oracle/work/BACKUP/tbs* /oracle/dbs # restores datafiles
% cp /oracle/work/BACKUP/cf.f /oracle/dbs # restores control file
```

4. Mimic incomplete database recovery by issuing the following statement:

```
ALTER DATABASE RECOVER DATABASE UNTIL CANCEL;
```

5. Open the database and reset the current redo log sequence to 1:

```
ALTER DATABASE OPEN RESETLOGS;
```

A RESETLOGS operation invalidates all redo in the online logs. Restoring from a whole database backup and then resetting the log discards all changes to the database made from the time the backup was taken to the time of the failure.

To restore the most recent whole database backup to a new location:

1. If the database is open, shut it down:

```
SHUTDOWN NORMAL
```

2. If the hardware problem has not been corrected and some or all of the database files must be restored to alternative locations, restore the most recent whole database backup to a new location. Restore all of the datafiles and control files of the whole database backup, not just the damaged files. For example, enter:

```
% cp /disk2/BACKUP/tbs* /disk3/oracle/dbs
% cp /disk2/BACKUP/cf.f /disk3/oracle/dbs
```

3. If necessary, edit the restored parameter file to indicate the new location of the control files.

```
CONTROL_FILES = '/disk3/oracle/dbs/cf.f'
```

4. Start an instance using the restored and edited parameter file and mount, but do not open, the database. For example, this statements mounts the database using the initialization file `initPROD1.ora`:

```
STARTUP MOUNT pfile=initPROD1.ora
```

5. If the restored datafile filenames will be different, rename the restored datafiles in the control file. For example, you might enter:

```
ALTER DATABASE RENAME FILE '/disk1/oracle/dbs/tbs1.f' TO '/disk3/oracle/dbs/tbs1.f';
```

6. If applicable, rename the online redo log files. For example, enter:

```
ALTER DATABASE RENAME FILE '/disk1/oracle/dbs/log1.f' TO '/disk3/oracle/dbs/log1.f';
```

7. Mimic incomplete database recovery by issuing the following statement:

```
ALTER DATABASE RECOVER DATABASE UNTIL CANCEL;
```

8. Mimic incomplete database recovery by issuing the following statement:

```
ALTER DATABASE RECOVER CANCEL;
```

9. Open the database and reset the current redo log sequence to 1:

```
ALTER DATABASE OPEN RESETLOGS;
```

A RESETLOGS operation invalidates all redo in the online logs. Restoring from a whole database backup and then resetting the log discards all changes to the database made from the time the backup was taken to the time of the failure.

See Also: For more information about renaming and relocating datafiles, see the *Oracle8i Administrator's Guide*.

Recovering a Database in ARCHIVELOG Mode

To begin media recovery operations when your database is running in ARCHIVELOG mode, use one of the following options:

- The ALTER DATABASE RECOVER statement
- The SQL*Plus RECOVER statement (recommended)

To start any type of media recovery, you must adhere to the following restrictions:

- You must have administrator privileges.
- All recovery sessions must be compatible.
- One session cannot start complete media recovery while another performs incomplete media recovery.
- You cannot start media recovery if you are connected to the database via a multi-threaded server process.

See Also: For more information about SQL*Plus, see the *SQL*Plus User's Guide and Reference*.

Performing Media Recovery in Parallel

Use *parallel block recovery* to tune the roll forward phase of media recovery. In parallel block recovery, Oracle uses a "division of labor" approach to allocate different processes to different data blocks while rolling forward, thereby making the procedure more efficient. For example, if the redo log contains a substantial number of entries, slave 1 takes responsibility for one part of the log file, slave 2 takes responsibility for another part, slave 3 takes responsibility for a third part, etc. Crash, instance, and media recovery of many datafiles on different disk drives are good candidates for parallel block recovery.

Use the RECOVERY_PARALLELISM initialization parameter to specify the number of concurrent recovery processes for any instance or media recovery operation. Because crash recovery occurs at instance startup, this parameter is useful for specifying the number of processes to use for crash recovery. The value of this parameter is also the default number of processes used for media recovery if you do not specify the PARALLEL clause of the RECOVER command. The value of this parameter must be greater than 1 and cannot exceed the value of the PARALLEL_

`MAX_SERVERS` parameter. Parallel block recovery requires a minimum of eight recovery processes to improve upon serial recovery.

Note that recovery is usually I/O bound on reads to data blocks. Consequently, parallelism at the block level may only help recovery performance if it increases total I/Os, e.g., by bypassing O/S restrictions on asynchronous I/Os. Systems that have efficient asynchronous I/O typical see little improvement from using parallel block recovery.

See Also: For more information on parallel recovery, see *Oracle8i Tuning*. For more information about the `RECOVERY_PARALLELISM` parameter, see the *Oracle8i Reference*.

Performing Complete Media Recovery

When you perform complete recovery, you can either recover the whole database at once or recover individual tablespaces or datafiles. Because you do not have to open the database with the `RESETLOGS` option after complete recovery as you do after incomplete recovery, you have the option of recovering some datafiles at one time and the remaining datafiles later.

This section describes the steps necessary to complete media recovery operations, and includes the following topics:

- [Performing Closed Database Recovery](#)
- [Performing Open Database Recovery](#)

See Also: Familiarize yourself with the fundamental recovery concepts and strategies in [Chapter 3, "Developing a Backup and Recovery Strategy"](#).

Performing Closed Database Recovery

This section describes steps to perform closed database recovery of either all damaged datafiles in one operation, or individual recovery of each damaged datafile in separate operations.

This section describes how to perform cancel-based media recovery in these stages:

1. Shut down the database and correct the media damage if possible.
2. Restore the necessary files.
3. Recover the database.

To prepare for closed database recovery:

1. If the database is open, shut it down using the ABORT option:

```
SHUTDOWN ABORT
```

2. If you are recovering from a media error, correct it if possible.
3. If the hardware problem that caused the media failure was temporary, and the data was undamaged (for example, a disk or controller power failure), simply start the database and resume normal operations:

```
STARTUP
```

To restore the necessary files:

1. Determine which datafiles to recover using the techniques described in "[Determining Which Files to Recover](#)" on page 14-3.
2. If files are permanently damaged, identify the most recent backups for the damaged files. Restore *only* the datafiles damaged by the media failure: do not restore any undamaged datafiles or any online redo log files.

For example, if `/oracle/dbs/tbs_10.f` is the damaged file, you may consult your records and determine that `/oracle/backup/tbs_10.backup` is the most recent backup. If you do not have a backup of a specific datafile, you may be able to create an empty replacement file that can be recovered.

3. Use an O/S utility to restore the files to their default location or to a new location. For example, a UNIX user restoring `/oracle/dbs/tbs_10.f` to its default location might enter:

```
% cp /oracle/backup/tbs_10.backup /oracle/dbs/tbs_10.f
```

Follow these guidelines when determining where to restore datafile backups:

If	Then
The hardware problem is repaired and you can restore the datafiles to their default locations	Restore the datafiles to their default locations and begin media recovery.
The hardware problem persists and you cannot restore datafiles to their original locations	Restore the datafiles to an alternative storage device. Indicate the new location of these files to the control file. Use the operation described in "Renaming and Relocating Datafiles" in the <i>Oracle8i Administrator's Guide</i> , as necessary.

To recover the restored datafiles:

1. Connect to Oracle with administrator privileges, then start a new instance and mount, but do not open, the database. For example, enter:

```
STARTUP MOUNT
```

2. Obtain the datafile names of all datafiles by checking the list of datafiles that normally accompanies the current control file or querying the V\$DATAFILE view. For example, enter:

```
SELECT name FROM v$datafile;
```

3. Ensure that all datafiles of the database are online. For example, to guarantee that a datafile named `/oracle/dbs/tbs_10.f` is online, enter the following:

```
ALTER DATABASE DATAFILE '/oracle/dbs/tbs_10.f' ONLINE;
```

If a specified datafile is already online, Oracle ignores the statement. If you prefer, create a script to bring all datafiles online at once as in the following:

```
SQL> SPOOL onlineall.sql
SQL> SELECT 'ALTER DATABASE DATAFILE '''||name||''' ONLINE;' FROM v$datafile;
SQL> SPOOL OFF
SQL> @onlineall
```

4. Issue the command to recover the database, tablespace, or datafile. For example, enter:

```
RECOVER DATABASE      # recovers whole database
RECOVER TABLESPACE users  # recovers specific tablespace
RECOVER DATAFILE '/oracle/dbs/tbs_10'; # recovers specific datafile
```

Follow these guidelines when deciding which statement to execute:

To	Then
Recover all damaged files in one step	Execute RECOVER DATABASE (recommended) or ALTER DATABASE RECOVER DATABASE
Recover an individual tablespace	Execute RECOVER TABLESPACE (recommended) or ALTER DATABASE RECOVER TABLESPACE
Recover an individual damaged datafile	Execute RECOVER DATAFILE (recommended) or ALTER DATABASE RECOVER DATAFILE
Parallelize recovery of the whole database or an individual datafile	See " Choosing Parallel Recovery Options " on page 14-8.

5. If you choose not to automate the application of archived redo logs, accept or reject each required redo log file that Oracle prompts you for. If you automated recovery, Oracle applies the necessary logs automatically. Oracle continues until all required archived and online redo log files have been applied to the restored datafiles.
6. Oracle notifies you when media recovery is complete. If no archived redo log files are required for complete media recovery, Oracle applies all necessary online redo log files and terminates recovery.
7. Open the database:

```
ALTER DATABASE OPEN;
```

See Also: For more information about applying redo log files, see "[Performing Complete Media Recovery](#)" on page 14-20.

Performing Open Database Recovery

It is possible for a media failure to occur while the database remains open, leaving the undamaged datafiles online and available for use. Oracle automatically takes the damaged datafiles offline.

This procedure cannot be used to perform complete media recovery on the datafiles of the SYSTEM tablespace. If the media failure damages any datafiles of the SYSTEM tablespace, Oracle automatically shuts down the database.

This section describes how to perform cancel-based media recovery in these stages:

1. Prepare the database for recovery by making sure it is open and taking the affected tablespaces offline.
2. Restore the necessary files.
3. Recover the database.

See Also: To proceed with complete media recovery of SYSTEM tablespaces datafiles, follow the procedure in "[Performing Closed Database Recovery](#)" on page 14-20.

To prepare for open database recovery when the database is shut down:

1. Start a new instance, and mount and open the database. For example, enter:

```
STARTUP
```

2. After the database is open, take all tablespaces that contain damaged datafiles offline. For example, if tablespace TBS_1 contains damaged datafiles, enter:

```
ALTER TABLESPACE tbs_1 OFFLINE TEMPORARY;
```

3. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, proceed with database recovery by restoring damaged files to an alternative storage device.

To prepare for recovery in an open database:

1. If the database is open when you discover that recovery is required, take all tablespaces containing damaged datafiles offline. For example, if tablespace TBS_1 contains damaged datafiles, enter:

```
ALTER TABLESPACE tbs_1 OFFLINE TEMPORARY;
```

If possible, take the damaged tablespaces offline with temporary priority to minimize the amount of recovery.

2. Correct the hardware problem that caused the media failure. If the hardware problem cannot be repaired quickly, proceed with database recovery by restoring damaged files to an alternative storage device.

To restore datafiles in an open database:

1. If files are permanently damaged, restore the most recent backup files of *only* the datafiles damaged by the media failure. Do not restore undamaged datafiles, online redo log files, or control files. If the hardware problem has been repaired and the datafiles can be restored to their original locations, do so. If the hardware problem persists, restore the datafiles to an alternative storage device of the database server.

Note: If you do not have a backup of a specific datafile, you can use ALTER DATABASE CREATE DATAFILE to create an empty replacement file, which can be recovered.

2. If you restored one or more damaged datafiles to alternative locations, indicate the new locations of these files to the control file of the associated database by using the procedure in "Renaming and Relocating Datafiles" in the *Oracle8i Administrator's Guide*, as necessary.

To recover offline tablespaces in an open database:

1. Connect to the database with administrator privileges. For example, connect as SYS to database PROD1:

```
% sqlplus sys/sys_pwd@prod1
```

2. Start offline tablespace recovery of all damaged datafiles in one or more offline tablespaces using one step:

```
RECOVER TABLESPACE tbs_1 # begins recovery on datafiles in tbs_1
```

Note: For maximum performance, use parallel recovery to recover the datafiles. See "[Performing Media Recovery in Parallel](#)" on page 14-19.

3. Oracle begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored datafiles. Unless the applying of files is automated using SET AUTORECOVERY ON, Oracle prompts for each required redo log file.

Oracle continues until all required archived redo log files have been applied to the restored datafiles. The online redo log files are then automatically applied to the restored datafiles to complete media recovery.

If no archived redo log files are required for complete media recovery, Oracle does not prompt for any. Instead, all necessary online redo log files are applied, and media recovery is complete.

4. When the damaged tablespaces are recovered up to the moment that media failure occurred, bring the offline tablespaces online. For example, to bring tablespace TBS_1 online, issue:

```
ALTER TABLESPACE tbs_1 ONLINE;
```

See Also: For more information about redo log application, see "[Performing Complete Media Recovery](#)" on page 14-20. For more information about creating datafiles, see the *Oracle8i Administrator's Guide*.

Performing Incomplete Media Recovery

This section describes the steps necessary to complete the different types of incomplete media recovery operations, and includes the following topics:

- [Performing Cancel-Based Recovery](#)
- [Performing Time-Based Recovery](#)
- [Performing Change-Based Recovery](#)

Note that if your database is affected by seasonal time changes (for example, daylight savings time), you may experience a problem if a time appears twice in the redo log and you want to recover to the second, or later time. To deal with time changes, perform cancel-based or change-based recovery to the point in time where the clock is set back, then continue with the time-based recovery to the exact time.

Performing Cancel-Based Recovery

This section describes how to perform cancel-based media recovery in these stages:

1. Prepare for recovery by backing up the database and correct any media failures.
2. Restore backup control files (if necessary) and backup datafiles.
3. Perform media recovery on the restored backup using the RECOVER DATABASE statement, terminating it with a CANCEL.

To prepare for cancel-based recovery:

1. If the database is still open and incomplete media recovery is necessary, abort the instance:

```
SHUTDOWN ABORT
```
2. Make a whole backup of the database—all datafiles, a control file, and the parameter files of the database—as a precautionary measure in case an error occurs during the recovery procedure.
3. If a media failure occurred, correct the hardware problem that caused the failure.

To restore the files necessary for cancel-based recovery:

1. If the current control files do not match the physical structure of the database at the intended time of recovery, e.g., if a datafile was added after the point in time to which you intend to recover, then restore a backup of the control file.

The restored control file should reflect the database's physical file structure, i.e., contain the names of datafiles and online redo log files, at the point at which incomplete media recovery is intended to finish. Review the list of files that correspond to the current control file as well as each control file backup to determine the correct control file to use.

If necessary, replace all current control files of the database with the correct control file backup. Alternatively, create a new control file.

Note: If a database control file cannot function or be replaced with a control file backup, take it out of the CONTROL_FILES parameter list in the parameter file associated with the database.

2. Restore backups taken as part of a full or partial backup of *all* the datafiles of the database. You must have taken all backup files used to replace existing datafiles before the intended time of recovery. For example, if you intend to recover to redo log sequence number 38, then restore all datafiles with backups completed before redo log sequence number 38.
3. If you do not have a backup of a specific datafile, create an empty replacement file that can be recovered. If you added a datafile after the intended time of recovery, you do not need to restore a backup for this file since it will no longer be used for the database after recovery is complete.
4. If you solved the hardware problem that caused a media failure and can restore all datafiles to their original locations, then restore the files. If a hardware problem persists, restore damaged datafiles to an alternative storage device.

Note: Files in read-only tablespaces should be taken offline if you are using a control file backup. Otherwise, recovery will try to update the headers of the read-only files.

To perform cancel-based recovery:

1. Start SQL*Plus and connect to Oracle with administrator privileges. For example, enter:

```
% sqlplus sys/change_on_install@prod1
```

2. Start a new instance and mount the database:

```
STARTUP MOUNT
```

3. If you restored one or more damaged datafiles to alternative locations, indicate the new locations of these files to the control file of the associated database.
4. Begin cancel-based recovery:

```
RECOVER DATABASE UNTIL CANCEL
```

If you are using a backup control file with this incomplete recovery, specify the **USING BACKUP CONTROLFILE** option in the **RECOVER** command.

```
RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE
```

Note: If you do not specify the **UNTIL CANCEL** clause on the **RECOVER** statement, you will not be able to open the database until a complete recovery is done.

5. Oracle applies the necessary redo log files to reconstruct the restored datafiles. Unless the application of files is automated, Oracle supplies the name it expects to find from **LOG_ARCHIVE_DEST_1** or **LOG_ARCHIVE_DEST** and requests you to stop or proceed with applying the log file. Note that if the control file is a backup file, you must supply names of online logs.

Note: If you use an OPS configuration, and you are performing incomplete recovery or using a backup control file, then Oracle can only compute the name of the first archived redo log file from the *first* thread. The first redo log file from the other threads must be supplied by the user. Once the first log file in a given thread has been supplied, Oracle can suggest the names of the subsequent log files in those threads.

6. Continue applying redo log files until the most recent, undamaged redo log file has been applied to the restored datafiles.

7. Cancel recovery after Oracle has applied the redo log file just prior to the damaged file:

```
CANCEL
```

Oracle returns a message indicating whether recovery is successful. Note that if you cancel recovery before it is complete and then try to open the database, you will get an ORA-1113 error if more recovery is necessary for the file.

Performing Time-Based Recovery

This section describes how to perform the time-based media recovery procedure in these stages:

1. Back up the database as a precaution and correct any media failures.
2. Restore backup control files (if necessary) and backup datafiles.
3. Perform media recovery on the restored backup using the RECOVER DATABASE statement with the UNTIL TIME option.

Note: If you are performing time-based, incomplete recovery using a backup control file and have read-only tablespaces, contact Oracle Support before attempting this procedure.

To prepare for time-based recovery:

Follow the same preparation procedure described in the section "[Performing Cancel-Based Recovery](#)" on page 14-29.

To restore the files necessary for time-based recovery and bring them online:

1. If the current control files do not match the physical structure of the database at the intended time of recovery, restore a backup control file that reflects the database's physical file structure at the point at which incomplete media recovery should finish. To determine which control file backup to use:
 - Review the list of files that corresponds to the current control file and each control file backup to determine the correct control file to use.
 - If necessary, replace all current control files of the database with the correct control file backup.
 - Alternatively, create a new control file to replace the missing one.

Note: If a database control file cannot function or be replaced with a control file backup, take it out of the CONTROL_FILES parameter list in the parameter file associated with the database.

2. Restore backups of *all* the datafiles of the database. All backups used to replace existing datafiles must have been taken before the intended time of recovery. For example, if you intend to recover to January 2 at 2 p.m., then restore all datafiles with backups completed before this time. Follow these guidelines:

If	Then
You do not have a backup of a datafile	Create an empty replacement file, which can be recovered.
A datafile was added after the intended time of recovery	Do not restore a backup of this file, since it will no longer be used for the database after recovery completes.
The hardware problem causing the failure has been solved and all datafiles can be restored to their default locations	Restore the files and skip Step 5 of this procedure.
A hardware problem persists	Restore damaged datafiles to an alternative storage device.

Note: Files in read-only tablespaces should be offline if you are using a control file backup. Otherwise, the recovery will try to update the headers of the read-only files.

3. Start SQL*Plus and connect to Oracle with administrator privileges. For example, enter:

```
% sqlplus sys/change_on_install@prod1
```

4. Start a new instance and mount the database:

```
STARTUP MOUNT
```

5. If one or more damaged datafiles were restored to alternative locations in Step 2, indicate the new locations of these files to the control file of the associated database. For example, enter:

```
ALTER DATABASE RENAME FILE '/oracle/dbs/df2.f' TO '/oracle/newloc/df2.f';
```

6. Obtain the names of all datafiles requiring recovery by:
 - Checking the list of datafiles that normally accompanies the control file being used.
 - Querying the V\$DATAFILE view.
7. Make sure that all datafiles of the database are online. All datafiles of the database must be online unless an offline tablespace was taken offline normally. For example, to guarantee that a datafile named `user1` (a fully specified filename) is online, enter the following statement:

```
ALTER DATABASE DATAFILE 'users1' ONLINE;
```

If a backup of the control file is being used with this incomplete recovery (that is, a control file backup or re-created control file was restored), indicate this in the dialog box or command used to start recovery. If a specified datafile is already online, Oracle ignores the statement.

To perform time-based recovery:

1. Issue the `RECOVER DATABASE UNTIL TIME` statement to begin time-based recovery. The time is always specified using the following format, delimited by single quotation marks: `'YYYY-MM-DD:HH24:MI:SS'`. The following statement recovers the database up to a specified time using a control file backup:

```
RECOVER DATABASE UNTIL TIME '1992-12-31:12:47:30' USING BACKUP CONTROLFILE
```
2. Apply the necessary redo log files to reconstruct the restored datafiles. Unless the application of files is automated, Oracle supplies the name it expects to find from `LOG_ARCHIVE_DEST_1` or `LOG_ARCHIVE_DEST` and requests you to stop or proceed with applying the log file. If the control file is a backup, you must supply names of online logs.
3. Apply redo log files until the last required redo log file has been applied to the restored datafiles. Oracle automatically terminates the recovery when it reaches the correct time, and returns a message indicating whether recovery is successful.

Performing Change-Based Recovery

This section describes how to perform recovery to a specified SCN in these stages:

1. Back up the database as a precaution and correct any media failures.
2. Restore backup control files (if necessary) and backup datafiles.
3. Perform media recovery on the restored backup using the RECOVER DATABASE statement with the UNTIL TIME option.

To prepare for change-based recovery:

Follow the same preparation procedure described in the section ["Performing Cancel-Based Recovery"](#) on page 14-29.

To restore files necessary for change-based recovery:

Follow the same restore procedure described in the section ["Performing Time-Based Recovery"](#) on page 14-29.

To perform change-based recovery:

1. Begin change-based recovery, specifying the SCN for recovery termination. The SCN is specified as a decimal number without quotation marks. For example, to recover until SCN 100 issue:

```
RECOVER DATABASE UNTIL CHANGE 100;
```

2. Oracle begins the roll forward phase of media recovery by applying the necessary redo log files (archived and online) to reconstruct the restored datafiles. Unless the application of files is automated, Oracle supplies the name it expects to find from LOG_ARCHIVE_DEST_1 or LOG_ARCHIVE_DEST and requests you to stop or proceed with applying the log file. If the control file is a backup file, you must supply names of online logs. Oracle continues to apply redo log files.
3. Continue applying redo log files until the last required redo log file has been applied to the restored datafiles. Oracle automatically terminates the recovery when it reaches the correct time, and returns a message indicating whether recovery is successful.

Opening the Database after Media Recovery

To preserve the log sequence number when opening a database after recovery, execute the following statement:

```
ALTER DATABASE OPEN NORESETLOGS;
```

To reset the log sequence number when opening a database after recovery, execute the following statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

The RESETLOGS option is *required* after incomplete media recovery. Resetting the redo log:

- Discards any redo information that was not applied during recovery, ensuring that it will never be applied.
- Re-initializes the control file information about online redo logs and redo threads.
- Clears the contents of the online redo logs.
- Creates the online redo log files if they do not currently exist.
- Resets the log sequence number to 1.

WARNING: Resetting the redo log discards all changes to the database made since the first discarded redo information. Updates entered after that time must be re-entered manually.

Deciding Whether to Specify RESETLOGS or NORESETLOGS

Use the following rules when deciding whether to specify RESETLOGS or NORESETLOGS:

- *Always* specify the RESETLOGS option after incomplete media recovery. For example, you must have specified a previous time or SCN, not one in the future.
- Specify the NORESETLOGS option after performing complete media recovery (unless you used a backup control file). This rule applies when you intentionally performed complete recovery and when you performed incomplete recovery but actually recovered all changes in the redo logs anyway.

- Always specify the `RESETLOGS` option if you used a backup of the control file in recovery, regardless of whether you performed complete or incomplete recovery.
- Do not specify the `RESETLOGS` option if you are using the archived logs of the database for a standby database. If you must reset the log, then you will have to re-create your standby database.

If you reset the log sequence number, Oracle returns different messages depending on whether recovery was complete or incomplete. If the recovery was complete, the following message appears in the `alert.log` file:

```
RESETLOGS after complete recovery through change scn
```

If the recovery was incomplete, this message is reported in the `ALERT` file:

```
RESETLOGS after incomplete recovery UNTIL CHANGE scn
```

If you attempt to reset the log when you should not, or if you neglect to reset the log when you should, Oracle returns an error and does not open the database. Correct the error and try again.

Guidelines for Opening in `RESETLOGS` Mode

Perform the following actions after opening the database in `RESETLOGS` mode:

- [Making a Whole Database Backup](#)
- [Checking the Alert Log](#)

Making a Whole Database Backup Immediately shut down the database normally and make a full database backup. Otherwise, you will not be able to recover changes made after you reset the logs. Until you take a full backup, the only way to recover will be to repeat the procedures you just finished, up to resetting the logs. (You do not need to back up the database if you did not reset the log sequence.)

Checking the Alert Log After opening the database using the `RESETLOGS` option, check the `alert.log` to see whether Oracle detected inconsistencies between the data dictionary and the control file (for example, a datafile that the data dictionary includes but does not list in the new control file).

If a datafile exists in the data dictionary but not in the new control file, Oracle creates a placeholder entry in the control file under `MISSINGnnnn` (where `nnnn` is the file number in decimal). `MISSINGnnnn` is flagged in the control file as being offline and requiring media recovery.

The actual datafile corresponding to `MISSINGnnnn` can be made accessible by renaming `MISSINGnnnn` so that it points to the datafile only when the datafile was read-only or offline normal. If, on the other hand, `MISSINGnnnn` corresponds to a datafile that was not read-only or offline normal, then the rename operation cannot be used to make the datafile accessible, because the datafile requires media recovery that is precluded by the results of `RESETLOGS`. In this case, you must drop the tablespace containing the datafile.

In contrast, if a datafile indicated in the control file is not present in the data dictionary, Oracle removes references to it from the new control file. In both cases, Oracle includes an explanatory message in the `alert.log` file to let you know what was found.

See Also: For more information about applying redo logs, see "[Performing Complete Media Recovery](#)" on page 14-20.

Recovering a Pre-RESETLOGS Backup

In pre-Oracle8i releases, DBAs typically backed up online logs when performing cold consistent backups to avoid opening the database with the `RESETLOGS` option (if they were planning to restore immediately).

A classic example of this technique was disk maintenance, which required the database to be backed up, deleted, the disks reconfigured, and the database restored. DBAs realized that by not restarting in `RESETLOGS` mode, they would not have to perform a whole database backup immediately after the database was restored. This backup was required since it was impossible to perform recovery using a backup taken before using `RESETLOGS`—especially if any errors occurred after resetting the logs.

In Oracle version 8 and higher, there is only one situation in which you can use a pre-`RESETLOGS` backup to roll forward—if you have a *consistent* backup of the database, taken *immediately before* you open the database with the `RESETLOGS` option, *and* a control file that is valid *after* you open the database with `RESETLOGS`. It is then unnecessary to back up or restore online redo logs.

The following scenario illustrates a situation when you can use a pre-`RESETLOGS` backup. Suppose you wish to perform hardware striping reconfiguration, which requires the database files to be backed up and deleted, the hardware reconfigured, and the database restored.

On Friday night you perform the following actions:

1. Cleanly shut down the database:

```
SHUTDOWN IMMEDIATE
```

2. Perform a whole database backup (control files and datafiles). For example, enter

```
% cp /oracle/dbs/* /oracle/backup
```

Note: At this point you must not reopen the database.

3. Perform O/S maintenance.

4. Restore the datafiles and control files. For example, enter:

```
% cp /oracle/backup/* /oracle/dbs
```

5. Mount the database:

```
STARTUP MOUNT
```

6. Initiate cancel-based recovery:

```
RECOVER DATABASE UNTIL CANCEL
```

7. Open the database with the RESETLOGS option:

```
ALTER DATABASE OPEN RESETLOGS;
```

On Saturday morning the scheduled batch jobs run, generating archived redo logs. If a hardware error occurs on Saturday night that requires you to restore the whole database, you can restore the backup taken immediately before opening the database with the RESETLOGS option, and roll forward using the logs produced on Saturday.

On Saturday night you do the following:

1. Abort the instance (if it still exists):

```
SHUTDOWN ABORT
```

2. Restore all damaged files from the backup made on Friday night:

```
% cp /oracle/backup/* /oracle/dbs
```

Note: If you have the current control file, *do not* restore it; otherwise you must restore a control file that was valid after opening the database with RESETLOGS.

3. Begin complete recovery, applying all the archived logs produced on Saturday. Use SET AUTORECOVERY ON to automate the log application.

```
SET AUTORECOVERY ON  
RECOVER DATABASE
```

4. Open the database:

```
STARTUP
```

In this scenario, if you had opened the database after the Friday night backup and before opening the database with RESETLOGS, or, did not have a control file from after opening the database, you *would not* be able to use the Friday night backup to roll forward. You must have a backup after opening the database with the RESETLOGS option in order to be able to recover.

Operating System Recovery Scenarios

This chapter describes how to recover from common media failures, and includes the following topics:

- [Understanding the Types of Media Failures](#)
- [Recovering After the Loss of Datafiles](#)
- [Recovering Through an ADD DATAFILE Operation](#)
- [Recovering Transported Tablespaces](#)
- [Recovering After the Loss of Online Redo Log Files](#)
- [Recovering After the Loss of Archived Redo Log Files](#)
- [Recovering After the Loss of Control Files](#)
- [Recovering from User Errors](#)
- [Performing Media Recovery in a Distributed Environment](#)

Understanding the Types of Media Failures

Media failures fall into two general categories: *permanent* and *temporary*. Use different recovery strategies depending on the type of failure.

Permanent media failures are serious hardware problems that cause permanent loss of data on the disk. Lost data cannot be recovered except by repairing or replacing the failed storage device and restoring backups of the files stored on the damaged storage device.

Temporary media failures are hardware problems that make data temporarily inaccessible, but do not corrupt the data. Following are two examples of temporary media failures:

- A disk controller fails. Once the disk controller is replaced, Oracle can access the data on the disk.
- Power to a storage device is cut off. Once the power is returned, the storage device and all associated data is accessible again.

Recovering After the Loss of Datafiles

If a media failure affects datafiles, the appropriate recovery procedure depends on:

- The archiving mode of the database: ARCHIVELOG or NOARCHIVELOG.
- The type of media failure.
- The files affected by the media failure.

The following sections explain the appropriate recovery strategies for the database mode:

- [Losing Datafiles in NOARCHIVELOG Mode](#)
- [Losing Datafiles in ARCHIVELOG Mode](#)

Losing Datafiles in NOARCHIVELOG Mode

If either a permanent or temporary media failure affects *any* datafiles of a database operating in NOARCHIVELOG mode, Oracle automatically shuts down the database. Depending on the type of media failure, you can use one of two recovery methods:

- If the media failure is temporary, correct the hardware problem and restart the database. Usually, instance recovery is possible, and all committed transactions can be recovered using the online redo log.

- If the media failure is permanent, follow the procedure ["Recovering a Database in NOARCHIVELOG Mode"](#) on page 14-17.

Losing Datafiles in ARCHIVELOG Mode

If either a permanent or temporary media failure affects the datafiles of a database operating in ARCHIVELOG mode, the following scenarios can occur:

Damaged Datafiles	Database Status	Solution
Datafiles in the SYSTEM tablespace or datafiles with active rollback segments.	Oracle shuts down.	If the hardware problem is temporary, fix it and restart the database. Usually, instance recovery recovers lost transactions. If the hardware problem is permanent, follow the procedure in "Performing Closed Database Recovery" on page 14-20.
Non-SYSTEM datafiles or datafiles that do not contain active rollback segments.	Oracle takes affected datafiles offline, but the database stays open.	If the unaffected portions of the database must remain available, do not shut down the database. Take tablespaces containing problem datafiles offline using the temporary option, then follow the procedure in "Performing Closed Database Recovery" on page 14-20.

Recovering Through an ADD DATAFILE Operation

If database recovery rolls forward through an ADD DATAFILE operation, Oracle will stop the recovery when applying the ADD DATAFILE redo data and let you confirm the location of the file.

For example, suppose you create a new tablespace containing two datafiles: `/db/db2.f` and `/db/db3.f`. If you later perform media recovery through the CREATE TABLESPACE operation, Oracle may signal the following error when applying the CREATE TABLESPACE redo data:

```
ORA-00283: recovery session canceled due to errors
ORA-01244: unnamed datafile(s) added to controlfile by media recovery
ORA-01110: data file 3: '/db/db2.f'
ORA-01110: data file 2: '/db/db3.f'
```

To recover through an ADD DATAFILE operation:

1. View the files added by selecting from V\$DATAFILE:

```
SELECT file#, name FROM v$datafile;
```

FILE#	NAME
1	/db/db1.f
2	/db/UNNAMED00002
3	/db/UNNAMED00003

2. If multiple unnamed files exist, determine which unnamed file corresponds to which datafile using one of these methods:
 - Open the `alert.log`, which contains messages about the original file location for each unnamed file.
 - Derive the original file location of each unnamed file from the error message and V\$DATAFILE: each unnamed file corresponds to the file in the error message with the same file number.
3. Issue the ALTER DATABASE RENAME FILE command to rename the datafiles. For example, enter:

```
ALTER DATABASE RENAME FILE '/db/UNNAMED00002' TO '/db/db3.f';  
ALTER DATABASE RENAME FILE '/db/UNNAMED00003' TO '/db/db2.f';
```

4. Continue recovery by issuing the previous recovery command. For example, enter:

```
RECOVER DATABASE
```

Recovering Transported Tablespaces

The *transportable tablespace* feature of Oracle allows a user to transport a set of tablespaces from one database to another. Transporting or "plugging" a tablespace into a database is like creating a tablespace with pre-loaded data. Using this feature is often an advantage because:

- It is faster than import/export or load/unload, since it involves only copying datafiles and integrating metadata.
- You can use it to move index data, which allows you to avoid having to rebuild indexes.

Like normal tablespaces, plugged-in tablespaces are recoverable. While you can recover normal tablespaces without a backup, you must have a version of the plugged-in datafiles in order to recover a plugged-in tablespace.

To recover a plugged-in tablespace, restore a backup of the plugged-in datafiles and issue normal recovery commands. The backup can be the initial version of the plugged-in datafiles or any backup taken after the tablespace is plugged in. Just as when recovering through a CREATE TABLESPACE operation, Oracle may signal ORA-01244 when recovering through a tablespace plug-in operation. In this case, rename the unnamed files to the correct locations using the procedure in "[Recovering Through an ADD DATAFILE Operation](#)" on page 15-3.

See Also: For detailed information about using the transportable tablespace feature, see *Oracle8i Administrator's Guide*.

Recovering After the Loss of Online Redo Log Files

If a media failure has affected the online redo logs of a database, the appropriate recovery procedure depends on:

- The configuration of the online redo log: mirrored or non-mirrored.
- The type of media failure: temporary or permanent.
- The types of online redo log files affected by the media failure: current, active, unarchived, or inactive.

The following sections describe the appropriate recovery strategies these situations:

- [Recovering After Losing a Member of a Multiplexed Online Redo Log Group](#)
- [Recovering After the Loss of All Members of an Online Redo Log Group](#)

Recovering After Losing a Member of a Multiplexed Online Redo Log Group

If the online redo log of a database is multiplexed, and at least one member of each online redo log group is not affected by the media failure, Oracle allows the database to continue functioning as normal. Oracle writes error messages to the LGWR trace file and the `alert.log` of the database.

Solve the problem by taking one of the following actions:

- If the hardware problem is temporary, correct it. LGWR then accesses the previously unavailable online redo log files as if the problem never existed.

- If the hardware problem is permanent, drop the damaged member and add a new member using the procedure below.

Note: The newly added member provides no redundancy until the log group is reused.

To replace a damaged member of a redo log group:

1. Locate the filename of the damaged member in V\$LOGFILE. The status will be INVALID if the file is inaccessible:

```
SELECT group#, status, member FROM v$logfile;
```

GROUP#	STATUS	MEMBER
0001		/oracle/dbs/log1a.f
0001		/oracle/dbs/log1b.f
0002		/oracle/dbs/log2a.f
0002	INVALID	/oracle/dbs/log2b.f
0003		/oracle/dbs/log3a.f
0003		/oracle/dbs/log3b.f

2. Drop the damaged member. For example, to drop member log2b.f from group 2, issue:

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log2b.f';
```

3. Add a new member to the group. For example, to add log2c.f to group 2, issue:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.f' TO GROUP 2;
```

If the file you want to add already exists, it must be the same size as the other group members, and you must specify REUSE:

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.f' REUSE TO GROUP 2;
```

Recovering After the Loss of All Members of an Online Redo Log Group

If a media failure damages all members of an online redo log group, different scenarios can occur, depending on the type of online redo log group affected by the failure and the archiving mode of the database.

If the damaged log group is inactive, then it is not needed for instance recovery; if it is active, it is needed for instance recovery. Your first task is to determine whether the damaged group is active or inactive.

To determine whether the damaged groups are active:

1. Locate the filename of the lost redo log in V\$LOGFILE and then look for the group number corresponding to it. For example, enter:

```
SELECT group#, status, member FROM v$logfile;
```

```
GROUP#      STATUS      MEMBER
-----      -
0001                /oracle/dbs/log1a.f
0001                /oracle/dbs/log1b.f
0002      INVALID  /oracle/dbs/log2a.f
0002      INVALID  /oracle/dbs/log2b.f
0003                /oracle/dbs/log3a.f
0003                /oracle/dbs/log3b.f
```

2. Determine which groups are active. For example, enter:

```
SELECT group#, members, status, archived FROM v$log;
```

```
GROUP# MEMBERS      STATUS      ARCHIVED
-----
0001    2      INACTIVE  YES
0002    2      ACTIVE    NO
0003    2      CURRENT   NO
```

3. If the affected group is inactive, follow the procedure in ["Losing an Inactive, Online Redo Log Group"](#) on page 15-7. If the affected group is active (as in the above example), follow the procedure in ["Losing an Active Online Redo Log Group"](#) on page 15-9.

Losing an Inactive, Online Redo Log Group

If all members of an online redo log group with INACTIVE status are damaged, the procedure depends on whether you can fix the media problem that damaged the inactive redo log group.

Type of Media Failure	Procedure
Temporary	Fix the problem. LGWR can reuse the redo log group when required.
Permanent	The damaged inactive online redo log group will eventually halt normal database operation. Clear, i.e., reinitialize, the damaged group manually using ALTER DATABASE CLEAR LOGFILE as described below.

You can clear an active redo log group when the database is open or closed. The procedure depends on whether the damaged group has been archived.

To clear an inactive, online redo log group that has been archived:

1. If the database is shut down, start a new instance and mount the database, but do not open it:

```
STARTUP MOUNT
```

2. Reinitialize the damaged log group. For example, to clear redo log group 2, issue:

```
ALTER DATABASE CLEAR LOGFILE GROUP 2;
```

To clear an inactive, online redo log group that has not been archived:

Clearing an unarchived log allows it to be reused without archiving it. This action will make backups unusable if they were started before the last change in the log, unless the file was taken offline prior to the first change in the log. Hence, if you need the cleared log file for recovery of a backup, you cannot recover that backup.

1. If the database is shut down, start a new instance and mount the database, but do not open it:

```
STARTUP MOUNT
```

2. Clear the log using the UNARCHIVED keyword. For example, to clear log group 2, issue:

```
ALTER DATABASE CLEAR LOGFILE UNARCHIVED GROUP 2;
```

If there is an offline datafile that requires the cleared unarchived log to bring it online, the keywords UNRECOVERABLE DATAFILE are required. The datafile and its entire tablespace will have to be dropped because the redo necessary to bring it online is being cleared, and there is no copy of it. For example, enter:

```
ALTER DATABASE CLEAR LOGFILE UNARCHIVED GROUP 2 UNRECOVERABLE DATAFILE;
```

3. Immediately back up the database using an O/S utility. Now you can use this backup for complete recovery without relying on the cleared log group. For example, enter:

```
% cp /disk1/oracle/dbs/*.f /disk2/backup
```

4. Back up the database's control file:

```
ALTER DATABASE BACKUP CONTROLFILE TO 'filename';
```

Failure of CLEAR LOGFILE Operation The ALTER DATABASE CLEAR LOGFILE command can fail with an I/O error due to media failure when it is not possible to:

- Relocate the redo log file onto alternative media by re-creating it under the currently configured redo log file name.
- Reuse the currently configured log file name to recreate the redo log file because the name itself is invalid or unusable (for example, due to media failure).

In these cases, the CLEAR LOGFILE command (before receiving the I/O error) would have successfully informed the control file that the log was being cleared and did not require archiving. The I/O error occurred at the step in which CLEAR LOGFILE attempts to create the new redo log file and write zeros to it.

Losing an Active Online Redo Log Group

If your database is still running and the lost active log is not the current log, issue the ALTER SYSTEM CHECKPOINT command. If successful, your active log is rendered inactive, and you can follow the procedure in "[Losing an Active Online Redo Log Group](#)" on page 15-9. If unsuccessful, or if your database has halted, perform one of these procedures, depending on the archiving mode.

To recover from loss of an active online redo log group in NOARCHIVELOG mode:

1. If the media failure is temporary, correct the problem so that Oracle can reuse the group when required.
2. Restore the database from a whole database backup using an O/S utility. For example, enter:

```
% cp /disk2/backup/*.f /disk1/oracle/dbs
```

3. Mount the database:

```
STARTUP MOUNT
```

4. Open the database using the RESETLOGS option:

```
ALTER DATABASE OPEN RESETLOGS;
```

5. Shut down the database normally:

```
SHUTDOWN IMMEDIATE
```

6. Make a whole database backup. For example, enter:

```
% cp /disk1/oracle/dbs/*.f /disk2/backup
```

To recover from loss of an active online redo log group in ARCHIVELOG mode:

1. If the media failure is temporary, correct the problem so that Oracle can reuse the group when required.
2. Perform incomplete media recovery. Use the procedure given in "[Performing Incomplete Media Recovery](#)" on page 14-26, recovering up through the log before the damaged log.
3. Ensure that the current name of the lost redo log can be used for a newly created file. If not, rename the members of the damaged online redo log group to a new location. For example, enter:

```
ALTER DATABASE RENAME FILE '/oracle/dbs/log_1.rdo' TO '/temp/log_1.rdo';  
ALTER DATABASE RENAME FILE '/oracle/dbs/log_2.rdo' TO '/temp/log_2.rdo';
```

4. Open the database using the RESETLOGS option:

```
ALTER DATABASE OPEN RESETLOGS;
```

Note: All updates executed from the endpoint of the incomplete recovery to the present must be re-executed.

Loss of Multiple Redo Log Groups

If you have lost multiple groups of the online redo log, use the recovery method for the most difficult log to recover. The order of difficulty, from most difficult to least, follows:

1. The current online redo log
2. The active online redo log
3. The unarchived redo log
4. The inactive online redo log

Recovering After the Loss of Archived Redo Log Files

If the database is operating in ARCHIVELOG mode, and the only copy of an archived redo log file is damaged, it does not affect the present operation of the database. The following situations can arise, however, depending on when the redo log was written and when you backed up the datafile.

If you backed up	Then
<i>All</i> datafiles <i>after</i> the filled online redo log group (which is now archived) was written.	The archived version of the filled online redo log group is not required for complete media recovery operation.
A given datafile <i>before</i> the filled online redo log group was written.	If the corresponding datafile is damaged by a permanent media failure, use the most recent backup of the damaged datafile and perform incomplete recovery up to the damaged log.

WARNING: If you know that an archived redo log group has been damaged, immediately back up all datafiles so that you will have a whole database backup that does not require the damaged archived redo log.

Recovering After the Loss of Control Files

If a media failure has affected the control files of a database (whether control files are multiplexed or not), the database continues to run until the first time that an Oracle background process needs to access the control files. At this point, the database and instance are automatically shut down.

If the media failure is temporary and the database has not yet shut down, avoid the automatic shutdown of the database by immediately correcting the media failure. If the database shuts down before you correct the temporary media failure, however, then you can restart the database after fixing the problem and restoring access to the control files.

The appropriate recovery procedure for media failures that permanently prevent access to control files of a database depends on whether you have multiplexed the control files. The following sections describe the appropriate procedures:

- [Losing a Member of a Multiplexed Control File](#)
- [Losing All Copies of the Current Control File](#)

Losing a Member of a Multiplexed Control File

Use the following procedures to recover a database if all the following conditions are met:

- A permanent media failure has damaged one or more control files of a database.
- At least one control file has *not* been damaged by the media failure.

Note: If all control files of a multiplexed control file configuration have been damaged, follow the procedure in "[Losing All Copies of the Current Control File](#)" on page 15-13.

To restore a control file to its default location:

1. If the instance is still running, abort it:

```
SHUTDOWN ABORT
```

2. Correct the hardware problem that caused the media failure. If you cannot repair the hardware problem quickly, you can proceed with database recovery by restoring damaged control files to an alternative storage device.
3. Use an intact multiplexed copy of the database's current control file to copy over the damaged control files. For example, to replace `bad_cf.f` with `good_cf.f`, you might enter:

```
% cp /oracle/good_cf.f /oracle/dbs/bad_cf.f
```

4. Start an instance and open the database:

```
STARTUP
```

To restore a control file to its non-default location:

1. If the instance is still running, abort it:

```
SHUTDOWN ABORT
```

2. If you cannot correct the hardware problem that caused the media failure, copy the intact control file to alternative locations. For example, to rename `good_cf.f` as `new_cf.f` you might issue:

```
% cp /oracle/dbs/good_cf.f /oracle/dbs/new_cf.f
```


3. Edit the parameter file of the database so that the `CONTROL_FILES` parameter reflects the current locations of all control files and excludes all control files that were not restored. For example, to add `new_cf.f` you might enter:

```
CONTROL_FILES = '/oracle/dbs/good_cf.f', '/oracle/dbs/new_cf.f'
```

4. Start a new instance and open the database:

```
STARTUP
```

Losing All Copies of the Current Control File

If all control files of a database have been lost or damaged by a permanent media failure, but all online redo logfiles remain intact, you can recover the database by creating a new control file.

To create a new control file:

1. Create the control file using the `CREATE CONTROLFILE` statement with the `NORESETLOGS` option. See [Table 15-1](#) for options.
2. Recover the database as normal:

```
RECOVER DATABASE
```

Depending on the existence and currency of a control file backup, you have the following options for generating the text of the `CREATE CONTROLFILE` command

Table 15–1 Options for Creating the Control File

If you	Then
Executed ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS after you made the last structural change to the database, and if you have saved the SQL command output	Use the CREATE CONTROLFILE statement from the output as-is.
Performed your most recent execution of ALTER DATABASE BACKUP CONTROLFILE TO TRACE before you made a structural change to the database	Edit the output of ALTER DATABASE BACKUP CONTROLFILE TO TRACE to reflect that change. For example, if you recently added a datafile to the database, add that datafile to the DATAFILE clause of the CREATE CONTROLFILE statement.
Have not backed up the control file using the TO TRACE option, but used the TO <i>filename</i> option of ALTER DATABASE BACKUP CONTROLFILE	Use the control file copy to obtain SQL command output. Copy the backup control file and execute STARTUP MOUNT before executing ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS. If your control file copy predated a recent structural change, edit the TO TRACE output to reflect that structural change.
Do not have a backup of the control file in either TO TRACE format or TO <i>filename</i> format	Generate the CREATE CONTROLFILE statement manually.

Note: If your character set is not the default US7ASCII, then you must specify the character set as an argument to the CREATE CONTROLFILE statement.

Recovering from User Errors

An accidental or erroneous operational or programmatic change to the database can cause loss or corruption of data. Recovery may require a return to a state prior to the error.

Note: If you have properly granted powerful privileges (such as DROP ANY TABLE) to only selected, appropriate users, user errors that require database recovery are minimized.

The following scenario describes how to recover a table that has been accidentally dropped.

1. You can keep the database that experienced the user error online and available for normal use. The database can remain open or be shut down. Back up all datafiles of the existing database in case an error is made during the remaining steps of this procedure.
2. Create a temporary copy of the database to a past point-in-time using time-based recovery. Be careful not to cause a conflict with the existing control file of the permanent database. Restore a single control file backup to an alternative location (Step 4) and edit the parameter file, as necessary, or create a new control file at the alternative location. Also, restore all datafiles to alternative locations (Step 5) so that you do not affect the permanent copy of the database.
3. Export the lost data using the Oracle utility Export from the temporary, restored version of the database. In this case, export the accidentally dropped table.

Note: System audit options are exported.

4. Import the exported data (Step 3) into the permanent copy of the database using the Oracle Import utility.
5. Delete the files of the temporary, reconstructed copy of the database to conserve space.

See Also: For more information about the Import and Export utilities, see *Oracle8i Utilities*.

Performing Media Recovery in a Distributed Environment

The manner in which you perform media recovery depends on whether your database participates in a distributed database system. The Oracle distributed database architecture is autonomous. Therefore, depending on the type of recovery operation selected for a single, damaged database, you may have to coordinate recovery operations globally among all databases in the distributed database system.

[Table 15–2](#) summarizes different types of recovery operations and whether coordination among nodes of a distributed database system is required.

Table 15–2 Recovery Operations in a Distributed Database Environment

If you are	Then
Restoring a whole backup for a database that was never accessed from a remote node	Use non-coordinated, autonomous database recovery.
Restoring a whole backup for a database that was accessed by a remote node	Shut down all databases and restore them using the same coordinated full backup.
Performing complete media recovery of one or more databases in a distributed database	Use non-coordinated, autonomous database recovery.
Performing incomplete media recovery of a database that was never accessed by a remote node	Use non-coordinated, autonomous database recovery.
Performing incomplete media recovery of a database that was accessed by a remote node	Use coordinated, incomplete media recovery to the same global point in time for all databases in the distributed database.

Coordinating Time-Based and Change-Based Distributed Database Recovery

In special circumstances, one node in a distributed database may require recovery to a past time. To preserve global data consistency, it is often necessary to recover all other nodes in the system to the same point in time. This operation is called *coordinated, time-based, distributed database recovery*. The following tasks should be performed with the standard procedures of time-based and change-based recovery described in this chapter.

1. Recover the database that requires the recovery operation using time-based recovery. For example, if a database needs to be recovered because of a user error (such as an accidental table drop), recover this database first using time-based recovery. Do not recover the other databases at this point.
2. After you have recovered the database and opened it using the RESETLOGS option, look in the `alert.log` of the database for the RESETLOGS message.

If the message is, "**RESETLOGS after complete recovery through change xxx,**" you have applied all the changes in the database and performed a complete recovery. Do not recover any of the other databases in the distributed system, or you will unnecessarily remove changes in them. Recovery is complete.

If the message is, "**RESETLOGS after incomplete recovery UNTIL CHANGE xxx,**" you have successfully performed an incomplete recovery. Record the change number from the message and proceed to the next step.

3. Recover all other databases in the distributed database system using change-based recovery, specifying the change number (SCN) from Step 2.

Recovering a Database with Snapshots

If a master database is independently recovered to a past time (that is, coordinated, time-based distributed database recovery is not performed), any dependent remote snapshot that was refreshed in the interval of lost time will be inconsistent with its master table. In this case, the administrator of the master database should instruct the remote administrators to perform a complete refresh of any inconsistent snapshot.

Managing a Standby Database

This chapter describes how to maintain a standby database. It includes the following topics:

- [Planning a Standby Database](#)
- [Creating a Standby Database](#)
- [Choosing the Standby Database Mode](#)
- [Maintaining a Standby Database in Recovery Mode](#)
- [Opening a Standby Database in Read-Only Mode](#)
- [Activating a Standby Database](#)
- [Altering the Physical Structure of the Primary Database](#)
- [Using a Standby Database in an OPS Configuration](#)

Planning a Standby Database

When developing your backup and recovery strategy, consider whether to maintain a *standby database*. You can use a standby database to maintain a duplicate copy of your production database.

In the event that all media are destroyed at your production site, the standby database can replace the destroyed or damaged database. For maximum disaster protection, place the datafiles, redo log files, and control files of your primary and standby databases on separate physical media in separate geographical areas.

You can also locate the production and standby databases in the same data center or even on the same machine. This configuration is useful if you use the standby in read-only mode for reporting purposes.

This section contains the following topics:

- [Standby Database Advantages](#)
- [Standby Database Requirements](#)

Standby Database Advantages

A standby database can be a powerful tool for both disaster prevention and reporting. You can:

- Transmit archived redo logs automatically from the production database to the standby database, thereby keeping the standby up-to-date.
- Maintain a standby database in a location that is geographically remote from your production database, or maintain several standby databases in several different geographically diverse locations.
- Make a standby database the new production database with minimal loss of time and data in the event that the production database is completely destroyed.
- Protect against erroneous batch jobs or application corruptions on the production database by activating the standby before archived logs containing corrupt data are propagated.

Standby Database Requirements

Note the following requirements for maintaining a standby database:

- The standby database operates only on Oracle release 7.3 or higher.
- The redo logs that you apply to the standby database must be either archived or non-current redo logs.
- You must use the same version, release, and patch of the operating systems on the production and standby hosts. The standby host can, however, use a different disk configuration.
- Use the same version, release, and patch of the Oracle RDBMS for the production and standby databases so that failover is not compromised.
- You cannot use the control file of the primary database on the standby database.
- If you place your production and standby databases on the same host, some operating systems will not allow you to mount two instances with the same database name on the same machine simultaneously.
- A standby database cannot be activated and then returned to standby recovery mode; an activated standby functions as a normal production database and must be re-created as a standby database.

Creating a Standby Database

You can create a standby database on the same host as your production database or on a remote host. If you create your standby on the same host, follow the procedure carefully so that you do not overwrite important files.

To create a standby database:

1. Create a standby `init.ora` file by copying the production `init.ora` file. Configure the standby initialization parameters using the considerations described in "[Configuring Initialization Parameters](#)" on page 16-25.
2. Start a SQL*Plus session on your primary database and issue a `SELECT` on `V$DATAFILE` to obtain a list of datafiles. For example, enter:

```
SELECT name FROM v$datafile;
NAME
-----
/oracle/dbs/tbs_01.f
/oracle/dbs/tbs_02.f
/oracle/dbs/tbs_03.f
```

```
/oracle/dbs/tbs_11.f  
/oracle/dbs/tbs_12.f  
/oracle/dbs/tbs_21.f  
/oracle/dbs/tbs_22.f  
7 rows selected.
```

3. Shut down your primary database cleanly:

```
SHUTDOWN IMMEDIATE
```

4. Make a consistent backup of the datafiles from your primary database using an O/S utility.
5. Open the database:

```
STARTUP
```

6. Connect to the production database and create the control file for your standby database. For example, enter the following (where *filename* is the fully specified pathname):

```
ALTER DATABASE CREATE STANDBY CONTROLFILE AS 'filename';
```

7. Archive the current online redo logs of the primary database:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

This operation ensures consistency among the datafiles, the control file, and the redo log files.

8. Transfer the standby database control file, archived log files, and backed up datafiles to the standby site using operating system commands or utilities. Use an appropriate method if transferring binary files (see "[Maintaining a Standby Database in Recovery Mode](#)" on page 16-6). For example, enter:

```
% cp /oracle/dbs/*.f /standby/oracle/dbs/*.f
```

9. If your standby database is on a remote host with the same filesystem as your primary database, you can use the same filenames for the standby database as the primary database. If not, use the filename conversion initialization parameters described in "[Converting Filenames for Datafiles and Archived Redo Logs](#)" on page 16-15.

See Also: For ALTER DATABASE and ALTER SYSTEM syntax, see *Oracle8i SQL Reference*.

Choosing the Standby Database Mode

Oracle allows you to perform one of three mutually exclusive operations on your standby database. You can mount the standby database and then:

- Maintain it in manual or managed recovery mode (see "[Maintaining a Standby Database in Recovery Mode](#)" on page 16-6).
- Open it read-only for queries (see "[Opening a Standby Database in Read-Only Mode](#)" on page 16-19). You can then place it back in recovery mode.
- Activate it, at which point it is no longer a standby database (see "[Activating a Standby Database](#)" on page 16-21). You can open it read-write or read-only.

To protect against disaster, keep your standby database in *recovery mode*, which means that you cannot query or open it for any purpose other than to initiate disaster recovery. You have the option of placing the database in either *manual recovery mode*, in which you must continually transfer and apply archived redo logs to the standby database, or *managed recovery mode*, in which the procedure is automated.

You can also open your standby database in *read-only mode*. This option allows you to query the database and even store data in temporary tablespaces without affecting the datafiles or redo logs. If you need to return to recovery mode, you can do so at any time.

Note: Opening a standby database in read-only mode requires you to set the COMPATIBLE parameter to 8.1 or higher.

Once you activate your standby database, it ceases to become a standby database and functions as a production database. You cannot return it to standby recovery mode unless you re-create it as another standby database.

WARNING: Activating a standby database resets the online logs of the standby database. Hence, after activation, the logs from your standby database and production database are incompatible.

Maintaining a Standby Database in Recovery Mode

This section describes how to maintain your standby database in recovery mode, which allows you to restore and recover the database in case of disaster. Depending on whether you want archived logs to be applied manually or automatically, you can choose between manual recovery mode and managed recovery mode. You can also control how Oracle manages obsolete online logfiles and filename conversion.

This section contains the following topics:

- [Placing the Standby in Manual Recovery Mode](#)
- [Placing the Standby Database in Managed Recovery Mode](#)
- [Transmitting Archived Redo Logs to a Standby Database](#)
- [Maintaining the Standby Database in Recovery Mode](#)

Placing the Standby in Manual Recovery Mode

Once you have started and mounted your database, you can place it in manual recovery mode. To keep your standby database current, you must manually apply archived redo logs from your target database to your standby database. For details about various media recovery options, e.g., recovering a database to a non-current date, see "[Performing Incomplete Media Recovery](#)" on page 14-26.

To place the standby database in manual recovery mode:

1. Configure the initialization parameters for the standby site. See "[Maintaining a Standby Database in Recovery Mode](#)" on page 16-6 and "[Configuring Initialization Parameters](#)" on page 16-25.
2. Use SQL*Plus to start the Oracle instance at the standby database. For example, enter:

```
STARTUP NOMOUNT
```
3. Mount the standby database:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```
4. Transfer the archived redo logs to the desired location on the standby host. Use an appropriate operating system utility for transferring binary data.
5. Place the standby database in recovery mode, optionally specifying the FROM *'location'* option. If you omit this parameter, Oracle assumes the archived redo log file group is in the location specified by the initialization parameter LOG_

ARCHIVE_DEST_ *n* (where *n* is an integer from 1 to 5) or LOG_ARCHIVE_DEST initialization parameter. For example, enter:

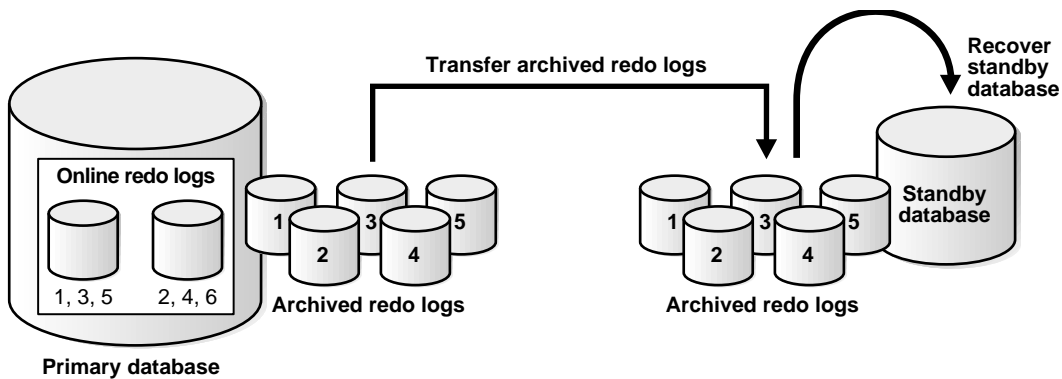
```
RECOVER STANDBY DATABASE # uses archiving location for logs specified in init.ora
RECOVER FROM '/logs' STANDBY DATABASE # specifies non-default location
```

Note: As Oracle generates archived redo logs, continually transfer and apply them to the standby database to keep the standby current.

Placing the Standby Database in Managed Recovery Mode

When you operate your production database in *managed recovery mode*, you can automate archiving to either a local or remote host. Oracle keeps the standby database synchronized with the primary database by waiting for archived logs from the primary and then automatically applying them to the standby. This feature eliminates the need for you to interactively provide the recovery process with the filenames of the archived logs.

Figure 16–1 Transmitting and Applying Archived Redo Logs to a Standby Database



When placing the database in managed recovery mode, use the TIMEOUT option of the RECOVER statement to specify an optional *timeout interval*. In this case, the managed recovery operation waits the specified number of minutes for Oracle to write the requested archived log entry to the standby control file's directory.

If Oracle times out because it cannot find the required next log entry in the standby control file, the system issues an appropriate message and exits managed recovery

mode. By default the managed recovery operation waits indefinitely for a requested archived redo log; it terminates only through user intervention, a shutdown, or crash.

Cancel the operation at any time by issuing `RECOVER MANAGED STANDBY DATABASE CANCEL` with or without the `IMMEDIATE` option. `RECOVER MANAGED STANDBY DATABASE CANCEL` waits for the managed recovery operation to finish with the current redo log file before terminating the recovery operation.

If you use the `CANCEL` statement with the `IMMEDIATE` option, Oracle stops the managed recovery operation either before reading another block from the redo log file or before opening the next redo log file—whichever comes first.

If Oracle terminates recovery before opening the next redo log, then `CANCEL IMMEDIATE` is equivalent to `CANCEL`. If Oracle terminates recovery while processing a log, then `CANCEL IMMEDIATE` leaves the database in an inconsistent state. Note that Oracle does not allow a database to be opened in an inconsistent state.

To place the standby database in managed recovery mode:

1. Use SQL*Plus to start the standby database without mounting it, specifying a parameter file if necessary:

```
STARTUP NOMOUNT pfile=initSTANDBY.ora
```

2. Mount the database:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```

3. Put the standby database in managed recovery mode:

```
RECOVER MANAGED STANDBY DATABASE
```

If you wish to use the optional time-out option, add `TIMEOUT integer` to the command syntax, as in the following:

```
RECOVER MANAGED STANDBY DATABASE TIMEOUT 60
```

Note: Do not use `FROM` with the `MANAGED` option.

Transmitting Archived Redo Logs to a Standby Database

For the background archiver processes to archive to a standby location, the following must be true:

- The `LOG_ARCHIVE_DEST_n` parameters (where *n* is an integer from 1 to 5) must be correctly defined using the `SERVICE` attribute. The `SERVICE` attribute must be used for both local and remote locations.
- The `tnsnames.ora` on the primary and the `listener.ora` on the standby have the correct corresponding entries.
- The standby instance must be started on the standby site in order for the *remote file server* (RFS) to be started and begin receiving network requests from the archiving primary processes. An RFS process is required when archiving to a standby destination.

Each `ARCn` process creates a corresponding RFS for each standby destination. For example, if three `ARCn` processes are archiving to two standby databases, then Oracle establishes six RFS connections.

The `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` parameters in the standby `init.ora` file determine the filenames for the archived redo logs at the standby site. These filenames are stored in the standby database control file; access them using the `V$ARCHIVED_LOG` dynamic performance view.

This section contains these topics:

- [Specifying Archive Destinations in the Primary Parameter File](#)
- [Specifying Mandatory and Optional Destinations](#)
- [Enabling Archive Destination States](#)
- [Re-Archiving to a Failed Destination](#)
- [Specifying Archived Log Filenames for the Standby Database](#)
- [Specifying Archive Destinations: Example](#)

See Also: To learn how to manage archived redo logs, see the chapter on archived redo logs in the *Oracle8i Administrator's Guide*. For more information about Oracle networking options, see the *Net8 Administrator's Guide*. For an overview of the `ARCn` archiver process, see *Oracle8i Concepts*.

Specifying Archive Destinations in the Primary Parameter File

Specify the number of locations for your primary database archived logs by setting the following initialization parameters:

Parameter	Host	Example
LOG_ARCHIVE_DEST_ <i>n</i> (where <i>n</i> is an integer from 1 to 5)	Remote or local	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc/' LOG_ARCHIVE_DEST_2 = 'SERVICE = standby1'
LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST	Local only	LOG_ARCHIVE_DEST = /oracle/arc LOG_ARCHIVE_DUPLEX_DEST = /bak

When you maintain a standby database, use the LOG_ARCHIVE_DEST_*n* parameter to specify from one to five different destinations for archival. Each numerically-suffixed parameter uniquely identifies an individual destination, e.g., LOG_ARCHIVE_DEST_1, LOG_ARCHIVE_DEST_2, etc.

Alternatively, you can use LOG_ARCHIVE_DEST in optional conjunction with LOG_ARCHIVE_DUPLEX_DEST to specify up to two locations. Note that you cannot use LOG_ARCHIVE_DEST in conjunction with LOG_ARCHIVE_DEST_*n*.

Specify the location using these keywords:

Keyword	Indicates	Example
LOCATION	A local filesystem location.	LOG_ARCHIVE_DEST_1= 'LOCATION=/arc/'
SERVICE	Remote archival via Net8 service name.	LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1'

If you use the LOCATION keyword, specify a valid pathname for your operating system. If you specify SERVICE, Oracle translates the net service name through the `tnsnames.ora` file to a connect descriptor. The descriptor contains the information necessary for connecting to the remote database. Note that the service name must have an associated database SID, so that Oracle correctly updates the log history of the control file for the standby database.

See Also: For a detailed account of LOG_ARCHIVE_DEST_*n* and the archiving process, see the chapter on archived redo logs in the *Oracle8i Administrator's Guide*. For information about STANDBY_ARCHIVE_DEST, SERVICE_NAME, or other

related initialization parameters, see the *Oracle8i Reference*. For information about the `tnsnames.ora` file or network configuration parameters, see the *Net8 Administrator's Guide*.

Specifying Mandatory and Optional Destinations

Using the `LOG_ARCHIVE_DEST_n` parameter, you can specify whether a destination has the attributes `OPTIONAL` (default) or `MANDATORY`. For example, you can set the parameter as follows:

```
LOG_ARCHIVE_DEST_3 = 'SERVICE=standby1 MANDATORY'
```

The `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` parameter uses all `MANDATORY` destinations plus some number of `OPTIONAL` non-standby destinations to determine whether LGWR can over-write the online redo log.

See Also: For a detailed account of the `OPTIONAL` and `MANDATORY` keywords, see the chapter on archived redo logs in the *Oracle8i Administrator's Guide*.

Enabling Archive Destination States

The `LOG_ARCHIVE_DEST_STATE_n` (where n is an integer from 1 to 5) parameter identifies the status of the specified destination. The destination parameters can have two values: `ENABLE` and `DEFER`. `ENABLE` indicates that Oracle can use the destination, whereas `DEFER` indicates that it should not.

For example, you can set the parameter as follows:

```
LOG_ARCHIVE_DEST_STATE_2 = ENABLE
```

See Also: For a detailed account of the archive destination states, see the chapter on archived redo logs in the *Oracle8i Administrator's Guide*. For a description of the `LOG_ARCHIVE_DEST_STATE_n` parameter, see the *Oracle8i Reference*.

Re-Archiving to a Failed Destination

Use the `REOPEN` attribute of the `LOG_ARCHIVE_DEST_n` parameter to determine whether and when `ARCn` attempts to re-archive to a failed destination following an error. `REOPEN` applies to all errors, not just `OPEN` errors.

`REOPEN=n` sets the minimum number of seconds before `ARCn` should try to reopen a failed destination. If you specify `REOPEN`, it has a default value 300 seconds. If you do not specify `REOPEN`, it has the value of 0, which is the same as turning off the option. If you do not specify the `REOPEN` keyword, `ARCn` will never reopen a destination following an error.

You cannot use REOPEN to specify a limit on the number of attempts to reconnect and transfer archived logs. The REOPEN attempt either succeeds or fails, in which case the REOPEN information is reset.

For example, you can set the parameter as follows to specify a reopen time of 60 seconds:

```
LOG_ARCHIVE_DEST_2 = 'SERVICE=standby2 OPTIONAL REOPEN=60'
```

Note: Archived logs that are not transmitted while communication is lost must be shipped manually.

See Also: For a detailed account of how to use the REOPEN option, see the chapter on archived redo logs in the *Oracle8i Administrator's Guide*.

Specifying Archived Log Filenames for the Standby Database

The standby RFS process uses the STANDBY_ARCHIVE_DEST parameter in the standby's `init.ora` file to determine the directory location for the archived redo logs. Oracle uses this value in conjunction with LOG_ARCHIVE_FORMAT to generate the archived log filenames on the standby host.

Parameter	Indicates	Example
STANDBY_ARCHIVE_DEST	Directory in which to place archived logs.	STANDBY_ARCHIVE_DEST= /arc_dest
LOG_ARCHIVE_FORMAT	Format for archived redo log filename.	LOG_ARCHIVE_FORMAT = "log%s.arc"

Oracle stores the fully qualified filenames in the standby control file. Managed recovery uses this information to perform the recovery operation. Access this information via V\$ARCHIVED_LOG:

```
SQL> SELECT name FROM v$archived_log;
NAME
```

```
-----
/arc_dest/log771.arc
/arc_dest/log772.arc
/arc_dest/log773.arc
/arc_dest/log774.arc
/arc_dest/log775.arc
```

Specifying Filenames with the RECOVER Command With the exception of RECOVER MANAGED STANDBY DATABASE, the RECOVER STANDBY DATABASE commands rely on one of the following to provide the location of the archived files:

- The LOG_ARCHIVE_DEST value
- A user-specified filename

If you run the database in managed recovery mode, then you must issue RECOVER STANDBY DATABASE if the following situations occur:

- An archived redo log is absent from the standby host, i.e., the RFS process has not recorded its name in the standby control file
- The managed recovery operation fails.

Issuing RECOVER STANDBY DATABASE in these circumstances requires you to use the LOG_ARCHIVE_DEST parameter to locate the necessary archived redo log. For a standby database in managed recovery mode, Oracle recommends setting STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_DEST to the same value. In this way, the various types of recovery operations can access the same set of archived redo logs.

Specifying Archive Destinations: Example

This example assumes the following:

- The primary database PROD1 is on host LOCAL.
- There are two standby databases, STANDBY1 and STANDBY2.
- STANDBY1 is on local host LOCAL, while STANDBY2 is on a remote host REMOTE2.

INIT.ORA Settings for PROD1 Following are sample settings for the LOG_ARCHIVE_DEST_1 and LOG_ARCHIVE_DEST_2 parameters in the primary database PROD1's `init.ora` file:

```
# This example specifies net service name "standby1", makes archiving mandatory, and
# enables the destination.

# A REOPEN value of 5 indicates that if the LOG_ARCHIVE_DEST_1 location
# encounters an error during archival of a redo log file, Oracle will remain inactive
# until the archival of a redo file is about to begin and 5 seconds has elapsed. At that
# time, Oracle re-attempts the archival to LOG_ARCHIVE_DEST_1.

# If Oracle encounters an error when archiving to a destination, that destination
# is inactive for the duration of the archival of the current redo log file.
```

```
# The destination may be reactivated (based on the REOPEN attribute) at the start
# of the archival of another redo log.
```

```
LOG_ARCHIVE_DEST_1 = 'SERVICE=standby1 MANDATORY REOPEN=5'
LOG_ARCHIVE_DEST_STATE_1 = ENABLE
```

```
# Specifies net service name "standby2", makes archiving optional, and specifies that
# Oracle should re-try archiving after 5 seconds should an error occur. The destination
# is enabled.
```

```
LOG_ARCHIVE_DEST_2 = 'SERVICE=standby2 OPTIONAL REOPEN=5'
LOG_ARCHIVE_DEST_STATE_2 = ENABLE
```

TNSNAMES.ORA Settings Following are settings in the `tnsnames.ora` file for the standby databases `STANDBY1` and `STANDBY2` in the above example:

```
# The standby1 standby database is on the same node as the primary.
```

```
standby1 = (DESCRIPTION=
            (ADDRESS=
              (PROTOCOL=ipc)
              (KEY=stby))
            (CONNECT_DATA=
              (SID=stby1)
              (SERVER=DEDICATED)))
```

```
# The standby2 standby database is on a different node from the primary.
```

```
standby2 = (DESCRIPTION=
            (ADDRESS=
              (PROTOCOL=tcp)
              (HOST=remote2)
              (PORT=1512)
            )
            (CONNECT_DATA=
              (SID=stby2)
              (GLOBAL_NAME=standby2)
              (SERVER=DEDICATED)))
```

LISTENER.ORA Settings Following are the settings in the `listener.ora` files for the standby databases `STANDBY1` and `STANDBY2`:

```
# The listener settings for standby1 on host local
```

```
LISTENER = (ADDRESS_LIST=
            (ADDRESS=
              (PROTOCOL=ipc)
              (KEY=stby1)))
```

```
SID_LIST_LISTENER = (SID_LIST=
                    (SID_DESC=(SID_NAME=stby1)(ORACLE_HOME=/oracle))
```

```
# The listener settings for standby2 on the remote host remote2
LISTENER = (ADDRESS_LIST=
  (ADDRESS=
    (PROTOCOL=tcp)
    (KEY=stby2)
    (HOST=remote2)
    (PORT=1512)))

SID_LIST_LISTENER = (SID_LIST=
  (SID_DESC=(SID_NAME=stby2)(ORACLE_HOME=/oracle))
```

INIT.ORA Settings for STANDBY1 and STANDBY2 Following are settings in the `init.ora` files for the standby databases STANDBY1 and STANDBY2 in the above example. These settings determine the filenames on the standby database for the archived redo logs:

```
# The init.ora values for the standby1 database, which is on the same host as the primary
STANDBY_ARCHIVE_DEST = /oracle/standby/arc
LOG_ARCHIVE_DEST = /oracle/standby/arc
LOG_ARCHIVE_FORMAT = log%s.arc

# The init.ora values for the standby2 database, which is on host remote2
STANDBY_ARCHIVE_DEST = /oracle/standby/arc
LOG_ARCHIVE_DEST = /oracle/standby/arc
LOG_ARCHIVE_FORMAT = log%s.arc
```

Maintaining the Standby Database in Recovery Mode

When running a standby database, be mindful of the various maintenance issues that can arise. If possible, research the solutions to possible problems before placing the standby in recovery mode.

This section contains the following topics:

- [Converting Filenames for Datafiles and Archived Redo Logs](#)
- [Determining the Most Recently Applied Archived Redo Log](#)
- [Clearing Online Redo Logs](#)
- [Making Backups of a Standby Database](#)

Converting Filenames for Datafiles and Archived Redo Logs

Set the following initialization parameters so that your standby database converts files from your primary database control file. If your primary and standby databases occupy the same node, these parameters allow you to distinguish the filenames for the standby and primary databases. Note that if you do not set the

LOCK_NAME_SPACE parameters differently for same-node systems using OPS, you will receive an ORA-1102 error.

Table 16–1 *Filename Conversion*

Parameter	Function
DB_FILE_NAME_CONVERT	Transforms primary database datafile filenames to standby datafile filenames, e.g., from tbs_* to standbytbs_*.
LOG_FILE_NAME_CONVERT	Transforms primary database redo log filenames to standby redo log filenames, e.g., from log_* to standbylog_*.
LOCK_NAME_SPACE	Specifies the name space that the distributed lock manager (DLM) uses to generate lock names. You may need to set this value in OPS configurations if there is a standby database with the same database name on the same cluster.

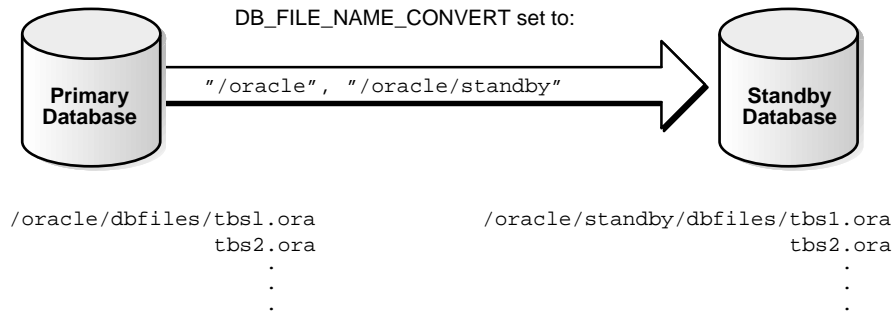
Use DB_FILE_NAME_CONVERT to convert the filename of a new datafile on the primary database to a filename on the standby database; use LOG_FILE_NAME_CONVERT to convert the filename of a new redo log on the primary database to a filename on the standby database. Adding a datafile or log to the primary database necessitates adding a corresponding file to the standby database.

When the standby database is updated, this parameter is used to convert the datafile name on the primary database to the a datafile name on the standby database. The file must exist and be writable on the standby database or the recovery process will halt with an error.

The DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT parameters must have two strings. The first string is a sequence of characters to be looked for in a primary database filename. If that sequence of characters is matched, it is replaced by the second string to construct the standby database filename.

Figure 16–2 shows how the filename conversion parameters work.

Figure 16–2 Setting Filename Conversion Parameters



If you execute the following statements, then the conversion parameters will not apply to the affected files:

- ALTER TABLESPACE RENAME DATAFILE
- ALTER DATABASE RENAME FILE
- ALTER DATABASE CREATE DATAFILE ... AS

See Also: To learn how to add datafiles to the standby database, see ["Adding Datafiles"](#) on page 16-23.

Determining the Most Recently Applied Archived Redo Log

Use either of these methods for determining when the most recent archived redo log was applied to the standby database:

- Query the V\$LOG_HISTORY view, which records the latest log sequence number that has been applied.
- View the trace data in alert.log. Oracle updates the trace data whenever it applies an archived redo log.

Note: V\$LOG is not updated during recovery.

Clearing Online Redo Logs

You can clear standby database online redo logs to optimize performance by issuing the ALTER DATABASE CLEAR LOGFILE GROUP *integer* command. If you prefer not to perform this operation during maintenance, the online logs will be cleared automatically during activation.

Making Backups of a Standby Database

If needed, you can back up your standby database—but not while the database is in manual or managed recovery mode. You must take the standby database out of managed recovery mode, take the backups, then resume managed recovery. You can make the backups when the database is shut down or when it is in read-only mode.

Depending on the size of your database, this procedure may be time-consuming, which means that the primary database may have to wait to archive its logs because the standby database is down. Consequently, the best solution is to follow one of the procedures below.

To back up tablespaces on a standby database when it is shut down:

1. Start a SQL*Plus session on the standby database and take the database out of managed recovery mode:

```
RECOVER MANAGED STANDBY DATABASE CANCEL
```

2. Shut down the database:

```
SHUTDOWN IMMEDIATE
```

3. Take cold backups of some tablespaces using O/S utilities. Minimize the time that the database is down. For example, to back up datafiles `tbs11.f`, `tbs12.f`, and `tbs13.f` in tablespace `TBS_1` on UNIX you might enter:

```
% cp /disk1/oracle/dbs/tbs11.f /disk2/backup/tbs11.bk
% cp /disk1/oracle/dbs/tbs12.f /disk2/backup/tbs12.bk
% cp /disk1/oracle/dbs/tbs12.f /disk2/backup/tbs13.bk
```

4. Use SQL*Plus to start the Oracle instance at the standby database without mounting it, specifying a parameter file if necessary:

```
STARTUP NOMOUNT pfile = initSTANDBY.ora
```

5. Mount the database:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```

6. Recover the database:

```
RECOVER MANAGED STANDBY DATABASE
```

If you wish to use the optional time-out option, add `TIMEOUT integer` to the command syntax as in the following:

```
RECOVER MANAGED STANDBY DATABASE TIMEOUT 60
```


7. Repeat the above steps until you have backed up each tablespace in the database.

To back up tablespaces on a standby database when it is in read-only mode:

Note that you must back up the primary database control file, not the standby database control file.

1. Start a SQL*Plus session on the standby database and take the database out of managed recovery mode:

```
RECOVER MANAGED STANDBY DATABASE CANCEL
```

2. Open the database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

3. Take backups of some tablespaces using O/S utilities. You should not back up the standby control file.

Minimize the time that the database is down. For example, to back up datafiles `tbs11.f`, `tbs12.f`, and `tbs13.f` in tablespace `TBS_1` on UNIX you might enter:

```
% cp /disk1/oracle/dbs/tbs11.f /disk2/backup/tbs11.bk
% cp /disk1/oracle/dbs/tbs12.f /disk2/backup/tbs12.bk
% cp /disk1/oracle/dbs/tbs13.f /disk2/backup/tbs13.bk
```

4. Terminate all active user sessions on the standby database.
5. Issue the following statement:

```
RECOVER MANAGED STANDBY DATABASE # you can also set the TIMEOUT option
```

6. Back up the control file on the primary database using an O/S utility.
7. Repeat the above steps until you have backed up each tablespace in the database.

Opening a Standby Database in Read-Only Mode

The read-only mode allows users to query an open database, thereby eliminating the potential for online data modifications. This functionality enables you to use your standby database as a temporary reporting database. Temporary tablespaces allow you to add tempfile entries in read-only mode for the purposes of making queries. Adding and modifying tempfiles will not generate redo entries.

If you maintain your standby database primarily for disaster prevention, you should not rely too heavily on your standby database as a source of information. If a disaster does occur, you will be forced to activate it quickly and hence immediately cease all user activity. Furthermore, using a standby database for queries makes it unavailable for managed recovery. At some point, you will need to run a recovery operation against the standby to resynchronize it with the primary. This action limits the standby's role as a disaster recovery database.

If you need both disaster prevention and a standby available for queries, you can maintain multiple standby databases, some read-only and some in managed recovery mode. You will need to resynchronize the read-only database, but the recovery mode databases give you protection against disaster.

See Also: For more information about using tempfiles and temporary tablespaces, see the *Oracle8i Administrator's Guide*.

To open the standby database in read-only mode when the database is shut down:

1. Use SQL*Plus to start the Oracle instance for the standby database without mounting it:

```
STARTUP NOMOUNT pfile=initSTANDBY.ora
```

2. Mount the standby database:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```

3. Open the database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

To open the standby database in read-only mode when in manual recovery mode:

1. Cancel the recovery by entering the following (terminate the flow of archived redo logs to get the prompt):

```
RECOVER CANCEL
```

2. Open the database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

To open the standby database in read-only mode when in managed recovery mode:

1. Start a SQL*Plus session and execute the following:

```
RECOVER MANAGED STANDBY DATABASE CANCEL
```

2. Open the database in read-only mode:

```
ALTER DATABASE OPEN READ ONLY;
```

To move standby database from read-only mode back to managed recovery mode:

1. Terminate all active user sessions on the standby database.
2. Issue the following statement:

```
RECOVER MANAGED STANDBY DATABASE # you can also set the TIMEOUT option
```

To move standby database from read-only mode back to manual recovery mode:

1. Terminate all active user sessions on the standby database.
2. Issue the following statement:

```
RECOVER STANDBY DATABASE # you can also set the TIMEOUT option
```

Activating a Standby Database

You should not activate the standby database unless it is an emergency. Once activated, the standby database becomes a normal production database and loses its standby status.

Depending on the nature of the disaster, you may not have access to your primary database files. If you do have access, you should do the following if possible:

1. Archive your primary database online redo logs:


```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```
2. Transfer them to your standby site.
3. Apply them *before* activating your standby database.

Following this procedure rolls forward your standby database to the time immediately before the failure of your primary database. You can apply any redo log other than the current redo log to the standby database. If you have lost your non-current online redo logs and they have not been archived, then activate the standby database without recovering the transactions from the unarchived redo logs of the primary database.

After you activate your standby database, reset the online redo logs. Note that the redo logs from the standby database and primary database are now incompatible. Also, the standby database is not mounted when activated; therefore, you are unable to look at tables and views immediately after activation.

To activate a standby database:

1. Ensure that your standby database is mounted in EXCLUSIVE mode.
2. Activate the standby database:

```
ALTER DATABASE ACTIVATE STANDBY DATABASE;
```
3. Shut down your standby instances:

```
SHUTDOWN IMMEDIATE
```
4. As soon as possible, back up your new production database. At this point, the former standby database is now your production database. This task, while not required, is a recommended safety measure because you cannot recover changes made after activation without a backup.
5. Start the new production instance in read-write or read-only mode:

```
STARTUP MOUNT
ALTER DATABASE READ ONLY; # opens the database in read-only mode
ALTER DATABASE READ WRITE; # opens the database in read-write mode
```

Note: After you activate your standby database, you lose all transactions from unarchived logs at your original production database.

Altering the Physical Structure of the Primary Database

The following sections describe the effects of primary database structural alterations on a standby database.

This section contains the following topics:

- [Adding Datafiles](#)
- [Renaming Datafiles](#)
- [Altering Redo Logs](#)
- [Altering Control Files](#)

- [Configuring Initialization Parameters](#)
- [Taking Datafiles in the Standby Database Offline](#)
- [Performing Direct Path Operations](#)
- [Refreshing the Standby Database Control File](#)

Adding Datafiles

Adding a datafile to your primary database generates redo data that, when applied at your standby database, automatically adds the datafile name to the standby control file. If the standby database locates the new file with the new filename, the recovery process continues. If the standby database is unable to locate the new datafile, recovery terminates.

If the recovery process stops, perform the procedure below. Note that if you do not want the new datafile in the standby database, you can take it offline using the following syntax:

```
ALTER DATABASE DATAFILE 'filename' OFFLINE DROP;
```

To add a tablespace or datafile to the primary database and transmit to the standby:

1. Create a tablespace on the primary database as usual. For example, to create new datafile `t_db2.f` in tablespace `TBS_2` issue:

```
CREATE TABLESPACE tbs_2 DATAFILE 't_db2.f' SIZE 2M;
```

2. Copy the newly created empty datafile to the standby site. For example, if the databases are on the same host, you might enter:

```
% cp t_db2.f /private1/stby/t_db2.f
```

3. Start the standby instance without mounting it. For example, enter:

```
STARTUP NOMOUNT pfile=/private1/stby/initSTANDBY.ora
```

4. Mount the standby database, then place it in managed recovery mode:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
RECOVER MANAGED STANDBY DATABASE
```

5. Switch redo logs on the primary to initiate redo archival to standby database:

```
ALTER SYSTEM SWITCH LOGFILE;
```

6. After all archived redo logs have been applied, cancel managed recovery:

```
RECOVER MANAGED STANDBY DATABASE CANCEL
```

Applying CREATE TABLESPACE redo adds the new filename to the standby control file. The following `alert.log` entry is generated;

```
WARNING! Recovering data file 2 from a fuzzy file. If not the current file it might
be an online backup taken without entering the begin backup command.
Successfully added datafile 2 to media recovery
Datafile #2: '/private1/stby/t_db2.f'
```

7. Create the datafile on the standby database. For example, issue;

```
ALTER DATABASE CREATE DATAFILE '/private1/stby/t_db2.f' AS '/private1/stby/t_db2.f';
```

8. Place the standby in managed recovery mode:

```
RECOVER MANAGED STANDBY DATABASE
```

Continue normal processing on the primary database. The primary and standby databases are now synchronized.

See Also: For more information on offline datafile alterations, see ["Taking Datafiles in the Standby Database Offline"](#) on page 16-27.

Renaming Datafiles

Datafile renames on your primary database do not take effect at the standby database until the standby database control file is refreshed. To keep the datafiles at your primary and standby databases synchronized when you rename primary database datafiles, perform analogous operations on the standby database.

Altering Redo Logs

You can add redo log file groups or members to the primary database without affecting your standby database. Similarly, you can drop log file groups or members from the primary database without affecting your standby database. Enabling and disabling of threads at the primary database has no effect on the standby database.

You may want to keep the online redo log configuration the same at the primary and standby databases. If so, when you enable a log file thread with the ALTER DATABASE ENABLE THREAD statement at the primary database, create a new control file for your standby database before activating it. See ["Refreshing the Standby Database Control File"](#) on page 16-29 for procedures.

If you clear log files at the primary database by issuing the `ALTER DATABASE CLEAR UNARCHIVED LOGFILE` statement, or open the primary database using the `RESETLOGS` option, you invalidate the standby database. Because the standby database recovery process will not have the archived logs it requires to continue, you will need to re-create the standby database.

Altering Control Files

If you use the `CREATE CONTROLFILE` command at the primary database to perform any of the following operations, you may invalidate the standby database's control file:

- Change the maximum number of redo log file groups or members.
- Change the maximum number of instances that can concurrently mount and open the database.

If you have invalidated the standby database's control file, re-create it using the procedures in "[Refreshing the Standby Database Control File](#)" on page 16-29.

Using the `CREATE CONTROLFILE` command with the `RESETLOGS` option on your primary database will force the next open of the primary database to reset the online logs, thereby invalidating the standby database.

Configuring Initialization Parameters

Most initialization parameters at your primary and standby databases should be identical. Specific initialization parameters such as `CONTROL_FILES` and `DB_FILE_NAME_CONVERT` should be changed. Differences in other initialization parameters may cause performance degradation at the standby database, and in some cases, bring standby database operations to a halt.

The following initialization parameters play a key role in the standby database recovery process:

Table 16–2 Configuring Standby Initialization Parameters

Parameter	Guideline
COMPATIBLE	This parameter must be the same at the primary and standby databases. If it is not, you may not be able to apply the logs from your primary database to your standby database. You must set the COMPATIBLE parameter to 8.1 or higher if you want to open your standby database in read-only mode.
DB_FILES	MAXDATAFILES must be the same at both databases so that you allow the same number of files at the standby as you allow at the primary database.
CONTROL_FILES	This parameter must be different between the primary and standby databases. The names of the control files that you list in this parameter for the standby database must exist at the standby database.
DB_FILE_NAME_CONVERT	Set when you want to make your standby datafile filenames distinguishable from your primary database filenames. For more information, see "Converting Filenames for Datafiles and Archived Redo Logs" on page 16-15.
LOG_FILE_NAME_CONVERT	Set when you want to make your standby log filenames distinguishable from your primary database log filenames. For more information on this parameter see "Converting Filenames for Datafiles and Archived Redo Logs" on page 16-15.
STANDBY_ARCHIVE_DEST	<p>This parameter is used solely by the standby RFS process to determine the directory in which to place the archived logs. Oracle uses this value along with LOG_ARCHIVE_FORMAT to generate the log filename for the standby site. Oracle stores the fully qualified filenames in the standby control file (query V\$ARCHIVED_LOG for this data). Managed recovery uses this information to drive the recovery operation.</p> <p>The RECOVER STANDBY DATABASE commands (excluding the MANAGED option) rely on either LOG_ARCHIVE_DEST to provide the location of the archived files or a user-entered filename. If a log is missing at the standby site, i.e., the RFS has not recorded its name in the standby control file, and the managed recovery operation fails, you must issue RECOVER STANDBY DATABASE. This statement requires you to use the LOG_ARCHIVE_DEST parameter to locate the archived log.</p> <p>For a managed standby database, set the parameters STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_DEST to the same value. If manual recovery is required, copy the missing archived log to the same location as all the other archived logs, run the manual standby recovery operation, and place the standby back into managed recovery mode.</p>

See Also: For more information on initialization parameters, see the *Oracle8i Reference*.

Taking Datafiles in the Standby Database Offline

You can take standby database datafiles offline as a means to support a subset of your primary database's datafiles. For example, you may decide that it is undesirable to recover the primary database's temporary tablespaces on the standby database.

Take the datafiles offline using the following statement on the standby database:

```
ALTER DATABASE DATAFILE 'filename' OFFLINE DROP;
```

If you execute this statement, then the tablespace containing the offline files must be dropped after opening the standby database.

Performing Direct Path Operations

When you perform a direct load originating from any of the following, the performance improvement applies *only* to the primary database (there is no corresponding recovery process performance improvement on the standby database):

- Direct path load
- CREATE TABLE via subquery
- CREATE INDEX on the primary database

The standby database recovery process still sequentially reads and applies the redo information generated by the unrecoverable direct load.

Primary database processes using the UNRECOVERABLE option are not propagated to the standby database because these processes do not appear in the archived redo logs. To propagate such processes to your standby database, perform any *one* of the following tasks:

- Take the affected datafiles offline in the standby database and drop the tablespace after activation (see ["Taking Datafiles in the Standby Database Offline"](#) on page 16-27).
- Re-create the standby database from a new database backup (see ["Creating a Standby Database"](#) on page 16-3).
- Back up the affected tablespace and archive the current logs in the primary database, transfer the datafiles to the standby database, and resume standby recovery. This is the same procedure that you would perform to guarantee ordinary database recoverability after an UNRECOVERABLE operation.

If you perform an UNRECOVERABLE operation at the primary database and then attempt to recover at the standby database, you will not receive error messages during recovery; instead, such error messages appear in the standby database alert log. Check the standby database alert log periodically.

If you attempt to read a block at the standby site that was loaded with the UNRECOVERABLE option, the following error message is displayed:

```
26040, 00000, "Data block was loaded using the NOLOGGING option\n"
/* Cause: Trying to access data in block that was loaded without
/*      redo generation using the NOLOGGING/UNRECOVERABLE option
/* Action: Drop the object containing the block.
```

See Also: For more details, see ["Taking Datafiles in the Standby Database Offline"](#) on page 16-27.

Note: Blocks loaded using the UNRECOVERABLE option will be marked logically corrupt during recovery at the standby site. Querying these data blocks will result in an error message.

Determining Whether a Backup is Required After UNRECOVERABLE Operations

If you have performed UNRECOVERABLE operations on your primary database, use the V\$DATAFILE view to determine the SCN or time at which Oracle generated the most recent invalidation redo data.

Issue the following SQL command to determine whether you need to perform another backup:

```
SELECT unrecoverable_change#, to_char(unrecoverable_time, 'mm-dd-yyyy hh:mi:ss')
FROM v$datafile;
```

If the query reports an unrecoverable time for a datafile that is more recent than the time when the datafile was last backed up, then make another backup of the datafile in question.

See Also: For more information about the V\$DATAFILE view, see the *Oracle8i Reference*.

Refreshing the Standby Database Control File

The following steps describe how to refresh, or create a copy, of changes you have made to the primary database control file. Refresh the control file after making major structural changes to the primary database such as adding or dropping files.

To refresh the standby database control file:

1. Start a SQL*Plus session on the standby instance and issue the CANCEL command on the standby database to halt its recovery process.

```
RECOVER CANCEL # for manual recovery mode
RECOVER MANAGED STANDBY DATABASE CANCEL # for managed recovery mode
```

2. Shut down the standby instances:

```
SHUTDOWN IMMEDIATE
```

3. Start a SQL*Plus session on the production instance and create the control file for the standby database:

```
ALTER DATABASE CREATE STANDBY CONTROLFILE AS 'filename';
```

4. Archive the current online redo log of your primary database:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

5. Transfer the standby control file and archived log files to the standby site using an O/S utility appropriate for binary files.

6. Connect to the standby instance and mount (but do not open) the standby database:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```

7. Restart the recovery process on the standby database:

```
RECOVER STANDBY DATABASE # recovers using location for logs in init.ora
RECOVER FROM 'location' STANDBY DATABASE # recovers using specified location
```

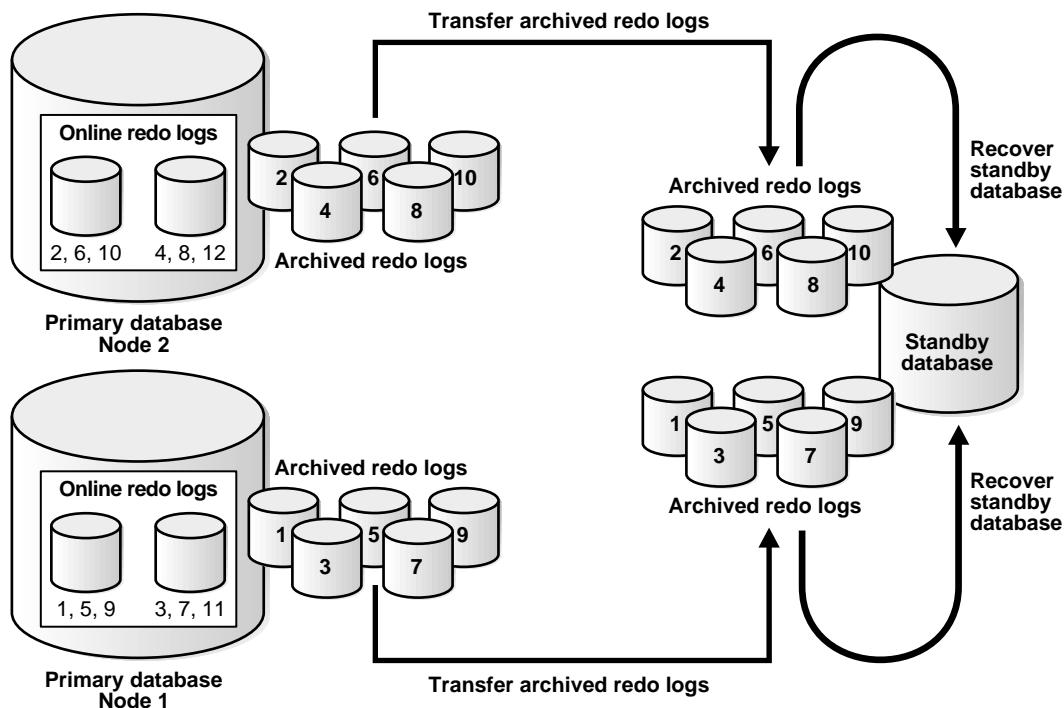
Using a Standby Database in an OPS Configuration

You can use a standby database in conjunction with the Oracle multi-threaded server option. The following table describes the possible legal and illegal combinations of nodes in the primary and standby databases:

	Single-Instance Standby	Multi-Instance Standby
Single-Instance Primary	Yes	No
Multi-Instance Primary	Yes	Yes

In each scenario, each node of the primary database transmits its own thread of archived redo logs to the standby database. For example, [Figure 16-3](#) illustrates an OPS database with two nodes transmitting redo logs to a single-instance standby:

Figure 16-3 *Transmitting Archived Logs from a Multi-Instance Primary Database*



In this case, node 1 of the primary database transmits logs 1,3,5,7,9 while node 2 transmits logs 2,4,6,8,10. If the standby database is in managed recovery mode, it automatically determines the correct order in which to apply the archived redo logs.

If both your primary and standby databases are in an OPS configuration, and the standby database is in managed recovery mode, then a single node of the standby database applies all sets of logs transmitted by the primary nodes. In this case, the standby nodes that are *not* applying redo cannot be in read-only mode while managed recovery is in progress; in most cases, the non-recovery nodes should be shut down, although they can also be mounted.

See Also: For information about configuring a database for OPS, see the *Oracle8i Parallel Server Setup and Configuration Guide*.

Performing Tablespace Point-in-Time Recovery with Recovery Manager

This chapter describes how to use Recovery Manager (RMAN) to perform tablespace point-in-time recovery (TSPITR), and includes the following topics:

- [Introduction to RMAN TSPITR](#)
- [Planning for TSPITR](#)
- [Preparing the Auxiliary Instance for TSPITR](#)
- [Performing TSPITR](#)
- [Preparing the Target Database for Use After TSPITR](#)
- [Responding to Unsuccessful TSPITR](#)
- [Tuning TSPITR Performance](#)

Introduction to RMAN TSPITR

Recovery Manager (RMAN) automated tablespace point-in-time recovery (TSPITR) enables you to quickly recover one or more tablespaces to a time that is different from that of the rest of the database.

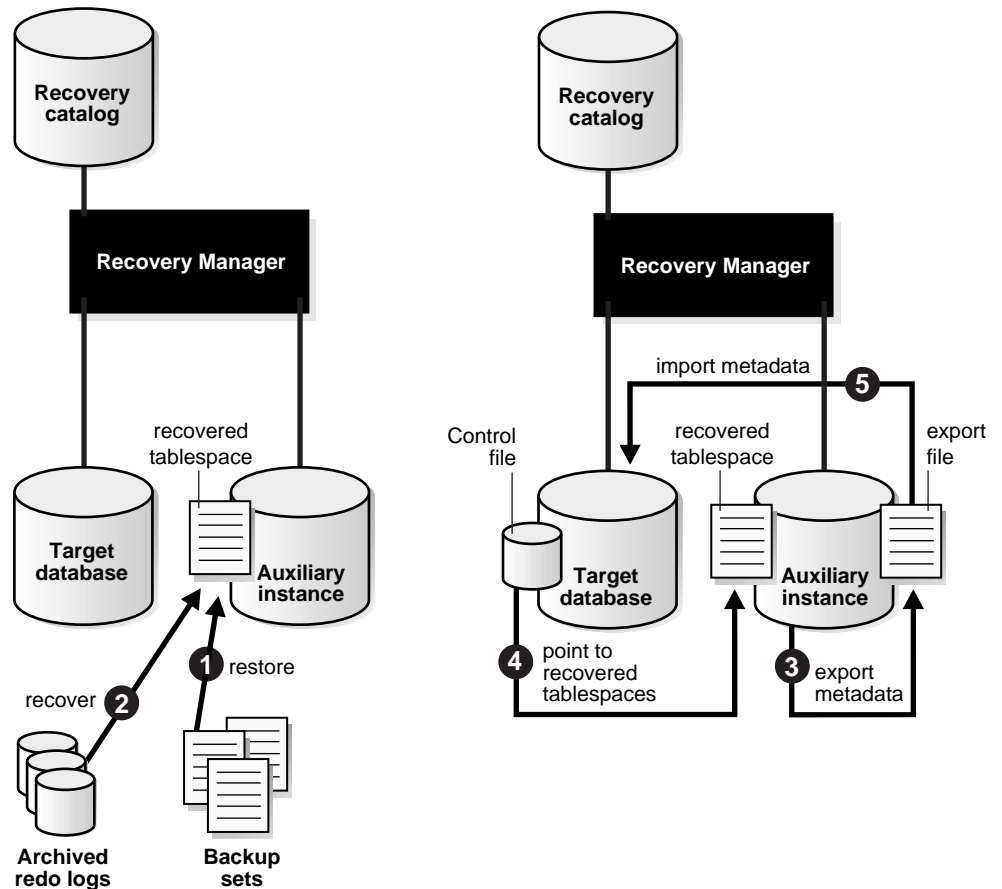
RMAN TSPITR is most useful for recovering:

- An erroneous DROP TABLE or TRUNCATE TABLE operation.
- A table that has become logically corrupted.
- An incorrect batch job or other DML statement that has affected only a subset of the database.
- A logical database to a point different from the rest of the physical database when multiple logical databases exist in separate tablespaces of one physical database.

Like a table export, RMAN TSPITR enables you to recover a consistent data set; however, the data set is the entire tablespace rather than one object. As [Figure 16-4](#) illustrates, Recovery Manager does the following:

1. Restores the specified tablespace backups to a temporary auxiliary instance.
2. Recovers the tablespace.
3. Exports metadata from the auxiliary instance.
4. Points the target database control file to the newly recovered datafiles.
5. Imports metadata into the target database.

Figure 16–4 RMAN TSPITR



TSPITR Terminology

Familiarize yourself with the following terms and abbreviations, which are used throughout this chapter:

TSPITR

Tablespace point-in-time recovery

Auxiliary Instance

The auxiliary instance used to recover the backup tablespaces. The database created by TSPITR never has independent existence: it is only an intermediate work area.

Recovery Set

Tablespaces requiring TSPITR to be performed on them.

Auxiliary Set

Any other items required for TSPITR, including:

- Backup control file
- SYSTEM tablespace
- Datafiles containing rollback segments
- Temporary tablespace (optional). A small space is required by Export for sort operations (for more information on sort space issues, see "[Responding to Unsuccessful TSPITR](#)" on page A-12).

Planning for TSPITR

Recovery Manager TSPITR requires careful planning. Before proceeding, read this chapter thoroughly.

This section covers the following topics:

- [Performing TSPITR Without a Recovery Catalog](#)
- [Understanding General Restrictions](#)
- [Researching and Resolving Inconsistencies](#)
- [Managing Data Relationships](#)

Note: Many of the limitations and planning steps in this chapter can also be found in [Chapter B, "Performing Operating System Tablespace Point-in-Time Recovery"](#); however, differences in limitations and planning exist. These differences are explicitly called to your attention in this chapter.

Performing TSPITR Without a Recovery Catalog

You can perform RMAN TSPITR either with or without a recovery catalog. If you do *not* use a recovery catalog, note these restrictions:

- Because RMAN has no historical record of the rollback segments in TSPITR, RMAN assumes that the current rollback segments were the same segments present at the time to which recovery is performed.

- If RMAN recovers to a remote time, Oracle may have reused the records of the copies and backups, thus making it impossible to perform TSPITR.

Understanding General Restrictions

When performing RMAN TSPITR, you *cannot*:

- Run the target and auxiliary databases on separate nodes. The target and auxiliary databases can be in a cluster configuration, however, using shared disks.
- Recover dropped tablespaces.
- Recover a tablespace that has been dropped and re-created with the same name.
- Remove a datafile that has been added to a tablespace. If the file was added after the point to which RMAN is recovering, then the file will still be part of the tablespace (and will be empty) after RMAN TSPITR is complete.
- Issue DML statements on the auxiliary database—the auxiliary database is for recovery only.
- Use existing backups of the recovery set datafiles for recovery after TSPITR is complete. Instead, take fresh backups of the recovered files. If you attempt to recover using a backup taken prior to performing TSPITR, recovery fails.
- Recover optimizer statistics for objects that have had statistics calculated on them; re-calculate statistics after performing TSPITR.
- Place any of the following objects within the recovery set:
 - Replicated master tables
 - Tables with VARRAY columns
 - Tables with nested tables
 - Tables with external files
 - Snapshot logs
 - Snapshot tables
 - Objects owned by SYS (including rollback segments)

WARNING: Do not perform RMAN TSPITR for the first time on a production system or when there is a time constraint.

Researching and Resolving Inconsistencies

The primary issue for RMAN TSPITR is the possibility of application-level inconsistencies between tables in recovered and unrecovered tablespaces due to implicit rather than explicit referential dependencies. Note the following issues and have the means to resolve possible inconsistencies before proceeding:

- [RMAN Only Supports Recovery Sets Containing Whole Tables](#)
- [Recovery Manager Does Not Support Tables Containing Rollback Segments](#)
- [TS_PITR_CHECK Does Not Check for Objects Owned by SYS](#)

RMAN Only Supports Recovery Sets Containing Whole Tables

Note: This limitation is specific to RMAN TSPITR.

RMAN TSPITR only supports recovery sets that contain whole tables. For example, if you perform RMAN TSPITR on partitioned tables and spread partitions across multiple tables, RMAN returns an error message during the export phase. Recovery sets that contain either tables without their constraints or the constraints without the table also result in errors.

Recovery Manager Does Not Support Tables Containing Rollback Segments

Note: This limitation is specific to RMAN TSPITR.

If you are performing O/S TSPITR, you can take rollback segments in the recovery set offline—thus preventing changes being made to the recovery set before recovery is complete. RMAN TSPITR does not support recovery of tablespaces containing rollback segments. For more information about TSPITR and rollback segments, see ["Step 3: Prepare the Primary Database"](#) on page B-9.

TS_PITR_CHECK Does Not Check for Objects Owned by SYS

The TS_PITR_CHECK view provides information on dependencies and restrictions that can prevent TSPITR from proceeding. TS_PITR_CHECK does not provide information, however, about dependencies and restrictions for objects owned by SYS.

If any objects—including rollback segments—owned by SYS are in the recovery set, then there is no guarantee that you can successfully recover these objects. TSPITR utilizes the Export and Import utilities, which do not support objects owned by SYS. To find out which recovery set objects are owned by SYS, issue the following statement:

```
SELECT object_name, object_type
FROM sys.dba_objects
WHERE tablespace_name IN ('tablespace_name_1', 'tablespace_name_2',
                          'tablespace_name_n')
AND owner = 'SYS';
```

See Also: For more details about the TS_PITR_CHECK view, see ["Step 2: Research and Resolve Dependencies on the Primary Database"](#) on page B-7.

Managing Data Relationships

TSPITR provides views that can detect any data relationships between objects in the recovery set and objects in the rest of the database. TSPITR will not successfully complete unless these relationships are managed, either by removing or suspending the relationship or by including the related object within the recovery set.

See Also: For more information see ["Step 2: Research and Resolve Dependencies on the Primary Database"](#) on page B-7.

Preparing the Auxiliary Instance for TSPITR

Satisfy the following requirements before performing RMAN TSPITR:

- [Create an Oracle Password File for the Auxiliary Instance](#)
- [Create a Parameter File for the Auxiliary Instance](#)
- [Start the Auxiliary Instance](#)
- [Ensure Net8 Connectivity to the Auxiliary Instance](#)
- [Start the Recovery Manager Command Line Interface](#)

Create an Oracle Password File for the Auxiliary Instance

For information about creating and maintaining Oracle password files, see the *Oracle8i Administrator's Guide*.

Create a Parameter File for the Auxiliary Instance

Create an `init.ora` file for the auxiliary instance and set the following required parameters:

Parameter	Specify
DB_NAME	The same name as the target database.
LOCK_NAME_SPACE	A value different from any database in the same \$ORACLE_HOME. For simplicity, specify <code>_DBNAME</code> .
DB_FILE_NAME_CONVERT	Patterns to convert filenames for the datafiles of the auxiliary database. You can use this parameter to generate filenames for those files that you did not name using <code>set auxname</code> .
LOG_FILE_NAME_CONVERT	Patterns to convert filenames for the online redo logs of the auxiliary database.
CONTROL_FILES	A different value from the CONTROL_FILES parameter in the target parameter file.

Set other parameters as needed, including the parameters that allow you to connect as SYSDBA through Net8.

Following are examples of the `init.ora` parameter settings for the auxiliary instance.

```
DB_NAME=prod1
LOCK_NAME_SPACE=_prod1
CONTROL_FILES=/oracle/aux/cf/aux_prod_cf.f
DB_FILE_NAME_CONVERT=("/oracle/prod/datafile", "/oracle/aux/datafile")
LOG_FILE_NAME_CONVERT=("/oracle/prod/redo_log", "/oracle/aux/redo_log")
```

See Also: For details about DB_FILE_NAME_CONVERT, see "[Tuning TSPITR Performance](#)" on page A-13. For more information about Net8, see the *Net8 Administrator's Guide*.

Note: After setting these parameters, ensure that you do not overwrite the `init.ora` settings for the production files at the target database.

Start the Auxiliary Instance

Before beginning RMAN TSPITR, use SQL*Plus to connect to the auxiliary instance and start it in NOMOUNT mode (specifying a parameter file if necessary):

```
SQL> connect sys/aux_pwd@aux_str;
SQL> startup nomount pfile='/oracle/aux/dbs/initAUX.ora';
```

Because the auxiliary instance does not yet have a control file, you can only start the instance in NOMOUNT mode. Do not create a control file or try to mount or open the auxiliary instance for TSPITR.

Ensure Net8 Connectivity to the Auxiliary Instance

The auxiliary instance must have a valid net service name. Before proceeding, use SQL*Plus to ensure that you can establish a connection to the auxiliary instance.

Start the Recovery Manager Command Line Interface

Use one of the following methods to start the RMAN command line interface:

- [Connect from the O/S Command Line](#)
- [Connect from the RMAN Prompt](#)

Connect from the O/S Command Line To connect to the auxiliary instance, target instance, and optional recovery catalog, supply the following information when starting Recovery Manager:

```
% rman target sys/target_pwd@target_str catalog rman/cat_pwd@cat_str auxiliary \
> sys/aux_pwd@aux_str
```

Where:

<i>sys</i>	User with SYSDBA privileges
<i>rman</i>	Owner of the recovery catalog
<i>target_pwd</i>	The password for connecting as SYSDBA specified in the target database's <code>orapwd</code> file
<i>target_str</i>	The net service name for the target database
<i>cat_pwd</i>	The password for user RMAN specified in the recovery catalog's <code>orapwd</code> file
<i>cat_str</i>	The net service name for the recovery catalog database

aux_pwd The password for connecting as SYSDBA specified in the auxiliary database's `orapwd` file.

aux_str The net service name for the auxiliary database.

Connect from the RMAN Prompt You can start the RMAN command line interface without a connection to the auxiliary instance, and then use the **connect** command at the RMAN prompt:

```
% rman
RMAN> connect auxiliary sys/aux_pwd@aux_str
RMAN> connect target target sys/target_pwd@target_str
RMAN> connect catalog rman/cat_pwd@cat_str
```

Performing TSPITR

After you have completed all planning requirements, perform RMAN TSPITR. Issue the following commands within **run**, where *tablespace_list* is the list of tablespace names in the recovery set and *recovery_end_time* is the point to which you want to recover:

```
allocate auxiliary channel . . .
recover tablespace tablespace_list until recovery_end_time;
```

You must allocate at least one auxiliary channel with the **allocate auxiliary channel** command.

Note: The tablespace recovery set should not contain the SYSTEM tablespace or any tablespace with rollback segments.

The following example statement performs RMAN TSPITR on tablespaces TBS_2 and TBS_3 to 8 p.m. on January 10, 1999:

```
run {
  allocate auxiliary channel dev1 type 'sbt_tape';
  recover tablespace tbs_2, tbs_3 until time 'Jan 10 1999 20:00:00';
}
```

Recovery Manager automatically performs the following steps during TSPITR:

1. Restores the datafiles to the auxiliary instance.
2. Recovers the restored datafiles to the specified time.
3. Opens the auxiliary database with the RESETLOGS option.

4. Exports the dictionary metadata about objects in the recovered tablespaces—the DDL to create the objects along with pointers to the physical locations of those in the recovered datafiles—to the target database.
5. Closes the auxiliary database.
6. Issues **switch** commands so that the target control file now points to the datafiles in the recovery set that were just recovered at the auxiliary database.
7. Imports the dictionary metadata that was exported from the auxiliary database, allowing the recovered objects to be accessed.

Note: RMAN attempts to find datafile copies instead of restoring the datafiles being recovered. If it finds none, it performs a restore operation and does not execute a switch. If you have configured names for the datafiles with the **set auxname** command, and suitable datafile copies exist in those **auxname** locations, RMAN optimizes away the restore and performs a switch to the **auxname** datafile copy.

Preparing the Target Database for Use After TSPITR

The tablespaces in the recovery set remain offline until after RMAN TSPITR completes successfully.

To prepare the target database for re-use after TSPITR:

1. Make backups of tablespaces in the recovery set before bringing these tablespaces online. Note that all previous backups of datafiles in the recovery set are no longer valid. For example, this command backs up tablespace TBS_4:

```
run {
  allocate channel ch1 type disk;
  backup tablespace tbs_4;
}
```

2. Bring the recovered tablespaces online. For example, enter:

```
sql "ALTER TABLESPACE TBS_4 ONLINE";
```

3. Because the auxiliary database is not usable after a successful RMAN TSPITR, release the memory by shutting down the database:

```
shutdown abort;
```

4. Delete the following:
 - Auxiliary set datafiles restored to temporary locations during RMAN TSPITR
 - Auxiliary database control files
 - Auxiliary database redo log files

Responding to Unsuccessful TSPITR

A variety of problems can cause TSPITR to abort. For example, if there is a conflict between the target database and the converted filename, you will have to shut down the auxiliary instance, correct the converted datafile name, issue a **startup nomount**, and then run RMAN TSPITR again.

Another possible cause for failure is a lack of sufficient sort space for the Export utility. In this case, you will need to edit the `recover.txt` file (in UNIX, it is located in `$ORACLE_HOME/admin`). This file contains the following:

```
#
# tsiptr_7: do the incomplete recovery and resetlogs. This member is used once.
#
define tspitr_7
<<<
# make the controlfile point at the restored datafiles, then recover them
recover clone database tablespace &l&;
sql clone "alter database open resetlogs";
# PLUG HERE the creation of a temporary tablespace if export fails due to lack of
# temporary space.
# For example in Unix these two lines would do that:
#sql clone "create tablespace aux_tspitr_tmp
#          datafile '/tmp/aux_tspitr_tmp.dbf' size 500K";
}
>>>
```

Remove the '#' symbols from the last two lines of comments and modify the statement to create a temporary tablespace. Retry the TSPITR operation, increasing the size of the tablespace until the export operation succeeds.

If TSPITR is unsuccessful for some reason, follow the procedure below.

To respond to unsuccessful TSPITR:

1. Should RMAN TSPITR be unsuccessful, shut down the auxiliary instance:

```
shutdown abort;
```

2. Identify and correct the error.
3. Start the auxiliary instance without mounting it. For example, enter:

```
startup nomount pfile=initAUX.ora;
```

4. Perform TSPITR again as in ["Performing TSPITR"](#) on page A-10.

Tuning TSPITR Performance

This section describes steps you can take to tune the performance of RMAN TSPITR:

- [Specify a New Name for Datafiles in Auxiliary Set Tablespaces](#)
- [Set the Auxiliary Name and Use a Datafile Copy for Recovery Manager TSPITR](#)
- [Use the Converted Filename in the Auxiliary Control File](#)
- [Summary: Datafile Naming Methods](#)

Specify a New Name for Datafiles in Auxiliary Set Tablespaces

Recovery Manager restores and recovers all datafiles belonging to the tablespaces in the recovery set and auxiliary set at the auxiliary instance. Note that the auxiliary set includes the SYSTEM tablespace plus all the tablespaces with rollback segments.

Specify a new name for any datafiles in the auxiliary set tablespace using the **set newname** Recovery Manager command. RMAN uses this new name as the temporary location in which to restore and recover the datafile. This new name will also override the setting in the `DB_FILE_NAME_CONVERT` parameter in the `init.ora` file. For example, to rename datafile 2 to `new_df_name.f` enter:

```
set newname for datafile 2 to '/oracle/dbs/new_df_name.f';
```

You can specify a new name for any datafiles in recovery set tablespaces. If you specify a new name, the datafiles will replace the original datafiles in the target control file—so the new filenames replace the existing filenames.

When setting new filenames, RMAN does not check for conflicts between datafile names at the auxiliary and target databases. Any conflicts will result in an RMAN error during TSPITR.

Set the Auxiliary Name and Use a Datafile Copy for Recovery Manager TSPITR

Using a datafile copy on disk is much faster than restoring a datafile. Hence, you may wish to use an appropriate copy of a datafile in the recovery or auxiliary set instead of restoring and recovering a datafile.

Recovery Manager uses a datafile copy if the following conditions are met:

1. The datafile copy name is registered in the recovery catalog as the auxiliary name of the corresponding datafile via the following command (where *filename* is the datafile name or number, and *auxiliary_datafile_name* is the datafile auxiliary name):

```
set auxname for datafile filename to auxiliary_datafile_name;
```

2. The datafile copy was made before the time specified in the *untilClause* using the following RMAN command (where '*filename*' is the datafile filename):

```
run {  
    copy datafile 'filename' to auxname;  
    ...  
}
```

Examples

The following commands are examples of the conditions required by Recovery Manager:

```
set auxname for datafile '/oracle/prod/datafile_1_1.dbf'  
to '/oracle/prod_copy/datafile_1_1.dbf';  
  
run {  
    allocate channel ch1 type disk;  
    copy datafile '/oracle/prod/datafile_1_1.dbf'  
    to auxname;  
}
```

Recovery Manager will not use a datafile copy if you use **set newname** for the same datafile.

If Recovery Manager uses a datafile copy and TSPITR completes successfully, the *auxiliary_datafile_name* will be marked **deleted** in the recovery catalog. The original datafile at the target will be replaced by this datafile copy after RMAN TSPITR completes.

Use the Converted Filename in the Auxiliary Control File

If neither a new name nor auxiliary name is set for a datafile in an auxiliary set tablespace, Recovery Manager can use the converted filename specified in the auxiliary database control file to perform the restore and recovery. Recovery Manager checks for conflicts between datafile names at the auxiliary and target databases. Any conflicts result in an error.

If neither a new name or auxiliary name is set for a datafile in a recovery set tablespace, or the file at the auxiliary name is unusable, Recovery Manager uses the original location of the datafile.

Summary: Datafile Naming Methods

The following commands and parameters are used to name datafiles in the auxiliary and recovery sets during TSPITR. The order of precedence in the table goes top to bottom, so **set newname** takes precedence over **set auxname** and **DB_FILE_NAME_CONVERT**:

	Command/Parameter	Auxiliary Set	Recovery Set
1	set newname	X	X
2	set auxname	X	X
3	DB_FILE_NAME_CONVERT	X	

If filenames are not converted in the auxiliary set, RMAN signals an error during TSPITR.

Performing Operating System Tablespace Point-in-Time Recovery

Note: Due to the complex nature of tablespace point-in-time recovery, Oracle recommends that you contact Worldwide Customer Support Services before using these procedures.

This chapter describes how to perform O/S tablespace point-in-time recovery (TSPITR), and includes the following topics:

- [Introduction to O/S Tablespace Point-in-Time Recovery](#)
- [Planning for Tablespace Point-in-Time Recovery](#)
- [Preparing the Databases for TSPITR](#)
- [Performing Partial TSPITR of Partitioned Tables](#)
- [Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped](#)
- [Performing TSPITR of Partitioned Tables When a Partition Has Split](#)
- [TSPITR Tuning Considerations](#)
- [Performing TSPITR Using Transportable Tablespaces](#)

Introduction to O/S Tablespace Point-in-Time Recovery

Tablespace Point-in-Time Recovery (TSPITR) enables you to quickly recover one or more non-SYSTEM tablespaces to a time that is different from that of the rest of the database. Like a table export, TSPITR enables you to recover a consistent data set; however, the data set is the entire tablespace rather than just one object.

TSPITR is most useful for recovering:

- An erroneous DROP TABLE or TRUNCATE TABLE operation.
- A table that has become logically corrupted.
- An incorrect batch job or other DML statement that has affected only a subset of the database.
- A logical database to a point different from the rest of the physical database when multiple logical databases exist in separate tablespaces of one physical database.
- A tablespace in a VLDB (very large databases) when TSPITR is more efficient than restoring the whole database from a backup and rolling it forward (see ["Planning for Tablespace Point-in-Time Recovery"](#) on page B-4 before making any decisions).

This section contains the following topics:

- [TSPITR Advantages](#)
- [TSPITR Methods](#)
- [TSPITR Terminology](#)

TSPITR Advantages

Prior to Oracle8, point-in-time recovery could only be performed on a subset of a database by:

1. Creating a copy of the database.
2. Rolling the copied database forward to the desired point in time.
3. Exporting the desired objects from the copied database.
4. Dropping the relevant objects from the production database.
5. Importing the objects into the production database.

There was a performance overhead associated with exporting and importing large objects, however, which created a need for a new method. TSPITR enables you to do the following:

1. Make a temporary copy of the database, called a *clone database*.
2. Recover a subset of a clone database.
3. Copy the relevant datafiles from the recovered database to the production database using an O/S utility.
4. Export data dictionary metadata about the datafile's content (for example, the recovered segments within the file) from the clone database to the production database. The copied file is also added to the production database via a special import option.

TSPITR Methods

You can perform O/S TSPITR in two different ways:

Method	Consequence
Traditional O/S TSPITR	You must follow special procedures for creating clone <code>init.ora</code> files, mounting the clone database, etc. The procedures provide error checks to prevent the corruption of the primary database on the same computer while recovering the clone database.
TSPITR using the transportable tablespace feature	This method differs from standard O/S TSPITR mainly in using transported tablespaces to perform the last step of TSPITR. You must set the COMPATIBLE initialization parameter to 8.1 or higher to use this method.

The one major difference between the two methods is that performing TSPITR via transportable tablespaces relaxes some of O/S TSPITR's special procedures. If you restore backups to a different host separate from the primary database, you can start the clone database as if it were the primary database using the normal database MOUNT command instead of the clone database MOUNT command.

See Also: For more information about the transportable tablespace feature, see the *Oracle8i Administrator's Guide*.

TSPITR Terminology

Familiarize yourself with the following terms and abbreviations, which are used throughout this chapter:

TSPITR

Tablespace point-in-time recovery

Clone Database

The copied database used for recovery in TSPITR. It has various substantive differences from a regular database.

Recovery Set

Tablespaces that require point-in-time recovery to be performed on them.

Auxiliary Set

Any other items required for TSPITR, including:

- Backup control file
- System tablespaces
- Datafiles containing rollback segments
- Temporary tablespace (optional)

A small amount of space is required by export for sort operations. If a copy of the temporary tablespace is not included in the auxiliary set, then provide sort space either by creating a new temporary tablespace after the clone has been started or by setting AUTOEXTEND to ON on the SYSTEM tablespace files.

Transportable Tablespace

A feature that enables you to take a tablespace from one database and plug it in to another database. For more information, see "[Recovering Transported Tablespaces](#)" on page 15-4. For a detailed account, see the *Oracle8i Administrator's Guide*.

Planning for Tablespace Point-in-Time Recovery

TSPITR is a complicated procedure and requires careful planning. Before proceeding you should read this chapter thoroughly.

WARNING: You should not perform TSPITR for the first time on a production system, or during circumstances where there is a time constraint.

TSPITR Limitations

The primary issue you should consider is the possibility of application-level inconsistencies between tables in recovered and unrecovered tablespaces due to implicit rather than explicit referential dependencies. Understand these dependencies and find the means to resolve any possible inconsistencies before proceeding.

This section deals with the following topics:

- [General Restrictions](#)
- [Data Consistency and TSPITR](#)

General Restrictions

TSPITR has several restrictions. You *cannot* use TSPITR to do the following:

- Recover a SYSTEM tablespace.
- Recover dropped tablespaces.
- Recover a tablespace that has been dropped and re-created with the same name.
- Remove a datafile that has been added to the wrong tablespace. If the file was added after the point to which the tablespace is being recovered, then the file will still be part of the tablespace (and will be empty) after TSPITR is complete.
- Use DML statements on the clone database—the clone database is for recovery only.
- Use existing backups of the recovery set datafiles for recovery after TSPITR completes. Instead, you must take fresh backups of the recovered files. If you attempt to recover using a backup taken prior to performing TSPITR, recovery will fail.
- Recover optimizer statistics for objects that have had statistics calculated on them; statistics must be re-calculated after performing TSPITR.
- Include any of the following object types within the TSPITR recovery set:
 - Replicated master tables
 - Snapshot logs
 - Snapshot tables

If any of these objects are included, they will have to be dropped before TSPITR.

Data Consistency and TSPITR

TSPITR provides views that can detect any data relationships between objects that are in the tablespaces being recovered and objects in the rest of the database. TSPITR will not successfully complete unless you manage these relationships, either by removing or suspending the relationship or by including the related object within the recovery set.

See Also: For more information see "[Step 2: Research and Resolve Dependencies on the Primary Database](#)" on page B-7.

TSPITR Requirements

Satisfy the following requirements before performing TSPITR.

- Ensure that all files constituting the recovery set tablespaces are in the recovery set on the clone database, otherwise the export phase of TSPITR will fail.
- Create the control file backup in the auxiliary set using the following SQL statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO 'controlfile_name';
```

This control file backup must be created at a later time than the backup that is being used. If it is not, then you may encounter an error message (ORA-01152, file 1 was not restored from a sufficiently old backup).

- Allocate enough disk space to accommodate the clone database.
- Provide enough real memory to start the clone instance.

See Also: For more information, see "[Step 4: Prepare the Clone Parameter Files](#)" on page B-10.

Preparing the Databases for TSPITR

This section describes how to prepare the clone database for TSPITR, and includes the following steps:

- [Step 1: Determine Whether Objects Will Be Lost](#)
- [Step 2: Research and Resolve Dependencies on the Primary Database](#)
- [Step 3: Prepare the Primary Database](#)
- [Step 4: Prepare the Clone Parameter Files](#)
- [Step 5: Prepare the Clone Database](#)

Step 1: Determine Whether Objects Will Be Lost

When TSPITR is performed on a tablespace, any objects created after the recovery time will be lost. To see which objects will be lost, query the `TS_PITR_OBJECTS_TO_BE_DROPPED` view on the primary database. The contents of the view are described in [Table 16-3](#):

Table 16-3 *TS_PITR_OBJECTS_TO_BE_DROPPED View*

Column Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
NAME	NOT NULL	VARCHAR2(30)
CREATION_TIME	NOT NULL	DATE
TABLESPACE_NAME		VARCHAR2(30)

When querying this view, supply all the elements of the date field, otherwise the default setting will be used. Also, use the `TO_CHAR` and `TO_DATE` functions. For example, with a recovery set consisting of `TS1` and `TS2`, and a recovery point in time of `'1997-06-02:07:03:11'`, issue the following:

```
SELECT owner, name, tablespace_name, to_char(creation_time, 'YYYY-MM-DD:HH24:MI:SS'),
FROM ts_pitr_objects_to_be_dropped
WHERE tablespace_name IN ('TS1','TS2')
AND creation_time > to_date('97-JUN-02:07:03:11','YY-MON- DD:HH24:MI:SS')
ORDER BY tablespace_name, creation_time;
```

See Also: For more information about the `TS_PITR_OBJECTS_TO_BE_DROPPED` view, see the *Oracle8i Reference*.

Step 2: Research and Resolve Dependencies on the Primary Database

Use the `TS_PITR_CHECK` view to identify relationships between objects that overlap the recovery set boundaries. If this view returns rows when queried, investigate and correct the problem. Proceed with TSPITR *only* when `TS_PITR_CHECK` view returns no rows. Record all actions performed during this step so that you can retrace these relationships after completing TSPITR.

The `TS_PITR_CHECK` view will return rows unless you:

- Fully contain all partitions and sub-partitions of a partitioned table in the recovery set. If you need to recover only one or more of the partitions of a partitioned table, convert them to stand-alone tables (see ["Performing Partial TSPITR of Partitioned Tables"](#) on page B-15).

- Fully contain the following in the recovery set:
 - Tables (and their indexes, partitioned or non-partitioned)
 - Clusters (and their indexes, partitioned or non-partitioned)
 - Primary key and foreign key relationships
 - All elements of a LOB (LOB segment, LOB index, and LOB locator)
- Exclude the following object types from the recovery set:
 - Replicated master tables
 - Snapshot logs
 - Snapshot tables

Supply a four-line predicate detailing the recovery set tablespace to query the TS_PITR_CHECK view. For example, with a recovery set consisting of TS1 and TS2, the SELECT statement against TS_PITR_CHECK would be as follows:

```
SELECT * FROM sys.ts_pitr_check
WHERE (ts1_name IN ('TS1','TS2') AND ts2_name NOT IN ('TS1','TS2'))
OR (ts1_name NOT IN ('TS1','TS2') AND ts2_name IN ('TS1','TS2'));
```

Because of the number and width of the columns in the TS_PITR_CHECK view, you may want to format the columns as follows:

```
column OBJ1_OWNER heading "own1"
column OBJ1_OWNER format a4
column OBJ1_NAME heading "name1"
column OBJ1_NAME format a5
column OBJ1_SUBNAME heading "subname1"
column OBJ1_SUBNAME format a8
column OBJ1_TYPE heading "obj1type"
column OBJ1_TYPE format a8 word_wrapped
column TS1_NAME heading "ts1_name"
column TS1_NAME format a8
column OBJ2_NAME heading "name2"
column OBJ2_NAME format a5
column OBJ2_SUBNAME heading "subname2"
column OBJ2_SUBNAME format a8
column OBJ2_TYPE heading "obj2type"
column OBJ2_TYPE format a8 word_wrapped
column OBJ2_OWNER heading "own2"
column OBJ2_OWNER format a4
column TS2_NAME heading "ts2_name"
column TS2_NAME format a8
column CONSTRAINT_NAME heading "cname"
column CONSTRAINT_NAME format a5
```

```
column REASON heading "reason"
column REASON format a57 word_wrapped
```

If the partitioned table TP has two partitions, P1 and P2, which exist in tablespaces TS1 and TS2 respectively, and there is a partitioned index defined on TP called TPIND, which has two partitions ID1 and ID2 (that exist in tablespaces ID1 and ID2 respectively) you would get the following output when TS_PITR_CHECK is queried against tablespaces TS1 and TS2 (assuming appropriate formatting):

```
own1   name1 subname1 obj1type ts1_name name2 subname2 obj2type own2 ts2_name cname reason
---   -
SYSTEM TP   P1      TABLE   TS1     TPIND IP1      INDEX   PARTITION PARTITION SYS
ID1 Partitioned Objects not fully contained in the recovery set

SYSTEM TP   P1      TABLE   TS1     TPIND IP2      INDEX   PARTITION PARTITION SYS
ID2 Partitioned Objects not fully contained in the recovery set
```

The table SYSTEM.TP has a partitioned index TPIND that consists of two partitions, IP1 in tablespace ID1 and IP2 in tablespace ID2. Either drop TPIND or include ID1 and ID2 in the recovery set.

See Also: For more information about the TS_PITR_CHECK view, see the *Oracle8i Reference*.

Step 3: Prepare the Primary Database

Perform the following tasks:

1. Archive the current online redo log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

2. Take offline any rollback segments in the recovery set (you do not have to take auxiliary set rollback segments offline):

```
ALTER ROLLBACK SEGMENT segment_name OFFLINE;
```

3. Take the recovery set tablespaces on the primary database offline normal. Use the OFFLINE FOR RECOVER option if you cannot write to the file due to I/O errors or if the file is unavailable. You can use OFFLINE FOR RECOVER for performance reasons, e.g., if you have a large number of datafiles and you do not care about updating the file headers since they are being recovered back to a point in time anyway:

```
ALTER TABLESPACE tablespace_name OFFLINE FOR RECOVER;
```

This statement prevents changes being made to the recovery set before TSPITR is complete.

Note: If there is a subset of data that is not physically or logically corrupt that you want to query within the recovery set tablespaces, alter the recovery set tablespaces on the primary database as READ ONLY for the duration of the recovery of the clone. Take the recovery set tablespaces offline before integrating the clone files with the primary database (see "[Step 5: Copy the Recovery Set Clone Files to the Primary Database](#)" on page B-13).

See Also: For more information about the ALTER SYSTEM and ALTER ROLLBACK SEGMENT statements, see the *Oracle8i SQL Reference*.

Step 4: Prepare the Clone Parameter Files

Create the parameter file from a new `init.ora` file rather than using the production database `init.ora` file. Save memory by using low settings for parameters such as `DB_BLOCK_BUFFERS`, `SHARED_POOL_SIZE`, or `LARGE_POOL_SIZE`. If the production parameter files are used for the clone database, however, reducing these parameters can prevent the clone database from starting when other parameters are set too high—for example, the parameter `ENQUEUE_RESOURCES`, which allocates memory from within the shared pool.

Set the following parameters in your clone `init.ora` file:

Parameter	Purpose
CONTROL_FILES	Identifies clone control files. Set to the name and location of the clone control files.
LOCK_NAME_SPACE	Allows the clone database to start even though it has the same name as the primary database. Set to a unique value, e.g., = CLONE. Note: Do not change the <code>DB_NAME</code> parameter.
DB_FILE_NAME_CONVERT	Converts datafile filenames. Set to new values if necessary.

Parameter	Purpose
LOG_FILE_NAME_CONVERT	<p>Renames redo logs files. For example, if the datafiles of the primary database reside in the directory <code>/ora/primary</code>, and the clone will reside in <code>/ora/clone</code>, set <code>DB_FILE_NAME_CONVERT</code> to <code>"primary","clone"</code>.</p> <p>Note: You can also rename the redo logs with the <code>ALTER DATABASE RENAME FILE</code> statement. See "Step 5: Prepare the Clone Database" on page B-11.</p>

Step 5: Prepare the Clone Database

Perform the following tasks to prepare the clone database for TSPITR:

1. Restore the auxiliary set and the recovery set to a location different from that of the primary database.

Note: It is possible, although not recommended, to place the recovery set files over their corresponding files on the primary database. For more information see ["Performing Partial TSPITR of Partitioned Tables"](#) on page B-15.

2. Configure your environment so that you can start up the clone database. For example, on UNIX, set `ORACLE_SID` to the name of the clone.
3. Start the clone database without mounting it, specifying the parameter file if necessary:

```
STARTUP NOMOUNT PFILE=/path/initCLONE.ora;
```

4. Mount the clone database:

```
ALTER DATABASE MOUNT CLONE DATABASE;
```

At this point, the database is automatically taken out of ARCHIVELOG mode because it is a clone. All files are offline.

5. If you did *not* set `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT`, rename the files to reflect their new locations:

```
ALTER DATABASE RENAME FILE 'name_of_file_in_primary_location'  
TO 'name_of_corresponding_file_in_clone_location';
```

If you did set `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` but there are files that have been restored to different locations, then rename them.

6. Even if you have set `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT`, do not assume that all the files of the clone database will be in the specified locations—some clone files may have been restored to different locations due to constraints of disk space. Bring online all recovery set and auxiliary set files using the following SQL statement:

```
ALTER DATABASE DATAFILE 'datafile_name' ONLINE;
```

Note: the export phase of TSPITR will not work if all the files of each recovery set tablespace are not online.

Performing TSPITR

This section describes how to execute TSPITR, and includes the following steps:

- [Step 1: Recover the Clone Database](#)
- [Step 2: Open the Clone Database](#)
- [Step 3: Prepare the Clone Database for Export](#)
- [Step 4: Export the Metadata](#)
- [Step 5: Copy the Recovery Set Clone Files to the Primary Database](#)
- [Step 6: Import the Metadata into the Primary Database](#)
- [Step 7: Prepare the Primary Database for Use](#)
- [Step 8: Back Up the Recovered Tablespaces in the Primary Database](#)

Step 1: Recover the Clone Database

Recover the clone database to the desired point by specifying the `USING BACKUP CONTROLFILE` option. Use any form of incomplete recovery as follows:

```
RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL TIME 'YYYY-MM-DD:HH24:MI:SS';  
RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;
```

If the clone database files are not online, Oracle issues an error message.

Step 2: Open the Clone Database

Open the clone database with the RESETLOGS option using the following statement:

```
ALTER DATABASE OPEN RESETLOGS;
```

Because the database is a clone database, only the SYSTEM rollback segment is brought online at this point, which prevents you from executing DML statements against any user tablespace. Any attempt to bring a user rollback segment online will fail and generate an error message.

Step 3: Prepare the Clone Database for Export

Prepare the clone database for export using the TS_PITR_CHECK view and resolving the dependencies just as you did for the primary database (see "[Step 2: Research and Resolve Dependencies on the Primary Database](#)" on page B-7). Only when TS_PITR_CHECK returns no rows will the export phase of TSPITR complete.

Step 4: Export the Metadata

Export the metadata for the recovery set tablespaces using the following statement:

```
exp sys/password point_in_time_recover=y  
recovery_tablespaces=tablespace_1,tablespace_2,tablespace_n
```

If the export phase fails and generates an error message, re-query TS_PITR_CHECK, resolve the problem, and re-run the export. Perform the export phase of TSPITR as the user SYS, otherwise the export will fail.

Shut down the clone database after a successful export:

```
SHUTDOWN IMMEDIATE;
```

Step 5: Copy the Recovery Set Clone Files to the Primary Database

If any recovery set tablespaces are read-only on the primary database, you should take them offline. Use an O/S utility to copy the recovery set files from the clone database to the primary database, taking care not to overwrite any auxiliary set files on the primary database.

Step 6: Import the Metadata into the Primary Database

Import the recovery set metadata into the primary database using the following command:

```
imp sys/password point_in_time_recover=true;
```

This import also updates the copied file's file headers and integrates them with the primary database.

Note: Object name conflicts may arise if objects of the same name exist already in primary database. Resolve these conflicts explicitly.

See Also: For more information about Export, see *Oracle8i Tuning*.

Step 7: Prepare the Primary Database for Use

To prepare the primary database for use, follow these steps:

1. Bring the recovery set tablespaces online in the primary database.
2. Change the recovery set tablespaces to read-write (if they had been altered to read-only, see "[Step 3: Prepare the Primary Database](#)" on page B-9).
3. Undo all the steps taken to resolve dependencies. For example, rebuild indexes or re-enable constraints (see "[Step 2: Research and Resolve Dependencies on the Primary Database](#)" on page B-7).
4. If statistics existed on the recovery set objects before TSPITR was performed, you will need to recalculate them. For partitioned tables, you have to exchange the stand-alone tables into the partitions of their partitioned tables (for more information, see "[Performing Partial TSPITR of Partitioned Tables](#)" on page B-15).

Step 8: Back Up the Recovered Tablespaces in the Primary Database

After TSPITR on a tablespace is complete, use an O/S utility to back up the tablespace.

WARNING: You must back up the tablespace, because otherwise you might lose it. For example, a media failure occurs, but the archived redo logs from the last backup of the database do not logically link to the recovered tablespaces. If you attempt to recover any recovery set tablespaces from a backup taken before TSPITR, you will fail.

Performing Partial TSPITR of Partitioned Tables

This section describes how to perform partial TSPITR of partitioned tables that have a range that has not changed or expanded, and includes the following steps:

- [Step 1: Create a Table on the Primary Database for Each Partition Being Recovered](#)
- [Step 2: Drop the Indexes on the Partition Being Recovered](#)
- [Step 3: Exchange Partitions with Stand-Alone Tables](#)
- [Step 4: Take the Recovery Set Tablespace Offline](#)
- [Step 5: Create Tables at Clone Database](#)
- [Step 6: Drop Indexes on Partitions Being Recovered](#)
- [Step 7: Exchange Partitions with Stand-Alone Tables](#)
- [Step 8: Export the Clone Database](#)
- [Step 9: Copy the Recovery Set Datafiles to the Primary Database](#)
- [Step 10: Import into the Primary Database](#)
- [Step 11: Bring Recovery Set Tablespace Online](#)
- [Step 12: Exchange Partitions with Stand-Alone Tables](#)
- [Step 13: Back Up the Recovered Tablespaces in the Primary Database](#)

Note: Often you will have to recover the dropped partition along with recovering a partition whose range has expanded. See ["Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped"](#) on page B-18.

Step 1: Create a Table on the Primary Database for Each Partition Being Recovered

This table should have the exact same column names and column datatypes as the partitioned table you are recovering. Create the table as follows:

```
CREATE TABLE new_table AS
  SELECT * FROM partitioned_table
  WHERE 1=2;
```

These tables will be used to swap each recovery set partition (see ["Step 3: Exchange Partitions with Stand-Alone Tables"](#) on page B-16).

Step 2: Drop the Indexes on the Partition Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, non-partitioned indexes that exist on the partition you wish to recover. If you drop the indexes on the partition being recovered you will also need to drop them on the clone database (see "[Step 6: Drop Indexes on Partitions Being Recovered](#)" on page B-16). Rebuild the indexes after TSPITR is complete.

Step 3: Exchange Partitions with Stand-Alone Tables

Exchange each partition in the recovery set with its associated stand-alone table (created in Step 1) by issuing the following command:

```
ALTER TABLE table_name EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

Step 4: Take the Recovery Set Tablespace Offline

On the primary database, take each recovery set tablespace offline:

```
ALTER TABLESPACE tablespace_name OFFLINE IMMEDIATE;
```

This prevents any further changes to the recovery set tablespaces on the primary database.

Step 5: Create Tables at Clone Database

After recovering the clone and opening it with the RESETLOGS option, create a table that has the same column names and column data types as the partitioned table you are recovering. Create a table for each partition you wish to recover. These tables will be used later to swap each recovery set partition.

Step 6: Drop Indexes on Partitions Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, non-partitioned indexes that exist on the partition you wish to recover (on the table created in Step 1).

Step 7: Exchange Partitions with Stand-Alone Tables

For each partition in the clone database recovery set, exchange the partitions with the stand-alone tables (created in Step 5) by issuing the following statement:

```
ALTER TABLE partitioned_table_name EXCHANGE PARTITION partition_name  
WITH TABLE table_name;
```

Step 8: Export the Clone Database

Execute export against the clone database for the recovery set tablespaces using the following statement:

```
exp sys/password point_in_time_recover=y  
recovery_tablespaces=tablespace_1,tablespace_2,tablespace_n
```

If the export phase fails (with the error message "ORA 29308 view TS_PITR_CHECK failure"), re-query TS_PITR_CHECK, resolve the problem, and re-run the export. Perform the export phase of TSPITR as the user SYS, otherwise the export will fail (with the error message "ORA-29303: user does not login as SYS"). Shut down the clone database after a successful export.

Step 9: Copy the Recovery Set Datafiles to the Primary Database

If any recovery set tablespaces are READ ONLY on the primary database, change them to OFFLINE. Copy the recovery set datafiles from the clone database to the primary database, taking care not to overwrite auxiliary set files on the primary database.

Step 10: Import into the Primary Database

Import the recovery set metadata into the primary database using the following command:

```
imp sys/password point_in_time_recover=true
```

This import also updates the copied file's file headers and integrates them with the primary database.

Step 11: Bring Recovery Set Tablespace Online

At the primary database, bring each recovery set tablespace online:

```
ALTER TABLESPACE tablespace_name ONLINE;
```

Step 12: Exchange Partitions with Stand-Alone Tables

For each recovered partition on the primary database, swap its associated stand-alone table using the following statement:

```
ALTER TABLE table_name EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

If the associated indexes have been dropped, re-create them.

Step 13: Back Up the Recovered Tablespaces in the Primary Database

Back up the recovered tablespaces on the primary database. Failure to do so will result in loss of data in the event of media failure.

WARNING: You must back up the tablespace, because otherwise you might lose it. For example, a media failure occurs, but the archived redo logs from the last backup of the database do not logically link to the recovered tablespaces. If you attempt to recover any recovery set tablespaces from a backup taken before TSPITR, you will fail.

Performing TSPITR of Partitioned Tables When a Partition Has Been Dropped

This section describes how to perform TSPITR on partitioned tables when a partition has been dropped, and includes the following steps:

- [Step 1: Find the Low and High Range of the Partition that Was Dropped](#)
- [Step 2: Create a Temporary Table](#)
- [Step 3: Delete Records From Partitioned Table](#)
- [Step 4: Take Recovery Set Tablespaces Offline](#)
- [Step 5: Create Tables at Clone Database](#)
- [Step 6: Drop Indexes on Partitions Being Recovered](#)
- [Step 7: Exchange Partitions with Stand-Alone Tables](#)
- [Step 8: Export the Clone Database](#)
- [Step 9: Copy the Recovery Set Datafiles to the Primary Database](#)
- [Step 10: Import into the Primary Database](#)
- [Step 11: Bring Recovery Set Tablespace Online](#)
- [Step 12: Insert Stand-Alone Tables into Partitioned Tables](#)
- [Step 13: Back Up the Recovered Tablespaces in the Primary Database](#)

Step 1: Find the Low and High Range of the Partition that Was Dropped

When a partition is dropped, the range of the partition above it expands downwards. Therefore, there may be records in the partition above that should actually be in the dropped partition after it has been recovered. To ascertain this, issue the following command at the primary database:

```
SELECT * FROM partitioned_table
WHERE relevant_key
BETWEEN low_range_of_partition_that_was_dropped
AND high_range_of_partition_that_was_dropped;
```

Step 2: Create a Temporary Table

If any records are returned, create a temporary table in which to store these records so that if necessary they can be inserted into the recovered partition later.

Step 3: Delete Records From Partitioned Table

Delete all the records stored in the temporary table from the partitioned table.

Step 4: Take Recovery Set Tablespaces Offline

At the primary database, take each recovery set tablespace offline:

```
ALTER TABLESPACE tablespace_name OFFLINE IMMEDIATE;
```

Step 5: Create Tables at Clone Database

After opening the clone with the RESETLOGS option, create a table that has the exact same column names and column datatypes as the partitioned table you are recovering. Create a table for each partition you wish to recover. These tables will be used later to swap each recovery set partition.

Step 6: Drop Indexes on Partitions Being Recovered

Drop the indexes on the partition you wish to recover, or create identical, non-partitioned indexes that exist on the partition you wish to recover.

Step 7: Exchange Partitions with Stand-Alone Tables

For each partition in the clone recovery set, exchange the partitions into the stand-alone tables created in Step 5 by issuing the following statement:

```
ALTER TABLE partitioned_table_name EXCHANGE PARTITION partition_name  
WITH TABLE table_name;
```

Step 8: Export the Clone Database

Execute export against the clone database for the recovery set tablespaces using the following statement:

```
exp sys/password point_in_time_recover=y  
recovery_tablespaces=tablespace_1,tablespace_2,tablespace_n
```

If the export phase fails (with the error message "ORA 29308 view TS_PITR_CHECK failure"), re-query TS_PITR_CHECK, resolve the problem, and re-run the export. You need to perform the export phase of TSPITR as the user SYS, otherwise the export will fail (with the error message "ORA-29303: user does not login as SYS"). Shut down the clone database after a successful export.

Step 9: Copy the Recovery Set Datafiles to the Primary Database

If any recovery set tablespaces are READ ONLY on the primary database, you should change them to OFFLINE. Copy the recovery set datafiles from the clone database to the primary database, taking care not to overwrite any auxiliary set files on the primary database.

Step 10: Import into the Primary Database

Import the recovery set meta-data into the primary database using the following command:

```
imp sys/password point_in_time_recover=true;
```

This import also updates the copied file's file headers and integrates them with the primary database.

Step 11: Bring Recovery Set Tablespace Online

Online each recovery set tablespace at the primary database by issuing the following statement:

```
ALTER TABLESPACE tablespace_name ONLINE;
```

Step 12: Insert Stand-Alone Tables into Partitioned Tables

At this point you must insert the stand-alone tables into the partitioned tables; you can do this by first issuing the following statement:

```
ALTER TABLE table_name SPLIT PARTITION partition_name AT (key_value) INTO  
(PARTITION partition_1_name TABLESPACE tablespace_name,  
PARTITION partition_2_name TABLESPACE tablespace_name);
```

Note that at this point, partition 2 is empty because keys in that range have already been deleted from the table.

Issue the following statement to swap the stand-alone table into the partition:

```
ALTER TABLE EXCHANGE PARTITION partition_name WITH TABLE table_name;
```

Now insert the records saved in Step 2 into the recovered partition (if desired).

Note: If the partition that has been dropped is the last partition in the table, add it using the ALTER TABLE ADD PARTITION statement.

Step 13: Back Up the Recovered Tablespaces in the Primary Database

Back up the recovered tablespaces in the primary database. Failure to do so will result in loss of data in the event of media failure.

WARNING: You must back up the tablespace, because otherwise you might lose it. For example, a media failure occurs, but the archived redo logs from the last backup of the database do not logically link to the recovered tablespaces. If you attempt to recover any recovery set tablespaces from a backup taken before TSPITR, you will fail.

Note: As described in "TSPITR Limitations" on page B-5, TSPITR cannot be used to recover a tablespace that has been dropped. Therefore, if the associated tablespace of the partition has been dropped as well as the partition, you cannot recover that partition using TSPITR. You will have to perform ordinary export/import recovery. Specifically, you will have to:

- Make a copy of the database
- Roll it forward
- Open the database
- Exchange the partition for a stand-alone table
- Make a table-level export of the stand-alone table

Import the table into the primary database and insert it into the partitioned table using the ALTER TABLE SPLIT PARTITION or ALTER TABLE ADD PARTITION statements.

Performing TSPITR of Partitioned Tables When a Partition Has Split

This section describes how to recover partitioned tables when a partition has been split, and includes the following sections:

- [Step 1: Drop the Lower of the Two Partitions at the Primary Database](#)
- [Steps 2-13: Follow Same Steps as for Partial TSPITR of Partitioned Tablespaces](#)

Step 1: Drop the Lower of the Two Partitions at the Primary Database

For each partition you wish to recover whose range has been split, drop the lower of the two partitions so that the higher expands downwards. In other words, the higher partition has the same range as before the split. For example, if P1 was split into partitions P1A and P1B, then P1B must be dropped, meaning that partition P1A now has the same range as P1.

For each partition that you wish to recover whose range has split, create a table that has exactly the same column names and column datatypes as the partitioned table you are recovering:

```
CREATE TABLE new_table
AS SELECT * FROM partitioned_table
WHERE 1=2;
```

These tables will be used to exchange each recovery set partition in Step 3.

Steps 2-13: Follow Same Steps as for Partial TSPITR of Partitioned Tablespaces

Follow steps 2-13 in the procedure for ["Performing Partial TSPITR of Partitioned Tables"](#) on page B-15.

TSPITR Tuning Considerations

This section describes tuning issues relevant to TSPITR, and includes the following topics:

- [Recovery Set Location Considerations](#)
- [Backup Control File Considerations](#)

Recovery Set Location Considerations

If space is at a premium, it is possible to recover the recovery set files "in place." In other words, recover them over their corresponding files on the primary database. Note that the recommended practice is to restore the files to a separate location and then copy across before the import phase of TSPITR is complete (see ["Step 6: Import the Metadata into the Primary Database"](#) on page B-13).

Advantages and Disadvantages of Recovering to a Separate Location

The advantages of recovering to a separate location are:

- Greater availability and flexibility. If recovery is abandoned at a point before integrating the recovery set with the primary database, then there is no need to restore the recovery set files on the primary database and recover them using normal means.
- The recovery set tablespaces can be accessible on the primary database while recovery occurs on the clone. For example, there may be a subset of undamaged data within the recovery set tablespaces that you wish to access (see ["Step 3: Prepare the Primary Database"](#) on page B-9). If so, change the recovery set tablespaces to READ ONLY on the primary database. If the files are recovered in place, this option is not available.

The disadvantage of recovering to a separate location is that more space is required for the clone database.

Advantages and Disadvantages of Recovering in Place

An advantage of recovering in place is that the amount of space taken up by the recovery set files is saved. After recovery of the clone is complete, there is no need to copy the recovery set files over to the primary database.

The disadvantage is that if the recovery is abandoned at a point before integrating the recovery set with the primary database (see "[Step 6: Import the Metadata into the Primary Database](#)" on page B-13), then you must restore the overwritten recovery set files of the primary database from a backup and recover by normal means, prolonging data unavailability. You cannot query any undamaged data within the recovery set tablespaces during recovery.

Backup Control File Considerations

The error "ORA-01152 file 1 was not restored from a sufficiently old backup" will be encountered in the situation where no recovery is performed on the clone before grafting it to the primary. For example, if a backup is taken at time A, and a database at time B requires TSPITR to be done on a particular tablespace to take that tablespace to time A, what actually happens is that the clone database is opened RESETLOGS without any recovery having been done. When recovering the clone, the SQL*Plus commands would be:

```
RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;  
CANCEL;  
OPEN DATABASE RESETLOGS;
```

At this point no redo logs have been applied, but we wish to open the database. However, since we save checkpoints to the control file in Oracle 8, it is a requirement for clone and standby databases that the backup control files need to be taken at a point after the rest of the backup was taken. Unless this is the case, "ORA-01152 file 1 was not restored from a sufficiently old backup" will be encountered on open, not because file 1 is too recent (because it is in sync with the rest of the database), but because it is more recent than the control file.

Note: A RESETLOGS would work with a regular database if a clean, consistent backup and an old backup control file is used. Otherwise, the behavior would not be compatible with existing backup scripts.

Performing TSPITR Using Transportable Tablespaces

You can use the *transportable tablespace* feature to perform tablespace point-in-time recovery. This method is similar to the O/S TSPITR described in previous sections, except you use the transportable tablespace feature to move recovered tablespaces from the clone database to the primary database. To learn how to transport tablespaces between databases, see the *Oracle8i Administrator's Guide*.

The major difference between O/S TSPITR and TSPITR via transportable tablespaces is that for the former you must follow the special procedures for creating clone `init.ora` files, mounting the clone database, etc. O/S TSPITR assumes that the user may place the clone database on the same computer as the primary database; the special clone database commands provide error checks to prevent the corruption of the primary database on the same computer while recovering the clone database.

Performing TSPITR via transportable tablespaces relaxes this requirement. If you restore backups to a different computer separate from the primary database, you can start the clone database as if it were the primary database, using the normal database MOUNT command instead of the clone database MOUNT command. If you restore backups on the same computer as the primary database, however, follow the special procedure to create the clone database as described in O/S TSPITR, since this procedure helps prevent accidental corruption of the primary database while recovering the clone database on the same computer.

TSPITR via transportable tablespaces provides basically the same functionality as O/S TSPITR, but is more flexible since:

- You can recover dropped tablespaces, which you cannot do using O/S TSPITR.
- You can transport the recovered tablespaces from the clone database to a different database to test the procedure before making permanent changes to the primary database.

To perform TSPITR using the transportable tablespace feature:

1. Restore the backup to construct the clone database. Create the clone database on the same computer as the primary database or on a different computer.

2. If you create the clone database using the special clone database procedure, place all recovery set and auxiliary set files online:

```
ALTER DATABASE DATAFILE 'datafile_name' ONLINE;
```

If you create the clone database as a normal database (on a computer different from the primary database), take all datafiles not in the recovery and auxiliary set offline:

```
ALTER DATABASE DATAFILE 'datafile_name' OFFLINE;
```

3. Recover the clone database to the specified point in time.
4. Open the clone database with the RESETLOGS option.
5. Make the tablespaces in the recovery set read-only by issuing the ALTER TABLESPACE READ ONLY command.
6. Generate the transportable set by running EXPORT. Include all tablespaces in the recovery set.
7. In the primary database, drop the tablespaces in the recovery set via the DROP TABLESPACE command.
8. Plug in the transportable set into the primary database by running IMPORT.
9. If necessary, make the recovered tablespaces read write by issuing the ALTER TABLESPACE READ WRITE command.

Glossary

advancing the checkpoint

The action that occurs when the redo log entry marking the checkpoint changes. For example, the CKPT process logs redo record 356 as the checkpoint, then three seconds later records redo record 358 as the checkpoint.

See Also: [checkpoint](#), [redo record](#)

archived redo log

A copy of one of the filled members of an online redo log group made when the database is in ARCHIVELOG mode. As the LGWR process fills each online redo log with redo records, Oracle copies the log to one or more offline archive log destinations. This copy is the archived redo log, also known as the *offline redo log*.

ARCHIVELOG mode

The mode of the database in which Oracle copies filled online redo logs to disk. Specify the mode at database creation or by using the ALTER DATABASE command. You can enable automatic archiving either dynamically using the ALTER SYSTEM command or by setting the initialization parameter LOG_ARCHIVE_START to TRUE.

Running your database in ARCHIVELOG mode has several advantages over NOARCHIVELOG mode. You can:

- Back up your database while it is open and being accessed by users.
- Recover your database to any desired point in time.

To protect your ARCHIVELOG mode database in case of failure, back up your archived logs.

See Also: [archived redo log](#), [NOARCHIVELOG mode](#)

archiving

The operation in which the ARC*n* background process copies filled online redo logs to offline destinations. You must run the database in ARCHIVELOG mode to archive redo logs.

ATL (automated tape library)

A unit that contains one or more tape drives, a robotic arm, and a shelf of tapes. The ATL, also called a *tape silo*, is able to load and unload tapes into the tape drive from the shelf without operator intervention. More sophisticated tape libraries are able to identify each tape; for example, the robotic arm can use a bar-code reader to scan each tape's barcode and identify it.

See Also: [media manager](#)

auxiliary database

(1) A database created from target database backups using the RMAN **duplicate** command.

(2) A temporary database that is restored to a new location and then started up with a new instance name during tablespace point-in-time recovery (TSPITR). A TSPITR auxiliary database contains the recovery set and auxiliary set.

See Also: [TSPITR](#), [recovery set](#), [auxiliary set](#)

auxiliary set

In TSPITR, the set of files that is not in the recovery set but which must be restored in the clone database for the TSPITR set to be successful. These auxiliary files include:

- Backup control file
- SYSTEM tablespace
- Any datafiles containing rollback segments
- Temporary tablespace (optional)

See also: [auxiliary database](#), [recovery set](#), [TSPITR](#)

backup

(1) A copy of data, i.e., a database, tablespace, table, datafile, control file, or archived redo log. You can make a backup by:

- Making a copy of one or more tables via the Export utility.

- Using Recovery Manager to back up one or more datafiles, control files, or archived redo logs.
- Making a copy either to disk or to tape using operating system utilities (such as **cp**, **tar**, **dd**).

(2) An RMAN command that creates a backup set. The output of a backup command is only usable by RMAN; the output of the RMAN **copy** command can be used without additional processing.

See Also: [copy](#), [backup set](#), [multiplexing](#), [RMAN](#)

backup, closed

See [closed backup](#)

backup, whole database

See [whole database backup](#)

backup control file

A backup of the control file. Make the backup by:

- Using the Recovery Manager **backup** or **copy** command. Never create a backup control file by using O/S commands.
- Using the SQL command ALTER DATABASE BACKUP CONTROLFILE TO *'filename'*.

Typically, you restore backup control files when all copies of the current control file are damaged; sometimes you restore them before performing certain types of point-in-time recovery.

See Also: [control file](#)

backup piece

A backup piece is a physical file in an RMAN-specific format that belongs to only one backup set. A backup set usually contains only one backup piece. The only time RMAN creates more than one backup piece is when you limit the piece size a **set limit kbytes** command. Use this command when the storage or media manager you are writing your backup to is not able to support writing a file larger than a certain size.

See Also: [backup](#), [backup set](#), [RMAN](#)

backup set

An RMAN-specific logical grouping of one or more physical files called *backup pieces*. The output of the RMAN **backup** command is a backup set. Extract the files in a backup set by using the RMAN **restore** command. You can multiplex files into a backup set, i.e., intermingle blocks from input files into a single backup set.

There are two types of backup sets:

- Datafile backup sets, which are backups of any datafiles or a control file. This type of backup set is *compressed*, which means that it only contains datafile blocks that have been used; unused blocks are omitted.
- Archivelog backup sets, which are backups of archived redo logs.

See Also: [backup piece](#), [compression](#), [multiplexing](#), [RMAN](#)

breaking a mirror

The termination of a disk mirroring procedure so that a mirror image is no longer kept up-to-date. You can create operating system database backups by placing the tablespaces in the database in hot backup mode and then breaking the mirror. After taking the tablespaces out of hot backup mode, back up the broken mirror side to tape. After the backup is complete, you can *resilver* the mirror.

See Also: [hot backup mode](#), [mirroring](#), [resilvering a mirror](#)

buffer cache

The portion of the SGA that holds copies of data blocks read from datafiles. All user processes concurrently connected to the instance share access to the database buffer cache.

The buffers in the cache are organized in two lists: the dirty list and the least recently used (LRU) list. The dirty list holds dirty buffers, which contain data that has been modified but has not yet been written to disk. The least recently used (LRU) list holds free buffers (unmodified and available), pinned buffers (currently being accessed), and dirty buffers that have not yet been moved to the dirty list.

See Also: [SGA \(System Global Area\)](#)

cancel-based recovery

A type of incomplete media recovery in which you use the RECOVER command with the UNTIL CANCEL clause. Recovery proceeds until you issue the CANCEL command.

See Also: [incomplete recovery](#), [media recovery](#)

change vector

A single change to a single data block. A change vector is the smallest unit of change recorded in the redo log.

See Also: [redo record](#)

change-based recovery

A type of incomplete media recovery that recovers up to a specified SCN. You can also perform cancel-based recovery, which recovers until you issue the CANCEL command, and time-based recovery, which recovers to a specified time.

See Also: [cancel-based recovery](#), [incomplete recovery](#), [media recovery](#), [system change number \(SCN\)](#), [time-based recovery](#)

channel

A connection between Recovery Manager and the target database. Each allocated channel starts a new Oracle server session; the session then performs backup, restore, and recovery operations. The type of channel determines whether the Oracle server process will attempt to read or write and whether it will work through a third-party media manager. If the channel is of type:

- **disk**, the server process attempts to read backups from or write backups to disk.
- **'sbt_tape'**, the server process attempts to read backups from or write backups to a third-party media manager.

Channels are always able to read and write datafiles to and from disk, no matter what their type.

See Also: [media manager](#), [target database](#)

checkpoint

A pointer indicating that all changes prior to the SCN specified by a redo record have been written to the datafiles by DBWn. Each redo record in the redo log describes a change or a set of atomic changes to database blocks; a checkpoint for a redo entry confirms that the changes described in previous redo entries have been written to disk, not just to memory buffers. The background process CKPT automatically records a checkpoint in the control file every three seconds.

See Also: [control file](#), [redo record](#)

checksum

A numeric value that is mathematically derived from the contents of an Oracle data block. The checksum allows Oracle to validate the consistency of the block.

See Also: [data block](#)

clean shutdown

A database shut down with the IMMEDIATE, TRANSACTIONAL, or NORMAL options of the SHUTDOWN command. A database shut down cleanly does not require recovery; it is already in a consistent state.

closed backup

A backup of one or more database files taken while the database is closed. Typically, closed backups are also whole database backups. If you closed the database cleanly, then all the files in the backup are consistent. If you shut down the database using a SHUTDOWN ABORT or the instance terminated abnormally, then the backups are inconsistent.

See Also: [clean shutdown](#), [consistent backup](#)

closed database

A database that is not available to users for queries and updates. When the database is closed you can start the instance and optionally mount the database.

See Also: [open database](#)

cold backup

See [closed backup](#)

command file

A file containing a sequence of RMAN commands that you can run from the command line. The contents of the command file should be identical to commands entered at the command line.

complete recovery

The recovery of a database by applying all online and archived redo generated since the restored backup. Typically, you perform complete media recovery when media failure damages one or more datafiles or control files. You fully recover the damaged files using all redo generated since the restored backup was taken. If you use RMAN, you can also apply incremental backups during complete recovery.

See Also: [incomplete recovery](#), [media recovery](#)

compression

The process of copying only used data blocks into RMAN backup sets. A newly created datafile contains many never-used blocks. When RMAN creates backup sets, it only includes blocks that have been used; it follows that RMAN does not write never-used blocks into backup sets.

consistent backup

A whole database backup that you can open with the RESETLOGS option without performing media recovery. In other words, you do not need to apply redo to any datafiles in this backup for it to be consistent. All datafiles in a consistent backup must:

- Have the same checkpoint SCN in their headers, unless they are datafiles in tablespaces that are read-only or offline normal (in which case they will have a clean SCN that is earlier than the checkpoint SCN).
- Contain no changes past the checkpoint SCN, i.e., are not fuzzy.
- Match the datafile checkpoint information stored in the control file.

You can only take consistent backups after a database has been shut down cleanly. The database must not be opened until the backup has completed.

See Also: [clean shutdown](#), [fuzzy file](#), [inconsistent backup](#), [system change number \(SCN\)](#), [whole database backup](#)

control file

A binary file associated with a database that maintains the physical structure and timestamps of all files in that database. Oracle updates the control file continuously during database use and must have it available for writing whenever the database is mounted or open.

See Also: [backup control file](#), [current control file](#)

copy

(1) To replicate data. You make copies of Oracle datafiles, control files, and archived redo logs in two ways:

- Using O/S utilities (for example, the UNIX **cp** or **dd**).
- Using the Recovery Manager **copy** command.

(2) A Recovery Manager command that makes a replica of a database's datafiles, control file, or archived redo logs. This replica is made by an Oracle server process, allocated to a Recovery Manager channel, which reads the Oracle file and writes a

replica out to disk. Recovery Manager can copy the files of an open database without putting the tablespaces into hot backup mode.

See Also: [backup](#), [hot backup mode](#)

current datafile

In RMAN, the datafile in the target database pointed to by the control file. You can make a backup datafile current again by executing a **switch** command.

corrupt block

An Oracle block that is not in a recognized Oracle format, or whose contents are not internally consistent. Oracle identifies corrupt blocks as one of two types:

- Logically corrupted, i.e., the block was corrupted by the application of redo.
- Media corrupted, i.e., the block format is not correct. The block may have:
 - An impossible format
 - A wrong data block address
 - An impossible block type

You can only repair a media corrupted block by:

- Replacing the block and initiating recovery. Replace the block by restoring the datafile or applying an incremental backup.
- Renewing the block. Renew a block by dropping the table (or other database object) that contains the corrupt block so that its blocks are reused for another object

If media corruption is due to faulty hardware, neither solution will work until the hardware fault is corrected.

See Also: [data block](#), [fractured block](#)

corrupt datafile

A datafile that contains one or more corrupt blocks.

See Also: [corrupt block](#)

crash recovery

The automatic application of online redo records to a database after either a single-instance database crashes or all instances of an OPS database crash. Crash recovery only requires redo from the online logs: archived redo logs are not required.

In crash recovery, an instance automatically recovers the database before opening it. In general, the first instance to open the database after a crash or SHUTDOWN ABORT automatically performs crash recovery.

See Also: [recovery](#), [redo record](#)

crosscheck

A check to determine whether files on disk or in the media management catalog correspond to the information in the recovery catalog (if used) and the control file. Because the media manager can mark tapes as expired or unusable, and because files can be deleted from disk or otherwise become corrupted, the recovery catalog and control file can contain outdated information about backups and image copies.

Use **change ... crosscheck** when you want to provide a list of backup sets or pieces to check; use **crosscheck backupset** when you wish to restrict the crosscheck to a specified device type, object type, or date range and let RMAN generate the list of backup sets or pieces. To determine whether you can restore a file, use **validate** or **restore ... validate**.

See Also: [media manager](#), [recovery catalog](#), [validation](#)

cumulative backup

An incremental backup that backs up all the blocks changed since the most recent backup at level $n-1$ or lower. For example, in a cumulative level 2 backup, RMAN determines which level 1 or level 0 backup is most recent and then backs up all blocks changed since that backup.

See Also: [data block](#), [differential backup](#), [incremental backup](#), [multi-level incremental backups](#)

current control file

The control file on disk; it is the most recently modified control file for the current incarnation of the database. For a control file to be considered current during recovery, it must not have been restored from backup.

See Also: [control file](#)

current online redo log

The online redo log file in which the LGWR background process is currently logging redo records. Those files to which LGWR is not writing are called inactive.

Every database must contain at least two online redo log files. If you are *multiplexing* your online redo log, LGWR concurrently writes the same redo data to multiple files. The individual files are called *members* of an online redo log *group*.

See Also: [online redo log](#), [redo log](#), [redo log buffer](#), [redo log groups](#)

data block

The smallest unit of data in an Oracle database, the size of which is determined by the parameter `DB_BLOCK_SIZE` at database creation.

See Also: [corrupt block](#)

database point-in-time recovery (DBPITR)

The recovery of a database to a specified non-current time, SCN, or log sequence number.

See Also: [incomplete recovery](#), [tablespace point-in-time recovery \(TSPITR\)](#)

datafile

A datafile is a physical O/S file on disk that was created by Oracle and contains data structures such as tables and indexes. A datafile can only belong to one database.

See Also: [inaccessible datafile](#)

datafile copy

A copy of a datafile on disk produced by either:

- The Recovery Manager **copy** command.
- An O/S utility.

See Also: [backup](#), [copy](#)

datafile header

See [file header](#)

db identifier

An internal, uniquely generated number that differentiates databases. Oracle creates this number automatically when you create the database.

differential backup

A type of incremental backup that backs up all blocks that have changed since the most recent backup at level *n* or lower. For example, in a differential level 2 backup RMAN determines which level 2, level 1, or level 0 backup is most recent and then backs up all blocks changed since that backup. Differential backups, also called *non-cumulative incremental backups*, are the default type of incremental backup.

See Also: [cumulative backup](#), [incremental backup](#), [multi-level incremental backups](#)

duplicate database

A database created from target database backups using the RMAN duplicate command.

See Also: [auxiliary database](#)

export

The extraction of logical data (i.e., not physical files) from a database using the Export utility. You can then use the Import utility to import the data into a database.

See Also: [full export](#)

file header

The first block of an Oracle datafile. The file header contains bookkeeping information related to the file, including the checkpoint SCN. Oracle requires media recovery when the checkpoint SCN in the datafile header does not match the file header information stored in the control file.

See Also: [checkpoint](#)

fractured block

A type of media corruption that can occur when DBWn is writing a block at the same time an O/S utility is reading the block for backup. The block that the O/S reads can be *split*, i.e., the top of the block is written at one point in time while the bottom of the block is written at another point in time. If you restore a file containing a fractured block and Oracle reads the block, then the block is considered *corrupt*.

The potential for fractured blocks necessitates putting tablespaces in hot backup mode before O/S online backups. A database in hot backup mode writes whole Oracle data blocks to the redo log, so that if a block is split during the backup, you can repair it by using redo. Recovery Manager does not experience this problem because the server process performing the backup or copy reads each block to determine whether it is split and re-reads the block until it gets a consistent version.

See Also: [corrupt block](#), [corrupt datafile](#), [hot backup mode](#)

full backup

A non-incremental RMAN backup. Note that "full" does not refer to how much of the database is backed up, but to the fact that the backup is not incremental. Consequently, you can make a full backup of one datafile.

The only difference between a full backup and an incremental level 0 backup is that the full backup will not affect the number of blocks backed up by any subsequent incremental backup.

See Also: [incremental backup](#)

full export

An export of the whole database.

See Also: [export](#)

full resynchronization

A Recovery Manager operation that updates the recovery catalog with all changed information in the database's control file. You can initiate full catalog resynchronizations by issuing the RMAN command **resync catalog**. Recovery Manager initiates resync operations as needed when executing certain commands.

See Also: [control file](#), [recovery catalog](#), [resynchronization](#)

fuzzy file

A datafile that contains at least one block with an SCN more recent than the checkpoint SCN in its header. For example, this situation occurs when Oracle updates a datafile that is in hot backup mode. A fuzzy file that is restored always requires recovery.

See Also: [checkpoint](#), [hot backup mode](#)

hot backup

See [open backup](#)

hot backup mode

The database mode initiated when you issue the ALTER TABLESPACE *tablespace_name* BEGIN BACKUP command before taking an open backup. You take a tablespace out of hot backup mode when you issue the ALTER TABLESPACE *tablespace_name* END BACKUP command.

You must use this command when you make an O/S backup of one or more datafiles in an online tablespace. Recovery Manager does not require you to put the database in hot backup mode. Updates to tablespaces in hot backup mode create more than the usual amount of redo because each change causes Oracle to write the entire block rather than just the changed data to the redo log.

See Also: [corrupt block](#), [fractured block](#), [open backup](#)

image copy

A copy of a single datafile, archived redo log file, or control file that is:

- Usable as-is to perform recovery (unlike a backup set, which is in an RMAN-specific format).
- Generated using the RMAN **copy** command or an O/S command such as the UNIX **dd**.

See Also: [copy](#)

inaccessible datafile

A datafile that Oracle is attempting to read, but cannot find. Attempts to access an inaccessible file result in errors. Typically, a file is inaccessible because the media on which it is stored is faulty or the file has been moved or deleted.

See Also: [datafile](#), [media failure](#)

inactive redo log

A redo log file that is not required for instance recovery because the changes contained in its redo records have already been applied to the database. The current redo log file is never inactive. If you operate your database in ARCHIVELOG mode, the ARCn process archives inactive redo log files.

See Also: [current online redo log](#), [online redo log](#), [redo log](#), [redo log buffer](#), [redo log groups](#)

incarnation

A separate version of a physical database. The incarnation of the database changes when you open it with the RESETLOGS option. Make a whole database backup of all files that are not offline-clean or read-only after opening with the RESETLOGS option. If using RMAN, issue the **reset database** command after opening in RESETLOGS mode.

incomplete recovery

The recovery of a database in which you do not apply all of the changes generated since you created the restored backup.

Incomplete recovery is usually performed when:

- The online logs are lost due to hardware failure. In this case, you recover the database until the last archived log generated before the failure.
- A user error necessitates recovery up until just before the error occurred.

The requirement is to recover up until some point in time before an incorrect action occurred in the database. For example, a user mistakenly deletes payroll transactions before the transactions are sent to the payroll agency. In this example, the DBA will need to restore the whole database and then perform incomplete recovery up until the point just before the user deleted the transactions.

- An archived redo log required for recovery is missing

An archived redo log which is needed for complete recovery was not backed up, or the archived redo log contents are corrupt. In this case, your only option is to recover up to the missing log.

In each case, open the database with the RESETLOGS option after performing media recovery. If you use RMAN with a recovery catalog, you must also reset the database.

See Also: [complete recovery](#), [media recovery](#), [recovery](#), [redo record](#)

inconsistent backup

A backup in which some of the files in the backup contain changes that were made after the files were checkpointed. This type of backup needs recovery before it can be made consistent. Inconsistent backups are usually created by taking open database backups; that is, the database is open while the files are being backed up. You can also make an inconsistent backup by backing up datafiles while a database is closed, either:

- Immediately after an Oracle instance crashed (or all instances in an Oracle Parallel Server cluster).
- After shutting down the database using SHUTDOWN ABORT.

Note that inconsistent backups are only useful if the database is in ARCHIVELOG mode.

See Also: [consistent backup](#), [open backup](#), [system change number \(SCN\)](#), [whole database backup](#)

incremental backup

An RMAN backup in which only modified blocks are backed up. Incremental backups are classified by *level*. An incremental level 0 backup performs the same function as a full backup in that they both back up all blocks that have ever been used. The difference is that a full backup will not affect blocks backed up by subsequent incremental backups, whereas an incremental backup will affect blocks backed up by subsequent incremental backups.

Incremental backups at levels greater than 0 back up only blocks that have changed since previous incremental backups. Blocks that have not changed are not backed up.

Incremental backups are divided into two types: *differential* and *cumulative*. Differentials back up all blocks that have changed since the most recent backup at level n or lower. For example, a differential level 2 backup backs up all blocks modified since a previous level 2, level 1, or level 0 backup, whichever is most recent. Cumulative backups back up all the blocks used since the most recent backup at level $n-1$ or lower. For example, a cumulative level 2 backup backs up all blocks modified since a previous level 1 or level 0 backup, whichever is most recent.

See Also: [cumulative backup](#), [incremental backup](#)

instance

An SGA, Oracle code, and background processes. Create an instance by issuing any of the following commands:

- **STARTUP NOMOUNT**—the instance starts, but does not mount the control files or open the database.
- **STARTUP MOUNT**—The instance starts, then mounts the database's control files. It does not open the database.
- **STARTUP**—The instance starts, mounts the database's control files, and opens the database.

An instance is stopped by issuing a SHUTDOWN statement.

See Also: [SGA \(System Global Area\)](#)

instance recovery

In an OPS configuration, the application of redo data to an open database by an instance when this instance discovers that another instance has crashed. A surviving instance automatically uses the redo log to recover the data in the instance's buffer cache. Oracle undoes any uncommitted transactions that were in progress on the failed instance when it crashed and then clears any locks held by the crashed instance after recovery is complete.

See Also: [recovery](#), [redo record](#)

job commands

RMAN commands such as **backup**, **copy**, and **recover** that you must execute within the brackets of a **run** command.

See Also: [stand-alone commands](#)

LogMiner

A utility that allows you read information contained in online or archived redo logs based on various selection criteria. For example, you can select information from the VSLOGMINER_CONTENTS view that enables you to:

- Track changes to a specific table.
- Track changes made by a specific user.
- Map data access patterns.
- View the SQL syntax for undoing or redoing a specific change made against the database.
- Use archived data for tuning and capacity planning.

See Also: [archived redo log](#)

log sequence number

A number that uniquely identifies a set of redo records in a redo log file. When Oracle fills one online redo log file and switches to a different one, Oracle automatically assigns the new file a log sequence number. For example, if you create a database with two online log files, then the first file is assigned log sequence number 1. When the first file fills and Oracle switches to the second file, it assigns log sequence number 2; when it switches back to the first file, it assigns log sequence number 3, and so forth.

See Also: [log switch](#), [redo log](#)

log sequence recovery

For RMAN, a type of incomplete recovery that recovers up to a specified log sequence number.

See Also: [incomplete recovery](#)

log switch

The point at which LGWR stops writing to the active redo log file and switches to the next available redo log file. LGWR switches when either the active log file is filled with redo records or you force a switch manually.

If you run your database in ARCHIVELOG mode, Oracle archives the redo data in inactive log files into archived redo logs. When a log switch occurs and LGWR begins overwriting the old redo data, you are protected against data loss because

the archived redo log contains the old data. If you run in NOARCHIVELOG mode, Oracle overwrites old redo data at a log switch without archiving it. Hence, you lose all old redo data.

See Also: [redo log](#)

logical backups

Backups in which the Export utility uses SQL to read database data and then export it into a binary file at the O/S level. You can then import the data back into a database using the Import utility.

Backups taken with the Export utility differ in the following ways from RMAN backups:

- Database logical objects are exported independently of the files that contain those objects.
- Logical backups can be imported into a different database, even on a different platform. RMAN backups are not portable between databases or platforms.

See Also: [physical backups](#)

managed recovery mode

A mode of a standby database in which the standby waits for archived log files from a target database and then automatically applies the redo logs once the files become available. This feature eliminates the need for you to interactively provide the recovery process with filenames of the archived redo logs.

See Also: [standby database](#)

media failure

A physical problem that arises when Oracle fails in its attempt to write or read a file that is required to operate the database. A common example is a disk head crash that causes the loss of all data on a disk drive. Disk failure can affect a variety of files, including the datafiles, redo log files, and control files. Because the database instance cannot continue to function properly, it cannot write the data in the buffer cache of the SGA to the datafiles.

See Also: [buffer cache](#), [media recovery](#)

media manager

A utility provided by a third party vendor that is capable of actions such as loading, labelling and unloading sequential media such as tape drives. Media managers also

allow you to configure media expiration and recycling, and may also have the ability to control Automated Tape Libraries (ATLs).

See Also: [ATL \(automated tape library\)](#)

media management interface

An Oracle published API to which media management vendors have written compatible software libraries. This software integrates with Oracle so that an Oracle server process is able to issue commands to the Media Manager to write backup files to sequential storage, and read files from sequential storage. When Oracle issues a request to backup or restore a file, the media manager handles the actions required to load, label, and unload the correct tape.

The media management interface is also called the *media management layer*, the *media management library (MML)*, and the *SBT interface*.

See Also: [media manager](#)

media recovery

The application of online or archived redo records to a restored backup to bring it current to a specified time. When performing media recovery, you can recover:

- The whole database
- A tablespace
- A datafile

If you use all redo data, you perform complete recovery; if you use only part of the redo data, you perform incomplete recovery. Typically, you perform media recovery after a media failure.

In ARCHIVELOG mode, you have the choice of complete or incomplete recovery. In NOARCHIVELOG mode, your only option is typically to restore from the most recent backup without applying redo data.

In exceptional circumstances, you can recover a datafile or database if the database is not in ARCHIVELOG mode, but only if none of the online logs has been overwritten since the backup.

See Also: [complete recovery](#), [incomplete recovery](#), [media failure](#), [recovery](#), [redo record](#)

mirroring

Using the O/S to maintain an identical copy of Oracle data. Typically, mirroring is performed on duplicate hard disks at the operating system level, so that if one of

the disks becomes unavailable, the other disk can continue to service requests without interruptions. For example, you can mirror a datafile so that Oracle writes the same information to two different disk drives. You can then *break* the mirror to create a backup and later *resilver* the mirror.

When you mirror files, Oracle writes once while the operating system writes to multiple disks; when you *multiplex* files, Oracle writes the same data into multiple files.

See Also: [breaking a mirror](#), [resilvering a mirror](#)

mounted database

An instance that is started and has the control files associated with the database open. You can mount a database without opening it; typically, you put the database in this state for maintenance or for restore and recovery operations.

See Also: [instance](#)

multi-level incremental backups

RMAN-generated incremental backups that allow you to conserve space by planning which blocks you want to back up and when. A level 0 incremental backup, which is the base for subsequent incremental backups, copies all blocks containing data. When you generate a level n incremental backup in which n is greater than 0, you back up either:

- All blocks that have changed since the most recent backup at level n or lower. This is the default type of incremental backup, called a *differential backup*.
- All blocks used since the most recent backup at level $n-1$ or lower. This type of backup is called a *cumulative backup*.

You can create a backup strategy in which you generate a backup at a different level each day, thereby controlling how much data you back up.

See Also: [cumulative backup](#), [differential backup](#), [incremental backup](#)

multiplexing

■ **online redo logs**

The automated maintenance of more than one identical copy of the online redo log. To multiplex the online logs, create multiple members in each redo log group. The degree of multiplexing is directly related to the number of members in each group.

■ **control file**

The automated maintenance of more than one identical copy of a database's control file. To multiplex the control file, create multiple entries in the CONTROL_FILES initialization parameter.

- **backup set**

Datafile blocks included in the same RMAN backup set are multiplexed, i.e., mixed together. Blocks from all datafiles in the backup set are interspersed with blocks from the other datafiles in the set.

- **archived redo logs**

The Oracle process ARC*n* is able to archive multiple copies of a redo log. You can multiplex archived redo logs by setting LOG_ARCHIVE_DEST_*n* (where *n* is an integer from 1 to 5, allowing up to four extra copies) or LOG_ARCHIVE_DUPLEX_DEST (allowing one extra copy) in your INIT.ORA file.

See Also: [mirroring](#)

multiple ARC*n* processing

Using multiple ARC*n* processes to archive online redo logs to one or more locations. Multiple ARC*n* processing prevents the bottleneck that occurs when LGWR writes to the online redo log faster than a single archive process can write to the archive destination(s). You can enable this feature at startup or at runtime by setting the initialization parameter LOG_ARCHIVE_MAX_PROCESS = *n*, where *n* is any integer from 1 to 10.

NOARCHIVELOG mode

The mode of the database in which Oracle does not require filled online redo logs to be archived to disk. Specify the mode at database creation or change it by using the ALTER DATABASE command. Oracle does not recommend running in NOARCHIVELOG mode because it severely limits the possibilities for recovery of lost data.

See Also: [archived redo log](#), [ARCHIVELOG mode](#)

normal archiving transmission

The transmittal of archived redo log files to a local disk.

See Also: [standby transmission](#)

offline tablespace

A tablespace that is not available to users when the database is open. You can only take a tablespace offline while the database is open. If a tablespace is taken offline, all online datafiles contained in the tablespace are taken offline.

You can take a tablespace offline using the ALTER TABLESPACE OFFLINE statement with three different options:

- **NORMAL**
All the files in the tablespace are checkpointed, then taken offline. This option is sometimes called *offline clean*. If any datafile belonging to the tablespace is not available, the tablespace cannot be taken offline normal. Datafiles in a tablespace taken offline cleanly do not need to be recovered before the tablespace is brought back online.
- **TEMPORARY**
All datafiles in the tablespace that are accessible to Oracle are checkpointed, then taken offline. Files that were checkpointed by the OFFLINE TEMPORARY command do not need recovery. Datafiles that were not checkpointed because they were not accessible at the time of an OFFLINE IMMEDIATE command must be recovered before the tablespace is brought back online.
- **IMMEDIATE**
All files in the tablespace are taken offline without any attempt to checkpoint the files first. All files in the tablespace must be recovered before the tablespace is brought online.

See Also: [offline datafile](#)

offline datafile

A datafile that is not available to users when the database is open. In exceptional circumstances, Oracle will automatically take a datafile offline if required. This file will need recovery before it can be brought online.

You can take a datafile offline either:

- As a consequence of an ALTER TABLESPACE OFFLINE operation.
- By issuing the command ALTER DATABASE DATAFILE *filename* OFFLINE. You must recover it before bringing it back online. You can issue this statement while the database is mounted or open.

See Also: [offline tablespace](#)

online datafile

A datafile that users can access. The database can be open or mounted when you issue the command `ALTER DATABASE DATAFILE filename ONLINE`. If the database is open, the datafile must be consistent with the rest of the database before you can bring it online. If the database is mounted, then you can bring the datafile online without being consistent with the other datafiles, but it will require recovery before the database is opened.

See Also: [online tablespace](#)

online redo log

The online redo log is a set of two or more files that record all changes made to Oracle datafiles and control files. Whenever a change is made to the database, Oracle generates a redo record in the redo buffer. The LGWR process flushes the contents of the redo buffer into the online redo log.

The *current online redo log* is the one being written to by LGWR. When LGWR gets to the end of the file, it performs a *log switch* and begins writing to a new log file. If you run the database in ARCHIVELOG mode, then the ARC*n* process or processes copy the redo data into an *archived redo log*.

See Also: [archived redo log](#)

online tablespace

A tablespace that is available to users while the database is open. You can make a tablespace available for access by users by issuing the command `ALTER TABLESPACE tablespace_name ONLINE`. The database must be open to alter a tablespace online, and all files in the tablespace must be consistent with the rest of the database before the tablespace can be made online.

See Also: [online datafile](#)

open backup

A backup of one or more datafiles taken while a database is open. When you make an O/S backup while the database is open, you must put the tablespaces in hot backup mode by issuing an `ALTER TABLESPACE BEGIN BACKUP` command. When you make a backup using Recovery Manager while the database is open, however, you do not need to put the tablespaces in hot backup mode.

See Also: [hot backup mode](#)

open database

A database that is available to users to query and update. The database is opened either automatically through a STARTUP statement or explicitly through an ALTER DATABASE OPEN statement.

orphaned backups

Backups and copies that are unusable because they belong to incarnations of the database that are not direct ancestors of the current incarnation. For a visual depiction of orphaned backups, see ["Reporting on Orphaned Backups"](#) on page 4-23.

parallel recovery

A form of recovery in which several processes simultaneously apply changes from redo log files. Instance and media recovery can be parallelized automatically by specifying an initialization parameter or options to the SQL/SQL*Plus RECOVER command. Oracle uses one process to read the log files sequentially and dispatch redo information to several recovery processes, which apply the changes from the log files to the datafiles.

See Also: [serial recovery](#)

parallelization

Allocating multiple channels for Recovery Manager backup and recovery operations. You can parallelize:

- Backup set creation by allocating multiple channels before issuing a **backup** command.
- File copy creation by allocating multiple channels and including multiple files to be copied within a single **copy** command.
- Restore operations, with the degree of parallelism depending on the number of channels allocated as well as the distinct number of backup sets or file copies that must be read during the restore operation.
- Recovery operations when applying incremental backups, with the degree of parallelism depending on the number of channels allocated and also the distinct number of backup sets that are available to read from.

partial resync

See [resynchronization](#)

password files

A file created by the ORAPWD command. A database must use password files if you wish to connect as SYSDBA over a network. For a more comprehensive explanation, see the *Oracle8i Administrator's Guide*.

physical backups

Physical database files that have been copied from one place to another. The files can be datafiles, archived redo logs, or control files. You can make physical backups using Recovery Manager or with O/S commands such as the UNIX **dd**.

physical schema

The datafiles, tablespaces, redo threads, and redo logs that exist in a database at a given time. Issue the RMAN **report schema** command to obtain a list of tablespaces and datafiles.

A full resynchronization of the recovery catalog updates all changed RMAN metadata, including physical schema information. If the database is open, RMAN also gathers information about rollback segments. A partial resynchronization of the recovery catalog does not update physical schema or rollback information.

See Also: [resynchronization](#)

pluggable tablespace

See [transportable tablespace](#)

proxy copy

The functionality that enables a media manager to take over the transfer of data between the media storage device and disk during RMAN backup and restore operations.

See Also: [media manager](#)

read-only database

A database opened with the ALTER DATABASE OPEN READ ONLY command. As their name suggests, read-only databases are for queries only and cannot be modified. Oracle allows a standby database to be run in read-only mode, which means that it can be queried while still serving as an up-to-date emergency replacement for the primary database.

read-only tablespace

A tablespace whose status has been changed to prevent it from being updated. You put in read-only mode by executing the SQL statement `ALTER TABLESPACE <tablespace> READ ONLY`. Typically, you put a tablespace in read-only mode to reduce the frequency with which it is backed up. For example, instead of backing up the tablespace nightly, you reduce the backup frequency to once a month.

Note: The longer the duration between backups of a tablespace, the longer you will need to retain your backup media and the larger the risk of failed backup media (as you will have backed it up fewer times).

recover

- (1) A Recovery Manager command that updates a restored datafile by the application of incremental backups (if they exist) and then by the application of archived or online redo logs.
- (2) A SQL*Plus command that updates a restored file by the application of archived or online redo logs.

See Also: [recovery](#)

recovery

The application of redo data or incremental backups to database files in order to reconstruct lost changes. The three types of recovery are *instance recovery*, *crash recovery*, and *media recovery*. Oracle performs the first two types of recovery automatically using online redo records; only media recovery requires you to restore a backup and issue commands. Only Recovery Manager allows you to recover datafiles by applying incremental backups.

See Also: [complete recovery](#), [incomplete recovery](#), [media recovery](#)

recovery catalog

A set of Oracle tables and views used by Recovery Manager to store information about Oracle databases. Recovery Manager uses this data to manage the backup, restore, and recovery of Oracle databases. If you choose not to use a recovery catalog, RMAN uses the target database control file.

See Also: [recovery catalog database](#)

recovery catalog database

An Oracle database that contains a recovery catalog schema. You should not store the recovery catalog in your target database.

Recovery Manager

A utility that backs up, restores, and recovers Oracle databases. You can use it with or without the central information repository called a *recovery catalog*. If you do not use a recovery catalog, RMAN uses the database's control file to store information necessary for backup and recovery operations. You can use RMAN in conjunction with a media manager, which allows you to back up files to tertiary storage.

See Also: [backup piece](#), [backup set](#), [copy](#), [media manager](#), [recovery catalog](#)

recovery set

One or more tablespaces that are being recovered to an earlier point in time during TSPITR. After TSPITR, all database objects in the recovery set have been recovered to

See Also: [auxiliary set](#), [tablespace point-in-time recovery \(TSPITR\)](#)

redo log

A file containing redo records. There are two types of redo logs: *online redo logs* and *archived redo logs*.

The online redo log is a set of two or more files that records all changes made to Oracle datafiles and control files. The LGWR process records the redo records in the log. The *current online redo log* is the one LGWR is currently writing to.

The archived redo log, also known as the offline redo log, is a copy of the online redo log that has been copied to an offline destination. If the database is in ARCHIVELOG mode, the ARC*n* process or processes copy each online redo log to one or more archive log destinations after it is filled.

See Also: [archived redo log](#), [online redo log](#), [redo record](#)

redo log buffer

The memory buffer in the system global area (SGA) in which Oracle logs redo records. The background process LGWR flushes the buffers into the current online redo log.

See Also: [redo record](#)

redo log groups

Each online redo log belongs to a group. A group has one or more identical members. A multiplexed redo log is a redo log in which the redo groups have multiple members.

redo record

A group of change vectors describing a single, atomic change to the database. Oracle constructs redo records for all data block changes and saves them on disk in the current online redo log. Redo records allow changes to database blocks to be reconstructed should data loss occur.

See Also: [redo log](#)

registration

In RMAN, the execution of a **register database** command in order to record the existence of a target database in the recovery catalog.

RESETLOGS option

A method for opening a database that results in a new database incarnation, the resetting of the log sequence number to 1, and the re-formatting or re-creation of the online redo logs. A database must be opened with the RESETLOGS keyword after:

- Incomplete recovery
- Recovery using a backup control file

resilvering a mirror

Informing the operating system or hardware managing the mirror that you want to refresh a broken mirror from the half that is up-to-date and then maintain both sides of the mirror.

See Also: [breaking a mirror](#), [mirroring](#)

restore

The replacement of a lost or damaged file with a backup. You can restore files either with O/S commands such as UNIX **cp** or the RMAN **restore** command.

See Also: [recover](#)

resync

See [resynchronization](#)

resynchronization

The operation that updates the recovery catalog with current information from the target database control file. You can initiate a full resynchronization of the catalog by issuing a **resync catalog** command.

Partial resynchronizations transfer information to the recovery catalog about archived redo logs, backup sets and datafile copies. Partial resynchronizations will not transfer information such as:

- New datafiles
- New or removed tablespaces
- New or removed online log groups and members

If Recovery Manager determines that a full or partial resynchronization is necessary, it initiates one automatically before commands such as **backup**, **copy**, **restore**, and **recover**.

RMAN

See [RMAN](#)

rolling back

The use of rollback segments to undo uncommitted transactions applied to the database during the rolling forward stage of recovery.

See Also: [recovery](#), [rolling forward](#)

rolling forward

The application of redo records or incremental backups to datafiles and control files in order to recover changes to those files.

See Also: [recovery](#), [rolling back](#)

SBT

System Backup to Tape

See Also: [media management interface](#)

serial recovery

A form of recovery in which a single process applies the changes in the redo log files sequentially.

See Also: [parallel recovery](#)

SGA (System Global Area)

A group of shared memory structures that contain data and control information for one Oracle database instance. The SGA and Oracle processes constitute an Oracle instance. Oracle automatically allocates memory for an SGA whenever you start an instance and the operating system reclaims the memory when you shut down the instance. Each instance has one and only one SGA.

snapshot control file

A copy of a database's control file taken by Recovery Manager. RMAN uses the snapshot control file to read a consistent version of a control file when either resynchronizing the recovery catalog or backing up the control file. A snapshot control file is created by Recovery Manager using the same Oracle code that creates backup control files: `ALTER DATABASE BACKUP CONTROL FILE TO 'location'`.

split block

See [fractured block](#)

staging

The process of restoring archived logs from tertiary storage to disk in order to allow recovery to proceed. RMAN stages the logs to disk when the **recover** command is executed. To use this feature, you must configure a media manager.

See Also: [media manager](#)

stand-alone commands

RMAN commands that you do not have to execute within the brackets of a **run** command.

See Also: [job commands](#)

standby database

An identical copy of a production database that you can use for disaster protection. You can update your standby database with archived redo logs from the production database in order to keep it current. Should a disaster destroy the production database, you can activate your standby database and make it the new production database.

standby transmission

The transmittal of archived redo log files via a network to either a local or remote standby database.

See Also: [standby database](#)

stored script

A sequence of RMAN commands stored in the recovery catalog.

See Also: [recovery catalog](#)

switch

A Recovery Manager command which converts a datafile copy into a datafile used by an Oracle database. It performs the equivalent function of the SQL statement `ALTER DATABASE RENAME FILE 'original_name' TO 'new_name'`, and also marks the datafile copy as no longer available.

system change number (SCN)

A stamp that defines a committed version of a database at a point in time. Oracle assigns every committed transaction a unique SCN.

tablespace

A database is divided into one or more logical storage units called tablespaces. Each tablespace has one or more physical datafiles exclusively associated with it.

See Also: [datafile](#)

tablespace point-in-time recovery (TSPITR)

The recovery of one or more non-SYSTEM tablespaces to a point in time that is different from the database. You can use either RMAN or O/S methods to perform TSPITR.

tag

A user-specified character string that acts as a symbolic name for a backup set or image copy. You can specify a tag when executing the **restore** or **change** command. The maximum length of a tag is 30 characters.

tail of the log

The most recent redo record in the redo log file. As users make changes to the database, the tail keeps moving forward. Since the latest checkpoint is always temporally behind the tail of the log, it is said to *lag* the tail of the log. If the checkpoint lags the tail of the log significantly, recovery time increases.

tape streaming

Writing output to a tape drive fast enough to keep the tape constantly busy.

tape drive

A piece of hardware that reads and writes magnetic tapes.

tape silo

See [ATL \(automated tape library\)](#)

tape volume

One physical piece of tape media.

target database

In RMAN, the database that you are backing up or restoring.

thread

Each Oracle instance has its own set of online redo log groups. These groups are called a *thread* of online redo. In non-OPS environments, each database has only one thread that belongs to the instance accessing it. In OPS environments, each instance has a separate thread, i.e., each instance has its own online redo log. Each thread has its own current log member.

time-based recovery

The incomplete recovery of database files to a non-current time. Time-based recovery is also known as *point-in-time recovery*. There are two types:

- Database point-in-time recovery (DBPITR), which is the incomplete recovery of all datafiles and control file to a time before the most recent time.
- Tablespace point-in-time recovery (TSPITR), which is the incomplete recovery of all datafiles in one or more tablespaces on an auxiliary database to a specific time before the most current time. The tablespace is then re-integrated into the original database.

See Also: [incomplete recovery](#), [media recovery](#), [recovery](#), [TSPITR](#)

transportable tablespace

A feature that allows you to transport a set of tablespaces from one database to another. Transporting or "plugging" a tablespace into a database is like creating a tablespace with pre-loaded data. This feature is often an advantage because:

- It is faster than import/export or unload/load, since it involves only copying datafiles and integrating metadata.
- You can use it to move index data, allowing you to avoid rebuilding indexes.

TSPITR

See [tablespace point-in-time recovery \(TSPITR\)](#)

validation

A test that checks whether a backup set or copy can be restored. RMAN scans all of the copies or backup pieces in the specified backup sets and looks at the checksums to verify that the contents can be successfully restored.

Use the **restore ... validate** or **validate backupset** command when you suspect that one or more copies or backup pieces in a backup set are missing or have been damaged. Note that **restore ... validate** and **validate backupset** actually test whether the files can be restored, whereas **change ... crosscheck** and **crosscheck** merely examine the file headers.

See Also: [crosscheck](#), [media manager](#), [recovery catalog](#)

whole database backup

A backup of the control file and all datafiles that belong to a database.

See Also: [backup](#)

Index

A

- ABORT option
 - SHUTDOWN command, 3-3, 14-17, 14-18, 14-26, 15-12
- active online redo log
 - loss of group, 15-9, 15-10
- ALERT files, 15-16
 - checking after RESETLOGS, 14-34
- allocate channel command (RMAN), 11-9
 - for delete option, 8-26, 11-13
 - for maintenance option, 11-13
- ALTER DATABASE command
 - BACKUP CONTROLFILE option, 3-6
 - BACKUP CONTROLFILE TO TRACE option, 13-12
 - CLEAR LOGFILE GROUP option, 16-17
 - CLEAR UNARCHIVED LOGFILE option, 2-14, 16-25
 - CREATE DATAFILE option, 14-6
 - DATAFILE OFFLINE DROP clause, 16-27
 - DATAFILE ONLINE option, 14-31
 - ENABLE THREAD option, 16-24
 - MOUNT STANDBY DATABASE option, 16-6, 16-29
 - NORESETLOGS option, 13-14, 14-33
 - OPEN READ ONLY option, 16-19, 16-20, 16-21
 - OPEN RESETLOGS option, 6-7
 - RECOVER ... FROM option, 14-8, 14-12
 - RECOVER AUTOMATIC LOGFILE option, 14-15
 - RECOVER LOGFILE option, 14-14
 - RENAME DATABASE clause, 9-9
 - RESETLOGS option, 13-14, 14-17, 14-18, 14-33
 - alter database command (RMAN), 11-15
- ALTER SYSTEM command
 - ARCHIVE ALL option
 - using to archive online redo logs, 3-5
 - ARCHIVE LOG CURRENT option, 16-4, 16-21
 - RESUME option, 13-10
 - SUSPEND option, 13-9
- ALTER TABLESPACE command
 - BEGIN BACKUP option, 13-6, 13-9
 - BEGIN/END BACKUP option, 6-28
 - END BACKUP option, 13-9
- application error
 - definition, 1-12
- applying log files
 - SQL*Plus, 14-12
- ARCHIVE LOG CURRENT option
 - ALTER SYSTEM command, 16-4, 16-21
- archived redo logs, 2-14, 2-18, 2-21
 - ALTER SYSTEM ARCHIVE ALL command, 3-5
 - applying during media recovery, 14-10, 14-14
 - backing up, 8-19
 - copies, listing, 7-2
 - deleting after recovery, 14-8
 - destination states, 16-11
 - destinations
 - re-archiving to failed, 16-11
 - errors during recovery, 14-16
 - location during recovery, 14-10
 - loss of, 15-10
 - multiplexing, 2-19, 16-10
 - normal transmission of, 2-21
 - registering, 6-5
 - restoring to disk, 14-7
 - restoring using RMAN, 9-13

- specifying destinations for, 2-19
- standby transmission of, 2-21
- status information, 2-4, 2-18
- transmitting, 2-21, 16-7

archivelog backups

- using RMAN, 8-8

ARCHIVELOG mode, 2-16

- advantages, 2-17
- archiving, 2-15
- backup options, 3-20
- backup strategies when using, 3-16
- datafile loss in, 15-3
- definition of, 2-16
- distributed databases, 3-22
- overview, 1-10
- running in, 2-16
- strategies for backups in, 3-16

archivelogRecoverSpecifier clause (RMAN), 11-17

archiving

- advantages, 2-15
- after inconsistent closed backups, 3-5
- after open backups, 3-5
- ALTER SYSTEM ARCHIVE ALL, 3-5
- destination states, 16-11
- disadvantages, 2-15
- to failed destinations, 16-11
- viewing information on, 2-4, 2-18

AS SELECT option

- CREATE TABLE command, 14-4

AUTORECOVERY option

- SET command, 14-12

auxiliary databases

- for RMAN TSPITR
 - converted filenames, A-15
 - using datafile copies, A-14

auxiliary sets

- for RMAN TSPITR, A-4
- naming datafiles in tablespaces, A-13

available option (RMAN)

- change command, 6-8

avoiding dangerous backup techniques, 3-23

B

backup command (RMAN), 4-29, 8-2, 8-26, 11-21

- format parameter, 4-31
- proxy only option, 4-19
- proxy option, 4-19
- skip offline option, 8-15

backup control files

- recovery using, 9-20

BACKUP CONTROLFILE option

- ALTER DATABASE command, 3-6, 13-3

BACKUP CONTROLFILE TO TRACE option

- ALTER DATABASE command, 13-3, 13-12

backup files

- user-created, cataloging, 6-28

backup formats, 3-12 to 3-14

- backup sets, 3-12
- logical backups, 3-14

backup methods, 3-8 to 3-11

- comparison of, 3-11
- Enterprise Backup Utility, 3-11
- Export utility, 3-10
- operating system, 3-10
- Recovery Manager, 3-9

backup options

- ARCHIVELOG mode, 3-20
- NOARCHIVELOG mode, 3-20

backup pieces

- naming, 4-31

backup sets

- creating using backup command, 4-31
- crosschecking, 6-10
- duplexing, 8-22
- listing, 7-2
- naming, 4-31
- organizing, 4-29
- sizing, 4-31
- testing restore of, 6-18
- type of backup format, 3-12

Backup Solutions Program (BSP), 4-20

- Legato Storage Manager (LSM), 4-20

backup strategy

- overview, 1-11 to 1-17

BACKUP_TAPE_IO_SLAVES initialization

- parameter, 8-23

backups

- after OPEN RESETLOGS option, 3-19
- after structural changes to database, 3-18

- after UNRECOVERABLE operations, 16-28
- archived redo logs, 8-19
 - using RMAN, 8-8
- ARCHIVELOG mode in, 3-16
- backup command (RMAN), 8-2
- backup sets, 4-28
- choosing a strategy, 3-15 to 3-24
- consistent, 1-16
- consistent whole database, 3-2
- control file, 2-7, 2-9, 3-8, 13-11
 - using RMAN, 8-6, 8-7
- cumulative incremental, 4-42, 4-50, 4-53, 8-22, 16-7, 16-30
- datafile, 3-8
 - using RMAN, 8-4, 8-5, 8-6
- DBVERIFY utility, 13-5
- definition, 1-2
- distributed databases, 3-22
 - ARCHIVELOG mode, 3-22
 - NOARCHIVELOG mode, 3-22
- duplexing, 8-22
- effects of archiving on, 2-16
- Enterprise Backup Utility, 3-10
- Export utility, 3-10, 3-14, 3-22
- frequency, 3-18
- full, 4-37
- generating reports for, 7-2
- guidelines, 3-14 to 3-24
 - distributed database constraints, 3-22
 - Export utility, 3-22
 - frequency, 3-18
 - multiplexing files, 3-17
 - often-used tablespaces, 3-19
 - storing old backups, 3-21
 - structural changes, 3-18
 - testing strategies, 3-24
 - unrecoverable operations, 3-19
 - whole database backups after OPEN RESETLOGS, 3-19
- image copies, 4-28, 4-45
- importance, 1-11
- inconsistent, 1-16
 - closed database, 3-4
 - in NOARCHIVELOG mode, 3-4
 - whole database, 3-4
- incremental, 8-21
 - differential, 4-40
 - using RMAN, 8-9
- keeping, 8-26
- listing files needed, 13-2
- logical, definition, 1-2
- NOARCHIVELOG mode, in, 3-16, 8-23
- noncumulative incremental, 4-40
- offline, 3-18
- online, 3-18
- online redo logs, 3-23
- options
 - ARCHIVELOG mode, 3-20
 - NOARCHIVELOG mode, 3-20
- Parallel Server Environment, 8-25
- parallelization, 4-34, 8-23
- physical, definition, 1-2
- planning before database creation, 3-14
- procedures, 13-3 to 13-17
 - offline tablespaces, 13-10
- procedures for
 - offline datafiles, 13-10
- recovery catalog, 4-14
- Recovery Manager types, 4-28
- Recovery Manager, user tags for, 4-46
- reporting objects needing backups, 7-5
- restoring whole database backup, 14-17
- RMAN error handling, 8-26
- status
 - checking backup, 13-7
 - stored scripts, 6-20
 - storing, 3-21
 - tablespace, 3-19, 13-8
 - using RMAN, 8-4, 8-5, 8-6
 - techniques to avoid, 3-23
 - test strategies, 3-24
 - to disk, 3-10
 - types, 3-2, 13-3 to 13-17
 - whole database, 3-2, 13-3 to 13-5
 - backup control files and, 3-3
 - preparing to take, 13-4
 - using RMAN, 8-3
- backups, using RMAN, 8-2
- BEGIN BACKUP option
 - ALTER TABLESPACE command, 13-6

BEGIN/END option
ALTER TABLESPACE command, 6-28
BSP
See Backup Solutions Program

C

cancel-based media recovery
definition, 3-29
procedures, 14-20, 14-23, 14-26, 14-29
CASE1.RCV sample script
setting size limits for backup pieces, 5-20
catalog command (RMAN), 6-5, 6-27, 11-32
cataloging O/S copies, 8-25
CATALOG.SQL script, 6-2
CATPROC.SQL script, 6-2
CATRMAN.SQL script, 6-5
change command (RMAN), 6-10, 11-35
available option, 6-8
delete option, 6-28
unavailable option, 6-8
change-based recovery, 14-32 to 14-35
coordinated in distributed databases, 15-16
definition, 3-29
channel control
overview of RMAN, 4-24
channels
allocating, 4-24
parallelization of, 4-26
character set
setting for use with RMAN, 5-3
CLEAR LOGFILE GROUP option
ALTER DATABASE command, 16-17
CLEAR UNARCHIVED LOGFILE option
ALTER DATABASE command, 16-25
clearing redo log files, 2-14
clone databases
exporting, B-13
opening, B-13
preparing for TSPITR, B-11
preparing parameter files for, B-10
recovering, B-12
closed backups, 3-4
code examples
description of, 11-4
cold backups, 3-4
whole database backups, 13-3
command files
Recovery Manager, 4-10
command line
arguments for RMAN, 4-9
command line parameters (RMAN), 11-39
commands, Recovery Manager
allocate, 11-9
allocate channel for delete, 8-26
allocate channel for maintenance/delete, 11-13
alter database, 11-15
archivelogRecoverSpecifier clause
(RMAN), 11-17
backup, 4-29, 8-2, 8-26, 11-21
proxy only option, 4-19
proxy option, 4-19
skip offline option, 8-15
catalog, 6-5, 6-27
change, 6-10, 11-35
delete option, 6-28
connect, 11-44, 11-46
copy, 11-48
create catalog, 11-52
create script, 11-54
crosscheck, 11-57
debug, 11-62
delete expired backupset, 11-63
delete script, 11-65, 11-66
drop catalog, 6-35, 11-68
duplicate, 4-52, 11-69
execute script, 6-20
host, 11-74
list, 4-20, 11-76
incarnation of database option, 6-7
listObjList clause, 11-84
overview, 4-5
print script, 11-86
recover, 4-49, 9-15, 11-88
register, 6-6, 11-93
release channel, 11-95
release channel (of type maintenance), 11-96
replace script, 11-97
replicate, 9-11, 11-100
replicate controlfile, 9-11

- report, 4-22, 11-102
 - need backup option, 7-5
- reset database, 6-7, 11-110
 - incarnation option, 6-7
- restore, 9-17, 11-112
- resync catalog, 4-14, 6-23, 11-118
 - from controlfilecopy option, 6-33
- rman, 11-121
- run, 11-124
- send, 5-20, 11-127
- set, 11-129
 - duplex parameter, 8-22
 - maxcorrupt for datafile option, 8-26
 - newname for datafile option, 9-9
- set (within run command), 11-133
- shutdown command, 11-137
- sql, 11-140
- startup, 11-142
- summary, 11-5
- switch, 11-144
- until, 11-42, 11-146
- upgrade catalog, 6-34, 11-148
- validate, 11-150
- commands, Recovery manager
 - catalog, 11-32
- commands, SQL
 - ALTER DATABASE, 14-8, 14-12, 14-14, 16-17, 16-25
 - CREATE CONTROLFILE, 16-25
- commands, SQL*Plus
 - RECOVER, 14-9
 - UNTIL TIME option, 14-31
 - RECOVER DATAFILE, 14-10
 - RECOVER TABLESPACE, 14-9
 - SET, 14-8, 14-12
- COMPATIBLE parameter, 16-26
 - standby database, 16-5
- compilation and execution of RMAN
 - commands, 4-6
- complete export, 13-17
- complete recovery
 - definition, 1-19, 3-28
 - procedures, 14-20
 - using RMAN, 9-18
- connect command (RMAN), 11-44, 11-46
- connection options
 - Recovery Manager, 5-8
- consistent backups
 - definition, 1-16
- consistent whole database backups, 3-2, 13-3
- constraints
 - restore, 4-48
- control file backups
 - definition, 3-8
 - using RMAN, 8-6, 8-7
- control files, 2-3 to 2-7
 - backing up, 13-3, 13-11
 - backup and recovery, 9-20
 - contents, 1-7, 2-3
 - CONTROL_FILES initialization parameter
 - for primary and standby databases, 16-25
 - definition, 1-7
 - effect on standby databases, 16-25
 - finding filenames, 13-3
 - incomplete recovery, 14-27
 - loss of, 15-11, 15-13
 - maintaining backups, 2-7, 2-9
 - mirrored, 2-9
 - loss of, 15-12
 - multiplexed, 2-7, 2-8, 2-9
 - number of, 2-9
 - overview, 1-7
 - purpose, 1-7
 - refreshing standby database, 16-29
 - restoring using RMAN, 9-11, 9-12
 - setting the CONTROL_FILES initialization parameter, 2-9
 - snapshot
 - specifying location of, 5-3
 - time-based recovery, 14-29
 - updating, 2-5
 - using instead of a recovery catalog, 4-15
- CONTROL_FILES initialization parameter, 2-9, 9-11, 15-13, 16-25
- coordinated time-based recovery
 - distributed databases, 15-16
- copy command (RMAN), 11-48
- corrupt datafile blocks
 - records in control file, 4-36
 - RMAN and, 4-36

- setting maximum for backup, 8-26
- corruption detection, 4-54, 4-55
- crash recovery
 - definition, 1-19
- create catalog command (RMAN), 11-52
- CREATE CONTROLFILE command
 - and recovery, 9-16
 - standby database, effect on, 16-25
- CREATE DATAFILE option
 - ALTER DATABASE command, 14-7
- create script command (RMAN), 11-54
- CREATE TABLE command
 - AS SELECT option, 14-4
- CREATE TABLESPACE command, 15-3
- creating
 - standby databases, 16-3
- creating duplicate databases, 10-2
 - on a remote host, 10-10
- creating test databases, 4-52
- creating the recovery catalog, 6-2
- crosscheck command (RMAN), 11-57
- crosschecking
 - definition, 4-18
 - recovery catalog with the media manager, 6-9
- cumulative export, 13-17
- cumulative incremental backups, 4-42, 8-22

D

- data dictionary
 - views, 13-6, 13-10
- database schema
 - generating reports, 7-8
- database structures
 - datafiles, 1-6
 - online redo log, 1-8
 - overview, 1-6 to 1-11
 - physical, 1-6 to 1-11
 - redo log files, 1-8
 - rollback segment, 1-8
- databases
 - altering physical structure
 - standby databases, impact on, 16-22
 - backing up
 - using Recovery Manager, 4-29

- creating duplicate
 - on a remote host, 10-10
 - creating duplicate using RMAN, 10-2
 - creating test, 10-2
 - db identifier, 6-5
 - disaster recovery planning, 16-1 to 16-29
 - listing for backups, 13-2
 - media recovery procedures, 14-1 to 14-32
 - media recovery scenarios, 15-1 to 15-15
 - name stored in control file, 2-3
 - physical structure, 1-6 to 1-11
 - recovery
 - after control file damage, 15-12
 - recovery after OPEN RESETLOGS option, 14-35
 - registering in recovery catalog, 6-4, 6-5
 - standby, 16-2 to 16-29
 - advantages, 16-2
 - control files, 16-25, 16-29
 - creating, 16-2
 - creating, procedures for, 16-3
 - datafiles, taking offline, 16-27
 - definition, 16-2
 - direct path operations, 16-27
 - initialization parameters, 16-25
 - maintaining, 16-6
 - manual recovery mode, 16-5, 16-6
 - modes, 16-5
 - read-only mode, 16-5, 16-19
 - redo log files, altering, 16-24
 - renaming datafiles, 16-24
 - sustained recovery mode, 16-5
 - transmitting archived redo logs, 16-7
 - suspending, 13-9
 - unregistering in recovery catalog, 6-6
 - whole database backups, 13-3 to 13-5
- datafile backups
 - using RMAN, 8-4, 8-5, 8-6
- DATAFILE ONLINE option
 - ALTER DATABASE command, 14-31
- datafiles, 1-6
 - adding to primary database
 - effect on standby database, 16-23
 - backing up, 3-8
 - offline, 13-10
 - using Recovery Manager, 4-29

- backups needed, listing, 7-10, 7-11, 7-12, 8-15
- backups, listing, 7-2
- checking backup status, 13-7
- copies, listing, 7-2
- listing
 - for backup, 13-2
 - unrecoverable, 7-5
- loss of, 15-2
- named in control files, 2-3
- overview, 1-6
- recovery
 - creation, 14-6
 - guidelines, 4-49, 9-16
 - using RMAN, 9-15
 - without backup, 14-6
- registering, 6-5
- renaming
 - effect on standby database, 16-24
- restoring, 4-47
- restoring using RMAN, 9-2
- usage, 1-6
- viewing
 - backup status, 13-7
 - files needing recovery, 14-3
- dates
 - specifying in RMAN commands, 5-2
- db identifier, 6-5
- DB_FILE_NAME_CONVERT initialization
 - parameter, 4-53
 - using with RMAN duplicate command, 10-7
- DB_FILE_STANDBY_NAME_CONVERT
 - initialization parameter, 16-26
 - for primary and standby databases, 16-25
- DB_FILES initialization parameter, 16-26
- DBA_DATA_FILES view, 13-6, 13-10
- DBVERIFY utility, 13-5
- debug command (RMAN), 11-62
- delete expired backupset command
 - (RMAN), 11-63
- delete script command (RMAN), 11-65, 11-66
- destination states for archived redo logs, 16-11
- differential incremental backups, 4-40
- direct path operations
 - standby databases, 16-27
- disaster recovery
 - using a standby database for, 16-1 to 16-29
- disconnecting
 - from Recovery Manager, 5-12
- disk failures, 1-13
- distributed databases
 - backups, 3-22
 - change-based recovery, 15-16
 - coordinated time-based recovery, 15-16
 - media recovery and snapshots, 15-17
 - recovery, 15-15
 - taking backups, 3-22
- drop catalog command (RMAN), 11-68
- dropped tables
 - recovery, 15-15
- dropping the recovery catalog, 6-35
- duplex parameter (RMAN)
 - set command, 8-22
- duplexing backup sets, 4-33, 8-22
- duplicate command (RMAN), 4-52, 11-69
- duplicate databases
 - creating, 4-52, 10-2
 - creating non-current, 10-19
 - creating on a remote host with same
 - filesystem, 10-10
 - creating on local host, 10-17
 - creating on remote host with different
 - filesystem, 10-12
 - DB_FILE_NAME_CONVERT parameter, 10-5
 - generating filenames, 10-3
 - nofilenamecheck option, 10-6
 - preparing for duplication, 10-7
 - restrictions, 10-3
 - set auxname command (RMAN), 10-5
 - set newname command (RMAN), 10-5
 - skipping tablespaces, 10-5
- duplicating a database, 4-52

E

- ENABLE THREAD option
 - ALTER DATABASE command, 16-24
- Enterprise Backup Utility, 3-11
- environment variables
 - NLS_DATE_FORMAT, 5-2
 - NLS_LANG, 5-2

- errors
 - during RMAN backups, 8-26
 - Recovery Manager, 4-12
- Export utility, 3-10, 3-14
 - backups, 3-14, 3-22, 13-16
 - backups and, 3-22
 - export types, 13-17
 - read consistency, 13-16
- exports
 - complete, 13-17
 - cumulative, 13-17
 - incremental, 13-17
 - modes, 13-16

F

- failures
 - disk, 1-13
 - hardware, 1-12
 - instance, 1-12
 - media, 1-13
 - multiplexed online redo logs, 2-12
 - process, 1-12
 - statement, 1-12
 - system, 1-12
 - types, 1-12
 - user errors, 1-12
- file, 6-3
- filenames
 - listing for backup, 13-2
- format parameter (RMAN)
 - backup command, 4-31
- fractured block detection, 4-55
- full backups, 4-37

G

- generating lists using RMAN, 4-20
- generating reports, 4-20, 7-5
- groups
 - archived redo log, 15-6, 15-7
 - online redo log, 15-6, 15-7
- guidelines
 - backups
 - distributed database constraints, 3-22

- Export utility, 3-22
- frequency, 3-18
- multiplexing files, 3-17
- often-used tablespaces, 3-19
- storing old backups, 3-21
- structural changes, 3-18
- testing strategies, 3-24
- unrecoverable operations, 3-19
- whole database backups after OPEN RESETLOGS, 3-19

H

- hardware failures, 1-12
- host command (RMAN), 11-74
- hot backup mode
 - necessary for online O/S backups, 13-7
- hot backups
 - inconsistent whole database backups, 3-4

I

- image copies, 4-45
 - crosschecking, 6-12
 - testing restore of, 6-18
- Import utility, 13-16
 - database recovery, 13-18
 - procedure for using, 13-18
- inactive online redo log
 - loss of, 15-7
- incarnation of database option (RMAN)
 - list command, 6-7
- incarnation option (RMAN)
 - reset database command, 6-7
- incomplete media recovery
 - change-based, 14-32 to 14-35
 - definition, 1-20, 3-28
 - in OPS configuration, 14-13
 - procedures for, 14-26 to 14-35
 - time-based, 14-29 to 14-31
 - using backup control file, 14-13
- incomplete recovery
 - change-based (RMAN), 9-25
 - log sequence-based (RMAN), 9-26
 - time-based (RMAN), 9-24

- using RMAN, 9-23
- with a recovery catalog, 9-24
- without a recovery catalog, 9-27

inconsistent backups

- definition, 1-16

inconsistent whole database backups

- definition, 3-4

incremental backups, 8-21

- differential, 4-40
- using RMAN, 8-9

incremental export, 13-17

initialization parameters

- BACKUP_TAPE_IO_SLAVES, 8-23
- COMPATIBLE, 16-26
- CONTROL_FILES, 9-11, 15-13, 16-25
 - for primary and standby databases, 16-25
- DB_FILE_STANDBY_NAME_CONVERT, 16-26
 - for primary and standby databases, 16-25
- DB_FILES, 16-26
 - for primary and standby databases, 16-25
- LOG_ARCHIVE_DEST, 2-19, 14-8, 14-10, 14-28, 14-31, 16-7
- LOG_ARCHIVE_DEST_n, 2-19, 14-8, 14-10, 16-6, 16-10
- LOG_ARCHIVE_DEST_STATE_n, 16-11
- LOG_ARCHIVE_DUPLEX_DEST, 2-19
- LOG_ARCHIVE_FORMAT, 14-7, 14-10
- PARALLEL_MAX_SERVERS, 14-20
- RECOVERY_PARALLELISM, 14-19
- STANDBY_ARCHIVE_DEST, 16-26

INIT.ORA files, 4-49, 9-15

instance

- failure, 1-12
- recovery, 1-19

integrity checking, 4-54

interrupting media recovery, 14-16

I/O errors

- effect on backups, 4-36

J

job, 4-8

K

keywords

- in syntax diagrams, 11-3

L

level 0 incremental backups, 4-39

list command (RMAN), 4-20, 11-76

- incarnation of database option, 6-7

listing backups and copies, 7-3

listObjList clause (RMAN), 11-84

lists, generating, 7-2 to 7-4

- listing backups and copies, 7-3
- scenarios, 7-10

log sequence number

- requested during recovery, 14-10

log sequence recovery, 3-29

log switches

- multiplexed redo log files and, 2-14
- recovery catalog records, 6-27
- waiting for archiving to complete, 2-14

log writer process (LGWR)

- multiplexed redo log files and, 2-13
- trace files and, 2-13

LOG_ARCHIVE_DEST initialization

- parameter, 14-8, 14-10, 14-28, 14-31
- specifying destinations using, 2-19, 16-10

LOG_ARCHIVE_DEST parameter, 16-7

LOG_ARCHIVE_DEST_n initialization

- parameter, 2-19, 14-8, 14-10, 16-6, 16-10
- REOPEN option, 16-11

LOG_ARCHIVE_DUPLEX_DEST initialization

- parameter
- specifying destinations using, 2-19, 16-10

LOG_ARCHIVE_FORMAT initialization

- parameter, 14-7, 14-10

logical backups

- definition, 3-14

LOGSOURCE variable

- SET command, 14-8, 14-12

loss of

- inactive log group, 15-7

M

- managing RMAN metadata, 6-1
- manual recovery mode, 16-5
- media failure
 - archived redo log file loss, 15-10
 - complete recovery, 14-20 to 14-25
 - control file loss, 15-11, 15-13
 - datafile loss, 15-2
 - definition, 1-13
 - description, 15-2
 - disk controller, 15-2
 - NOARCHIVELOG mode, 14-17
 - online redo log group loss, 15-6
 - online redo log loss, 15-5
 - permanent, 15-2
 - recovery, 14-20 to 14-32
 - distributed databases, 15-15
 - recovery procedures
 - examples, 15-2 to 15-15
 - understanding types, 15-2
- media management
 - backing up files, 4-17
 - Backup Solutions Program, 4-20
 - Legato Storage Manager (LSM), 4-20
 - crosschecking, 4-18
 - linking to software, 5-19
 - maximum file size, 5-20
 - proxy copy, 4-19
 - requirements, 5-19
 - restoring files, 4-18
 - sbttest program, 4-19
 - sending device-specific strings, 5-20
 - troubleshooting, 5-21
 - unique filenames, generating, 5-19
- Media Management Library (MML), 4-17
- media recovery, 14-1 to 14-36
 - ADD DATAFILE operation, 15-3
 - after control file damage, 15-12
 - after OPEN RESETLOGS operation, 14-35
 - applying archived redo logs, 14-10
 - applying redo logs during, 14-10
 - cancel-based, 3-29, 14-20, 14-23, 14-26, 14-29
 - change-based, 3-29, 14-26, 14-32 to 14-35
 - commands, 14-2 to 14-10
 - complete, 1-19, 3-28, 14-20 to 14-25
 - completion of, 14-23, 14-25
 - datafiles
 - guidelines for, 4-49, 9-16
 - without backup, 14-6
 - deciding which files need recovery, 14-3
 - definition, 14-2
 - distributed databases, 15-15
 - coordinated time-based, 15-16
 - dropped table, 15-15
 - effects of archiving on, 2-16
 - error messages, 14-16
 - errors with redo log files, 14-16
 - incomplete, 3-28, 14-26 to 14-35
 - incomplete, definition, 1-20
 - interrupting, 14-16
 - log sequence recovery, 3-29
 - lost control file, 15-11
 - lost files
 - lost archived redo log files, 15-10
 - lost control files, 15-11
 - lost datafiles, 15-2
 - lost mirrored control files, 15-12
 - methods
 - choosing, 3-31
 - NOARCHIVELOG mode, 14-17
 - offline tablespaces in open database, 14-23
 - online redo log files, 15-5
 - open database-offline tablespace, 14-23
 - opening database after, 14-33
 - overview, 1-19 to 1-23
 - restarting, 14-16
 - restoring
 - archived redo log files, 14-7
 - damaged files, 14-6
 - whole database backups, 14-17
 - resuming after interruption, 14-16
 - roll forward phase, 14-10
 - rolling back, 14-2
 - rolling forward, 14-2
 - See Also* recovery
 - snapshots, 15-17
 - successfully applied redo logs, 14-15
 - SYSTEM tablespace, 14-23
 - time-based, 14-26

- transportable tablespaces, 15-4
- types
 - distributed databases, 15-15
 - undamaged tablespaces online, 14-23
 - using Import utility, 13-18
 - using Recovery Manager, 4-49, 9-15
- metadata
 - managing, 6-1
 - Recovery Manager, 4-13
 - storing in control file, 4-15
- mirrored disk storage, 2-9
- mirrored files
 - control files, 2-9
 - loss of, 15-12
 - online redo log, 2-13
 - loss of, 15-5
 - loss of member, 15-5
 - splitting, 13-9
 - suspend/resume mode, 13-9
- mirroring
 - compared to multiplexing, 2-7
- mode
 - ARCHIVELOG, 1-10
 - NOARCHIVELOG, 1-10
 - recovery from failure, 14-17
- MOUNT option
 - STARTUP command, 14-28, 14-30
- MOUNT STANDBY DATABASE option
 - ALTER DATABASE command, 16-6, 16-29
- multiplexing
 - archived redo logs, 2-19, 16-10
 - compared to mirroring, 2-7
 - control files, 2-7, 2-8, 2-9
 - datafiles with Recovery Manager, 4-32
 - redo log files, 2-12
 - groups, 2-13
- multi-threaded recovery, 14-13

N

- name translation for RMAN commands, 4-6
- naming backup sets, 4-31
- newname for datafile option (RMAN)
 - set command, 9-9
- NLS_DATE_FORMAT environment variable, 5-2,

- 8-19, 9-24
- NLS_LANG environment variable, 5-2, 8-19, 9-24
- NOARCHIVELOG mode
 - archiving, 2-15
 - backing up, 8-23
 - backup options, 3-20
 - datafile loss in, 15-2
 - definition, 2-15
 - disadvantages, 14-17
 - distributed database backups, 3-22
 - inconsistent closed backups in, 3-4
 - overview, 1-10
 - recovery, 14-17
 - running in, 2-15
 - strategies for backups in, 3-16
- noncumulative incremental backups, 4-40
- NORESETLOGS option
 - ALTER DATABASE command
 - backing up control file, 13-14

O

- objects owned by SYS
 - and TSPITR using RMAN, A-6
- offline
 - backups, 3-18
 - taking standby database datafiles, 16-27
- online
 - backups, 3-18
- online redo log, 1-8, 15-7
 - active group, 15-6, 15-7
 - applying during media recovery, 14-10
 - archived group, 15-6, 15-7
 - backing up, 2-11, 3-23
 - current group, 15-6, 15-7
 - inactive group, 15-6, 15-7
 - loss of
 - active group, 15-9, 15-10
 - all members, 15-6
 - group, 15-6
 - mirrored members, 15-5
 - recovery, 15-5 to 15-10
 - multiple group loss, 15-10
 - multiplexed, 1-9
 - overview, 1-8

- recorded in control file, 2-3
- status of members, 15-6, 15-7
- unintentional restore of, 3-23
- online redo logs
 - listing log files for backup, 13-2
- open database backup
 - fractured block detection during, 4-55
- open database recovery
 - using RMAN, 9-22
- OPEN READ ONLY clause
 - ALTER DATABASE command, 16-19, 16-20, 16-21
- OPEN RESETLOGS option
 - ALTER DATABASE command, 6-7
- ORA-01578 error message, 14-5
- Oracle Enterprise Manager, 3-10
- Oracle8i utilities
 - Recovery Manager, 4-2
- O/S backups, 13-3 to 13-18
- O/S utilities
 - copying files with, 8-25

P

- PARALLEL clause
 - RECOVER command, 14-20
- parallel recovery, 14-19
- Parallel Server
 - backups and, 8-25
- PARALLEL_MAX_SERVERS initialization
 - parameter, 14-20
- parallelization
 - channels, 4-26
 - factors affecting degree of, 4-27
 - of backups using RMAN, 4-34, 8-23
 - RMAN backups, 4-34, 8-23
- parameters
 - in syntax diagrams, 11-3
- partitioned tables
 - and dropped partitions, B-18
 - and split partitions, B-22
 - performing partial TSPITR, B-15
- password files
 - and Recovery Manager, 5-2
 - connecting to Recovery Manager with, 5-9, 5-11

- connecting to Recovery Manager without, 5-10
- performing backups after unrecoverable operations, 3-19
- physical database structures, 1-6 to 1-11
- point-in-time recovery
 - tablespace, A-2 to A-15, B-1 to B-26
- primary databases
 - preparing for use, B-14
- print script command (RMAN), 11-86
- process failures, 1-12
 - definition, 1-12
- proxy copy
 - overview, 4-19
- proxy only option (RMAN)
 - backup command, 4-19
- proxy option (RMAN)
 - backup command, 4-19

Q

- querying
 - recovery catalog, 7-2

R

- read consistency
 - Export utility, 13-16
- read-only mode
 - standby databases, 16-5, 16-19
- read-only tablespaces
 - backing up, 8-15
 - effect on recovery, 14-5
 - recovering, 14-5
- RECOVER ... FROM option
 - ALTER DATABASE command, 14-8, 14-12
- RECOVER AUTOMATIC LOGFILE option
 - ALTER DATABASE command, 14-15
- RECOVER command, 14-9
 - FROM 'location' option, 16-29
 - MANAGED STANDBY DATABASE option, 16-8, 16-18
 - PARALLEL clause, 14-20
 - unrecoverable objects and standby databases, 14-5
 - UNTIL TIME option, 14-31

- recover command (RMAN), 4-49, 9-15, 11-88
- RECOVER DATAFILE command, 14-10
- RECOVER TABLESPACE command, 14-9
- recovered tablespaces
 - backing up, B-14
- recovery
 - cancel-based, 14-20, 14-23, 14-26, 14-29
 - change-based, 14-32 to 14-35
 - commands, 14-2 to 14-10
 - complete using RMAN, 9-18
 - crash, 1-19
 - disaster using RMAN, 9-35
 - distributed databases
 - with snapshots, 15-17
 - Import utility, 13-18
 - incomplete using RMAN, 9-23
 - instance, 1-19
 - media, 14-1, 15-1
 - complete, 3-28
 - incomplete, 3-28
 - multi-threaded, 14-13
 - of lost or damaged recovery catalog, 6-33
 - open database using RMAN, 9-22
 - parallel processes for, 14-19
 - PARALLEL_MAX_SERVERS initialization
 - parameter, 14-20
 - read-only tablespaces, 14-5
 - See Also* media recovery
 - setting number of processes to use, 14-19
 - strategies
 - deciding what to recovery, 3-28
 - determining when media recovery is
 - necessary, 3-30
 - time-based, 3-28, 14-29 to 14-31
 - using backup control file (RMAN), 9-20
 - using RMAN, 9-15
- recovery catalog, 4-13, 5-4 to 6-33
 - and incomplete recovery, 9-24
 - backing up, 4-14
 - cataloging
 - O/S backups, 6-27
 - changing availability of backup, 6-8
 - changing status of backups to deleted, 6-13
 - connecting to Recovery Manager with, 5-10, 5-11
 - connecting to Recovery Manager without, 5-9, 5-10
 - consequences of using, 5-4
 - creating, 6-2
 - in separate database, 5-5
 - crosschecking, 6-9
 - dropping, 6-35
 - log switch record, 6-27
 - managing size of, 6-26
 - operating with, 4-13
 - operating without, 4-15
 - overview, 4-13
 - querying, 4-20, 7-2
 - recovery of lost or damaged, 6-33
 - refreshing, 6-23
 - registering databases, 6-4, 6-5
 - removing records, 6-16
 - resynchronizing, 4-14, 6-23
 - schema, 5-5
 - setting up, 6-2
 - snapshot control file, 4-14
 - unregistering databases, 6-6
 - updating
 - after schema changes, 6-25
 - upgrading, 6-34
 - views, 12-1
- Recovery Manager
 - advantages to using, 4-4
 - allocate channel for maintenance, 11-13
 - backup method, 3-9
 - backup pieces, 4-31
 - backup sets, 4-31
 - backup types, 4-28
 - duplexed backup sets, 4-33
 - backups, 8-2
 - archived redo logs, 8-8
 - control file, 8-6, 8-7
 - datafile, 8-4, 8-5, 8-6
 - fractured block detection, 4-55
 - image copy, 4-45
 - incremental, 8-9
 - tablespace, 8-4, 8-5, 8-6
 - using tags, 4-46
 - whole database, 8-3
 - channel control

- overview, 4-24
- command line arguments, 4-9
- commands
 - allocate channel, 11-9
 - allocate channel for delete, 8-26
 - alter database, 11-15
 - backup, 4-19, 4-29, 8-2, 8-26
 - catalog, 6-5, 6-27
 - change, 6-10, 6-28
 - connect, 11-44
 - copy, 11-48
 - create catalog, 11-52
 - create script, 11-54
 - crosscheck, 11-57
 - debug, 11-62
 - delete expired backup, 11-63
 - delete script, 11-65
 - drop catalog, 11-68
 - duplicate, 11-69
 - execute script, 6-20
 - host, 11-74
 - interactive use of, 4-10
 - job, 4-9
 - job commands, 4-8
 - list, 6-7
 - overview, 4-5
 - print script, 11-86
 - recover, 4-49, 9-15
 - register, 6-6
 - release channel, 11-95
 - replace script, 11-97
 - replicate controlfile, 9-11
 - report, 7-5
 - reset database, 6-7
 - restore, 9-17
 - resync catalog, 6-33
 - rman, 11-121
 - run, 11-124
 - send, 5-20
 - set, 8-22, 9-9
 - shutdown, 11-137
 - sql, 11-140
 - stand-alone, 4-9
 - stand-alone commands, 4-8
 - startup, 11-142
 - switch, 11-144
 - upgrade catalog, 11-148
 - using command files, 4-10
 - validate, 11-150
- compilation and execution of commands, 4-6
- connecting
 - duplicate database, 5-11
 - with password files, 5-9, 5-11
 - with recovery catalog, 5-10, 5-11
 - without password files, 5-10
 - without recovery catalog, 5-9, 5-10
- connection options, 5-8
- constraints
 - backup, 4-44
 - restore, 4-48
- corrupt datafile blocks, 4-54, 4-55
 - handling I/O errors and, 4-36
- creating duplicate databases, 10-2
- crosschecking recovery catalog, 6-9
- database character set, 5-3
- dates in commands, 5-2
- disconnecting from, 5-12
- errors, 4-12
- fractured block detection in, 4-55
- image copy backups, 4-45
- incomplete recovery
 - with a recovery catalog, 9-24
 - without a recovery catalog, 9-27
- incremental backups
 - cumulative, 4-42
 - differential, 4-40
 - level 0, 4-39
- integrity checking, 4-54
- interactive use of commands, 4-10
- introduction, 4-2
- lists, 7-2
- media management
 - backing up files, 4-17
 - Backup Solutions Program (BSP), 4-20
 - crosschecking, 4-18
 - maximum file size, 5-20
 - media manager, linking with a, 5-19
 - proxy copy, 4-19
 - requirements, 5-19
 - restoring files, 4-18

- testing, 4-19
 - unique filenames, 5-19
- metadata, 4-13, 6-1
 - storing in control file, 4-15
- multiplexing
 - datafiles, 4-32
- name translation, 4-6
- overview, 4-4
- parallelization of backups, 4-34
- password files, 5-2
- PL/SQL job steps, 4-6
- recovery, 9-15
 - after total media failure, 9-35
 - complete, 9-18
 - incomplete, 9-23
 - open database, 9-22
 - using backup control file, 9-20
- recovery catalog, 4-13
 - backing up, 6-30
 - changing availability of backups and copies, 6-8
 - creating a separate database, 5-5
 - crosschecking, 6-9
 - deciding whether to use, 5-4
 - losing control files when not using a, 4-16
 - managing the size of, 6-26
 - operating with, 4-13
 - operating without, 4-15
 - querying, 4-20, 7-2
 - recovering lost or damaged, 6-33
 - registering databases, 6-5
 - removing records, 6-16
 - resynchronizing, 6-23
 - schema, 6-2
 - snapshot control file, 4-14
 - updating after schema changes, 6-25
 - upgrading, 6-34
- registering databases, 6-6
- reports, 7-5
 - database schema, 7-8
 - objects needing a backup, 7-5
 - obsolete backups, 7-6, 7-8
- resetting database information, 6-7
- restoring, 9-2
 - archived redo logs, 9-13
 - control files to new location, 9-11, 9-12
 - database to default location, 9-3
 - restoring datafiles using, 4-47
 - sample scripts, 5-21
 - sbttest program, 4-19, 5-21
 - setting time parameters, 5-2
 - snapshot control file location, 5-3
 - stored scripts, 4-10
 - syntax conventions, 11-2
 - tablespace point-in-time recovery, 4-51
 - types of backups, 4-45
 - user tags for backups, 4-46
 - using RMAN commands, 4-5
- recovery sets
 - containing whole tables, A-6
 - copying to primary database, B-13
 - for RMAN TSPITR, A-4
 - importing into primary database, B-13
- recovery strategy
 - overview, 1-17 to 1-25
- RECOVERY_PARALLELISM initialization parameter, 14-19
- redo log files, 1-8
 - "fuzzy" data in backups and, 3-17
 - archived, 2-15
 - advantages of, 2-14
 - contents of, 2-15
 - clearing, 2-14
 - files named in control file, 2-3
 - groups, 2-13
 - members, 2-13
 - listing files for backup, 13-2
 - members, 2-13
 - mirrored
 - log switches and, 2-14
 - multiplexed, 2-12
 - diagrammed, 2-13
 - groups, 2-13
 - if all inaccessible, 2-14
 - if some members inaccessible, 2-14
 - purpose of, 1-9
 - naming, 14-10
 - overview, 1-8
- register command (RMAN), 6-5, 6-6, 11-93
- registering

- archived redo logs, 6-5
- datafiles, 6-5
- release channel command (RMAN), 11-95
 - releasing a maintenance channel, 11-96
- remote file server (RFS) process, 16-9
- removing records from the recovery catalog, 6-16
- RENAME DATABASE clause
 - ALTER DATABASE command, 9-9
- REOPEN option
 - LOG_ARCHIVE_DEST_n initialization parameter, 16-11
- replace script command (RMAN), 11-97
- replicate command (RMAN), 9-11, 11-100
- report command (RMAN), 4-22, 11-102
 - need backup option, 7-5
- reports, generating, 7-2, 7-5
 - database schema, 7-8
 - objects needing a backup, 7-5
 - obsolete backups, 7-6, 7-8
 - scenarios, 7-10, 7-11, 7-12
 - unrecoverable backups, 7-6, 7-8
- reset database command (RMAN), 6-7, 11-110
 - incarnation option, 6-7
- RESETLOGS option
 - ALTER DATABASE command, 14-17, 14-18, 14-33
 - backing up control file, 13-14
 - database backups after using, 3-19
 - multiple timelines for redo logs, 3-23
 - recovery of database after using, 14-35
- restore command (RMAN), 9-17, 11-112
- restore constraints, 4-48
- restore validation, 6-18
- restoring
 - archived redo logs using RMAN, 9-13
 - backup file selection, 4-48
 - backups of online redo logs, 3-23
 - database using RMAN, 9-3
 - datafiles, 4-47
 - datafiles using RMAN, 9-2
 - testing, 6-18
 - whole database backups, 14-17
- RESUME option
 - ALTER SYSTEM command, 13-10
- resuming recovery after interruption, 14-16

- resync catalog command (RMAN), 4-14, 6-23, 11-118
 - from controlfilecopy option, 6-33
- resynchronizing the recovery catalog, 4-14, 6-23
- RFS
 - See remote file server
- RMAN
 - See Recovery Manager
- rman command (RMAN), 11-121
- rollback segments, 1-8
 - and RMAN TSPITR, A-6
- rolling back, 14-2
- rolling forward, 14-2
- run command (RMAN), 11-121, 11-124

S

- sample scripts
 - RMAN, 5-21
- sbttest program, 4-19, 5-21
- scenarios, Recovery Manager
 - backing up archived redo logs, 8-19
 - cataloging O/S copies, 8-25
 - deleting obsolete backups and copies, 7-11
 - duplexing backup sets, 8-22
 - handling backup errors, 8-26
 - incremental backups, 8-21
 - incremental cumulative backups, 8-22
 - listing backups and copies, 7-10
 - listing obsolete backups and copies, 7-10
 - maintaining backups and copies, 8-26
 - NOARCHIVELOG backups, 8-23
 - OPS backups, 8-25
 - parallelization of backups, 8-23
 - recovering pre-resetlogs backup, 9-37, 9-38
 - recovery after total media failure, 9-35
 - reporting database schema, 7-12
 - reporting datafiles needing backups, 7-10, 8-15
 - reporting obsolete backups, 7-11
 - reporting unrecoverable datafiles, 7-11
 - restoring when databases share name, 9-31
 - setting size of backup sets, 8-16
- schema changes
 - updating recovery catalog, 6-25
- SCN

- See system change number
- SCN (system change number)
 - definition, 1-9
- scripts
 - CATRMAN.SQL, 6-5
- send command (RMAN), 5-20, 11-127
- SET command
 - AUTORECOVERY option, 14-12
 - LOGSOURCE variable, 14-8, 14-12
- set command (RMAN), 11-129
 - duplex parameter, 8-22
 - executed within run, 11-133
 - maxcorrupt for datafile option, 8-26
 - newname parameter, 4-53, 9-9
- SHUTDOWN command
 - ABORT option, 13-7, 14-17, 14-18, 14-26, 15-12
 - consistent whole database backups, 3-3
- shutdown command (RMAN), 11-137
- size of backup sets, setting, 4-31
- skip offline option (RMAN)
 - backup command, 8-15
- skip readonly option (RMAN)
 - backup command, 8-15
- snapshot control files, 4-14
 - specifying location, 5-3
- snapshots
 - distributed database recovery, 15-17
 - media recovery and, 15-17
- specifying destinations
 - for archived redo logs, 2-19
- splitting mirrors
 - suspend/resume mode, 13-9
- sql command (RMAN), 11-140
- SQL commands
 - applying log files, 14-14
 - for recovering tablespace, 14-14
- SQL*Plus
 - applying log files, 14-12
- stand-alone Recovery Manager commands, 4-8
- standby databases, 16-2 to 16-29
 - advantages, 16-2
 - altering control files, 16-25
 - archived redo logs
 - transmitting, 16-7
 - control files
 - refreshing, 16-29
 - creating, 16-2
 - procedures, 16-3
 - datafiles
 - renaming, 16-24
 - taking offline, 16-27
 - definition, 16-2
 - direct path operations, 16-27
 - effect of altering primary database, 16-22
 - initialization parameters, 16-25
 - maintaining, 16-6
 - manual recovery mode
 - procedures, 16-6
 - standby database, 16-5
 - modes, 16-5
 - read-only mode, 16-5, 16-19
 - redo log files
 - altering, 16-24
 - sustained recovery mode, 16-5
 - transmitting archived redo logs, 16-7
- STANDBY_ARCHIVE_DEST initialization
 - parameter, 16-26
- STARTUP command
 - MOUNT option, 14-28, 14-30
- startup command (RMAN), 11-142
- statement failures, 1-12
 - definition, 1-12
- stored scripts
 - creating, 6-20
 - deleting, 6-22
 - executing, 6-21
 - listing all, 6-22
 - managing, 6-20
 - printing, 6-22
 - Recovery Manager, 4-10
 - replacing, 6-21
- strategies
 - backup, 3-15 to 3-24
 - backups
 - ARCHIVELOG mode, 3-16
 - NOARCHIVELOG mode, 3-16
 - recovery
 - deciding what to recovery, 3-28
 - determining when media recovery is necessary, 3-30

- SUSPEND option
 - ALTER SYSTEM command, 13-9
- suspending a database, 13-9
- suspend/resume mode, 13-9
- sustained recovery mode, 16-5
- switch command (RMAN), 11-144
- syntax conventions
 - Recovery Manager, 11-2
- syntax diagrams
 - explanation of, 11-2
 - keywords, 11-3
 - parameters, 11-3
- system change number (SCN)
 - definition, 1-9
 - use in distributed recovery, 15-17
- system failures, 1-12
- SYSTEM tablespace
 - recovery of, 14-23
- system time, changing
 - effect on recovery, 14-26
- using RMAN, 4-51
 - introduction, A-2
 - performing, A-10
 - planning for, A-4
 - preparing the auxiliary instance, A-7
 - recovery sets containing whole tables, A-6
 - restrictions, A-4
 - tuning considerations, A-13
- tablespaces
 - backing up, 3-19
 - frequency, 3-19
 - offline, 13-10
 - online, 13-8
 - read-only
 - backing up, 8-15
 - effect on recovery, 14-5
 - recovering offline in open database, 14-23
- test databases, creating, 4-52
- testing
 - backup strategies, 3-24
 - media manager, 4-19
- time format
 - RECOVER DATABASE UNTIL TIME
 - statement, 14-31
- time parameters
 - setting for Recovery Manager use, 5-2
- time-based recovery, 14-29 to 14-31
 - coordinated in distributed databases, 15-16
 - definition, 3-28
- TNSNAMES.ORA file, 16-10
- trace files
 - control file backups to, 13-12
 - log writer process and, 2-13
- transactions
 - rollback, 1-8
 - uncommitted, 1-8
- transmitting archived redo logs, 2-21, 16-7
 - in normal transmission mode, 2-21
- transportable tablespaces
 - recovery, 15-4
 - TSPITR and, B-3
- troubleshooting the media manager, 5-21
- TS_PITR_CHECK view, A-6
- TSPITR. See tablespace point-in-time recovery.

T

- tablespace backups
 - using RMAN, 8-4, 8-5, 8-6
- tablespace point-in-time recovery, B-25
 - advantages, B-2
 - clone database, B-4
 - different methods, B-25
 - introduction, B-2
 - limitations, B-5
 - methods, B-3
 - O/S, B-3
 - performing, B-1 to B-26
 - planning for, B-4
 - preparing, B-6
 - procedures for using transportable tablespace
 - feature, B-25
 - requirements, B-6
 - restrictions, B-5
 - terminology, B-4
 - transportable tablespace method, B-3, B-25
 - tuning considerations, B-23
 - backup control files, B-24
 - recovery set location, B-23

U

- unavailable option (RMAN)
 - change command, 6-8
- unrecoverable objects
 - and RECOVER operation, 14-5
- UNRECOVERABLE operations
 - backups after, 16-28
 - performing backups after, 3-19
- unregistering a database from the recovery catalog, 6-6
- until clause (RMAN), 11-42, 11-146
- UNTIL TIME option
 - RECOVER command, 14-31
- updating records in recovery catalog, 6-13
- upgrade catalog command (RMAN), 11-148
- upgrading the recovery catalog, 6-34
- user errors
 - database failures, 1-12
 - definition, 1-12
 - recovery from, 15-14
- user tags, 4-46
- user-created backup files
 - cataloging, 6-28
- USING BACKUP CONTROLFILE option
 - RECOVER command, 14-28
- utilities
 - O/S, using to make copies, 8-25
 - Recovery Manager, 4-2

V

- V\$ARCHIVE view, 2-4, 2-12, 2-18
- V\$BACKUP view, 13-7
- V\$BACKUP_CORRUPTION view, 4-36
- V\$DATABASE view, 2-4, 2-18
- V\$DATAFILE view, 14-31, 16-28
 - listing files for backups, 13-2
- V\$LOG view, 2-4, 2-12, 2-18
 - displaying archiving status, 2-4, 2-12, 2-18
- V\$LOG_HISTORY view, 14-7
- V\$LOGFILE view, 15-6, 15-7
 - listing files for backups, 13-2
- V\$LONGOPS view, 4-36
- V\$PROXY_ARCHIVEDLOG view, 4-19

- V\$PROXY_DATAFILE view, 4-19
- V\$RECOVER_FILE view, 1-24, 14-3
 - determining which files to recover, 3-27
- V\$RECOVERY_LOG view, 14-7
- validate command (RMAN), 11-150
- validation of restore, 6-18
- views
 - data dictionary, 13-6, 13-10
 - DBA_DATA_FILES, 13-6, 13-10
 - recovery catalog, 12-1
 - TS_PITR_CHECK, A-6
 - V\$ARCHIVE, 2-4, 2-12, 2-18
 - V\$BACKUP, 13-7
 - V\$BACKUP_CORRUPTION, 4-36
 - V\$DATABASE, 2-4, 2-18
 - V\$DATAFILE, 13-2, 14-31
 - V\$LOG, 2-4, 2-12, 2-18
 - V\$LOG_HISTORY, 14-7
 - V\$LOGFILE, 13-2, 15-6, 15-7
 - V\$LONGOPS, 4-36
 - V\$PROXY_ARCHIVEDLOG, 4-19
 - V\$PROXY_DATAFILE, 4-19
 - V\$RECOVER_FILE, 3-27
 - V\$RECOVERFILE, 14-3
 - V\$RECOVERY_LOG, 14-7

W

- warning
 - archiving mode for first backup, 3-16
 - consistency and Export backups, 13-16
- whole database backups, 13-3 to 13-5
 - ARCHIVELOG mode, 13-3
 - consistent, 3-2
 - backup control files and, 3-3
 - using SHUTDOWN ABORT command, 3-3
 - definition, 3-2
 - inconsistent, 3-4
 - NOARCHIVELOG mode, 13-3
 - preparing to take, 13-4
 - restoring from, 14-17
 - using RMAN, 8-3

