

Oracle® Database

Backup and Recovery Basics

10g Release 2 (10.2)

B14192-03

November 2005

An introduction to the basics of backup and recovery of Oracle databases, focusing on the use of Recovery Manager for common backup and recovery tasks.

Oracle Database Backup and Recovery Basics, 10g Release 2 (10.2)

B14192-03

Copyright © 2003, 2005, Oracle. All rights reserved.

Primary Author: Antonio Romero

Contributing Author: Lance Ashdown

Contributors: Tammy Bednar, Anand Beldalker, Timothy Chien, Raymond Guzman, Alex Hwang, Ashok Joshi, J. William Lee, Valarie Moore, Muthu Olagappan, Samitha Samaranyake, Francisco Sanchez, Steven Wertheimer, Wanli Yang

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be 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.

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

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

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

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	xiii
Audience	xiii
Documentation Accessibility	xiii
Related Documentation	xiv
Conventions	xiv
1 Backup and Recovery Overview	
What is Backup and Recovery?	1-1
Physical Backups and Logical Backups	1-1
Errors and Failures Requiring Recovery from Backup.....	1-2
Understanding User Error	1-2
Understanding Media Failure.....	1-2
Oracle Backup and Recovery Solutions: RMAN and User-Managed Backup.....	1-2
Backup and Recovery: Basic Concepts	1-3
Physical Database Structures Used in Recovering Data.....	1-3
Datafiles and Data Blocks	1-3
Redo Logs	1-4
Control Files.....	1-4
Undo Segments	1-5
The Database Recovery Process: Basic Concepts	1-5
Forms of Data Recovery	1-6
Datafile Media Recovery: Restore Datafiles, Apply Redo	1-6
Complete, Incomplete and Point-In-Time Recovery	1-7
Automatic Recovery After Instance Failure: Crash Recovery	1-8
Backup and Recovery with RMAN	1-8
Files That RMAN Can Back Up	1-9
RMAN Backup Destinations: Disk and Media Managers.....	1-10
Types of Oracle Database Backup under RMAN.....	1-10
About Consistent and Inconsistent Backups	1-10
About Full and Incremental Backups	1-10
About Image Copies, Backup Sets and Backup Pieces.....	1-10
Automatic Disk-Based Backup and Recovery: The Flash Recovery Area	1-11
Oracle Flashback Technology: Alternatives to Point-in-Time Recovery	1-11
About Restore Points	1-13
Matching Failures to Backup and Recovery Techniques	1-13

Responding to Media Failure	1-13
Responding to User Error	1-14
System Requirements for Backup and Recovery Methods.....	1-15
Feature Comparison of Backup Methods	1-15

2 Backup and Recovery Strategies

Data Recovery Strategy Determines Backup Strategy	2-1
Planning Data Recovery Strategy.....	2-3
Planning Responses to User Error: Point-in-Time Recovery and Flashback Features	2-3
Flashback Database.....	2-3
Creating Normal and Guaranteed Restore Points	2-3
Database Point-in-Time Recovery	2-4
Importing Lost Objects from Logical Backup	2-4
Planning a Response to Media Failure: Restore and Media Recovery	2-4
Example: Online Redo Log Recovery	2-4
Planning a Response to Datafile Block Corruption: Block Media Recovery	2-5
Planning Backup Strategy	2-5
Protecting Your Redundancy Set.....	2-5
Deciding Whether to Use a Flash Recovery Area	2-7
Deciding Between ARCHIVELOG and NOARCHIVELOG Mode	2-7
Implications of Running in NOARCHIVELOG Mode	2-7
Implications of Running in ARCHIVELOG Mode	2-8
Deciding Whether to Use Oracle Flashback Features and Restore Points.....	2-8
Choosing a Backup Retention Policy	2-8
Implementing Backup Retention Policy with RMAN	2-9
Recovery Window-Based Backup Retention Policy	2-9
Redundancy-Based Backup Retention Policy	2-10
Archiving Older Backups.....	2-10
Determining Backup Frequency	2-10
Performing Backups Before and After You Make Structural Changes	2-11
Scheduling Backups for Frequently-Updated Tablespace	2-11
Backing Up after NOLOGGING Operations	2-11
Exporting Data for Added Protection and Flexibility	2-12
Preventing the Backup of Online Redo Logs	2-12
Keeping Records of the Hardware and Software Configuration of the Server	2-12
Validating Your Data Recovery Strategy.....	2-13
Using BACKUP... VALIDATE	2-13
Validating RMAN Backups: VALIDATE and RESTORE VALIDATE.....	2-14
Testing Your Database Restore and Recovery Procedures	2-14

3 Setting Up and Configuring Backup and Recovery

Overview of Interacting With the RMAN Client	3-1
Starting and Exiting RMAN	3-1
Setting Globalization Support Environment Variables for RMAN	3-2
Entering RMAN Commands at the Command Prompt.....	3-2
Using Command Files with RMAN	3-3
Checking Syntax of RMAN Commands and Command Files: CHECKSYNTAX	3-3

Checking RMAN Syntax at the Command Line: Example.....	3-4
Checking RMAN Syntax in Command Files: Example.....	3-4
Using RMAN to Start Up and Shut Down Databases	3-5
Connecting the RMAN Client to Databases	3-5
Types of Database Connections Used with RMAN.....	3-6
Authentication for Database Connections.....	3-6
Connecting to the Target Database from the Command Line.....	3-6
Connecting to the Target Database from the RMAN Prompt.....	3-7
Setting Up a Database for RMAN Backup	3-7
Persistent Configuration Settings: Controlling RMAN Behavior	3-8
Displaying Current RMAN Configuration Settings: SHOW.....	3-8
Restoring Default RMAN Configuration Settings: CONFIGURE... CLEAR.....	3-9
Configuring the Default Device Type for Backups.....	3-9
Configuring the Default Backup Type for Disk Backups.....	3-10
Configuring Compressed Backupsets as Default for Tape or Disk.....	3-10
Configuring Disk Devices and Channels.....	3-10
Configuring Tape Devices and Channels.....	3-11
Configuring Control File and Server Parameter File Autobackup	3-11
Configuring the Control File Autobackup Format	3-12
Overriding the Configured Control File Autobackup Format.....	3-12
Setting Up a Flash Recovery Area for RMAN	3-13
Choosing a Location for the Flash Recovery Area	3-13
Flash Recovery Area, Automatic Storage Management, and Oracle Managed Files	3-14
Files That Can Be Stored in the Flash Recovery Area.....	3-14
Planning the Size of the Flash Recovery Area	3-15
Setting Initialization Parameters for Size and Location of the Flash Recovery Area.....	3-15
Flash Recovery Area Size: DB_RECOVERY_FILE_DEST_SIZE	3-16
Flash Recovery Area Location: Initialization Parameter DB_RECOVERY_FILE_DEST	3-16
Sharing a Flash Recovery Area Among Multiple Databases.....	3-17
Restrictions on Initialization Parameters When Using Flash Recovery Area	3-17
Adding a Flash Recovery Area to an Existing Database.....	3-17
Using V\$RECOVERY_FILE_DEST and V\$FLASH_RECOVERY_AREA_USAGE.....	3-18
Disabling the Flash Recovery Area	3-18
Configuring the Backup Retention Policy	3-18
Configuring a Recovery Window-Based Retention Policy	3-19
Configuring a Redundancy-Based Retention Policy	3-19
Showing the Current Retention Policy	3-19
Disabling the Retention Policy	3-20
How Oracle Manages Disk Space in the Flash Recovery Area	3-20
When Files are Eligible for Deletion from the Flash Recovery Area.....	3-20
When Space is Not Available in the Flash Recovery Area	3-21
Configure Flash Recovery Area for Disk-Based Backups: Example.....	3-21
Create a Database with Multiplexed Files in the Flash Recovery Area: Scenario	3-22
Creating a Database with Only Archived Logs in the Flash Recovery Area: Scenario	3-24

4 Backing Up Databases Using RMAN

Overview of RMAN Backups	4-1
---------------------------------------	------------

Files That RMAN Can Back Up	4-2
About RMAN Backup Formats: Image Copies and Backup Sets.....	4-2
About Image Copies	4-2
About Backup Sets	4-2
About RMAN Full and Incremental Datafile Backups.....	4-3
Specifying Options Affecting Output of the RMAN BACKUP Command	4-4
Specifying Output Device Type for RMAN BACKUP	4-4
Specifying Image Copy or Backup Set Output for RMAN BACKUP to Disk.....	4-4
Specifying Output File Locations for RMAN BACKUP.....	4-4
Specifying Tags for RMAN BACKUP.....	4-5
Using Compressed Backupsets for RMAN Backup.....	4-6
Backing Up Database Files and Archived Logs with RMAN.....	4-7
Making Consistent and Inconsistent Backups with RMAN	4-7
Making Whole Database Backups with RMAN	4-7
Backing Up Individual Tablespaces with RMAN	4-8
Backing Up Individual Datafiles and Datafile Copies with RMAN.....	4-8
Backing Up Datafiles	4-8
Backing Up Datafile Copies.....	4-8
Backing Up Control Files with RMAN	4-9
Including the Current Control File in a Backup of Other Files	4-9
Backing Up the Current Control File Manually	4-9
Backing Up a Control File Copy	4-10
Backing Up Server Parameter Files with RMAN	4-10
Backing Up Archived Redo Logs with RMAN.....	4-10
Backing Up Archived Redo Log Files with BACKUP ARCHIVELOG.....	4-10
Automatic Online Redo Log Switches During Backups of Archived Logs	4-10
Using BACKUP ARCHIVELOG with DELETE INPUT or DELETE ALL INPUT	4-11
Backing Up Logs with BACKUP ... PLUS ARCHIVELOG	4-11
RMAN Incremental Backups	4-12
Incremental Backup Algorithm.....	4-12
Level 0 and Level 1 Incremental Backups	4-13
Differential Incremental Backups	4-13
Cumulative Incremental Backups	4-14
Basic Incremental Backup Strategy	4-15
Making Incremental Backups: BACKUP INCREMENTAL.....	4-16
Incrementally Updated Backups: Rolling Forward Image Copy Backups.....	4-16
Incrementally Updated Backups: A Basic Example.....	4-16
Incrementally Updated Backups: A One Week Example	4-18
Improving Incremental Backup Performance: Change Tracking	4-19
Enabling and Disabling Change Tracking	4-19
Checking Whether Change Tracking is Enabled.....	4-20
Moving the Change Tracking File	4-20
Estimating Size of the Change Tracking File on Disk.....	4-20
Using RMAN to Validate Database Files	4-21
Overview of Reporting on Backups and the RMAN Repository	4-21
Listing RMAN Backups, Archived Logs, and Database Incarnations	4-22
About RMAN Reports Generated by the LIST Command	4-23

Listing Backups	4-23
Listing Backups by Backup	4-23
Listing Backups by File	4-24
Listing Backups in Summary Mode	4-25
Listing Selected Backups	4-25
Listing Database Incarnations	4-27
Reporting on Backups and Database Schema	4-27
About Reports of RMAN Backups	4-28
Reporting on Files Needing a Backup Under a Retention Policy	4-29
Using RMAN REPORT NEED BACKUP with Different Retention Policies	4-29
Using RMAN REPORT NEED BACKUP with Tablespaces and Datafiles	4-29
Using REPORT NEED BACKUP with Backups onTape or Disk Only	4-30
Reporting on Datafiles Affected by Unrecoverable Operations	4-30
Reporting Obsolete Backups	4-30
Reporting on the Database Schema	4-31

5 Data Protection with Restore Points and Flashback Database

Restore Points and Flashback Database: Concepts	5-1
About Flashback Database.....	5-2
About the Flashback Database Window	5-3
About Normal Restore Points	5-3
Commands Supporting the Use of Restore Points.....	5-4
About Guaranteed Restore Points	5-4
Using Guaranteed Restore Points Instead of Storage Snapshots.....	5-4
About Logging for Flashback Database and Guaranteed Restore Points.....	5-4
Guaranteed Restore Points and Flash Recovery Area Space Usage	5-5
Logging for Guaranteed Restore Points With Flashback Logging Disabled	5-5
Logging for Flashback Database With Guaranteed Restore Points Defined	5-6
Using Normal and Guaranteed Restore Points	5-6
Requirements for Using Guaranteed Restore Points	5-6
Creating Normal and Guaranteed Restore Points	5-7
Listing Restore Points	5-7
Dropping Restore Points	5-7
Monitoring Space Usage For Guaranteed Restore Points	5-8
Setup and Maintenance for Oracle Flashback Database	5-8
Limitations of Flashback Database	5-9
Requirements for Enabling Flashback Database	5-9
Enabling Logging for Flashback Database	5-9
Sizing the Flash Recovery Area to Include Flashback Logs.....	5-10
Estimating Disk Space Requirements for Flashback Database Logs.....	5-10
Managing Space For Flashback Logs in the Flash Recovery Area.....	5-11
Rules for Retention and Deletion of Flashback Logs	5-11
Determining the Current Window for Flashback Database	5-11
Performance Tuning for Flashback Database	5-12
Monitoring Flashback Database Performance Impact.....	5-13
Flashback Writer (RVWR) Behavior With I/O Errors	5-13

6 Performing Complete Restore and Recovery of Databases

Database Restore and Recovery with RMAN: Overview	6-1
Scope and Limitations of this Chapter	6-2
Restore and Recovery with Enterprise Manager	6-2
Basic Database Restore and Recovery Scenarios	6-3
Restore and Recovery of a Whole Database: Scenario	6-3
Recovery of Databases with Read-Only Tablespaces	6-4
Re-Creation of Temporary Tablespaces in Whole Database Restore and Recovery	6-4
Restore and Complete Recovery of Individual Tablespaces or Datafiles: Scenario	6-5
Preparing and Planning Database Restore and Recovery	6-5
Database Restore and Recovery Procedure: Outline	6-6
Determining Which Database Files to Restore or Recover	6-6
Recognizing a Lost Control File	6-6
Identifying Datafiles Requiring Media Recovery	6-6
Recovery of Read-Only Tablespaces	6-8
Determining your DBID	6-8
Previewing Backups Used in Restore Operations: RESTORE PREVIEW	6-9
Using RESTORE... PREVIEW	6-9
Using RESTORE... PREVIEW SUMMARY	6-9
Using RESTORE... PREVIEW RECALL	6-10
Validating the Restore of Backups: RESTORE VALIDATE and VALIDATE BACKUPSET	6-11
Validating Restore from Backup with RESTORE ... VALIDATE	6-12
Validating Backup Sets with VALIDATE BACKUPSET	6-12
RMAN RESTORE: Restoring Lost Database Files from Backup	6-13
Restoring the Control File from Backup	6-13
Default Destination for Restore of the Control File	6-13
Restore of the Control File from Control File Autobackup	6-14
Restore of the Control File When Using a Flash Recovery Area	6-14
Restoring a Control File When Using a Recovery Catalog	6-14
Restore of the Control File From a Known Location	6-15
Restore of the Control File to a New Location.....	6-15
Limitations When Using a Backup Control File.....	6-15
Restoring the Server Parameter File (SPFILE) from Backup	6-15
Restore of the SPFILE from the Control File Autobackup	6-16
Creating a Client-Side Initialization Parameter File (PFILE) with RMAN.....	6-17
Restoring and Recovering Datafiles and Tablespaces	6-17
Restoring Datafiles from Backup to a New Location	6-17
Performing Media Recovery of a Restored Database, Tablespace or Datafile.....	6-18
Restore and Recover of a Single Datafile to a New Location:Example.....	6-19
Restoring Archived Redo Logs from Backup	6-19
Restoring Archived Redo Logs to a New Location.....	6-20
Restoring Archived Redo Logs to Multiple Locations	6-20

7 Performing Flashback and Database Point-in-Time Recovery

About Point-in-Time Recovery and Flashback Features	7-1
About Database Point-in-Time Recovery	7-1
Oracle Flashback Technology:Alternatives to Point-in-Time Recovery	7-2

Oracle Flashback Query: Recovering at the Row Level	7-3
Oracle Flashback Table: Returning Individual Tables to Past States	7-4
Prerequisites for Using Flashback Table.....	7-4
Performing Flashback Table	7-5
Oracle Flashback Drop: Undo a DROP TABLE Operation	7-5
What is the Recycle Bin?	7-6
How Tables and Other Objects Are Placed in the Recycle Bin.....	7-6
Naming Convention for Objects in the Recycle Bin.....	7-7
Enabling and Disabling the Recycle Bin	7-7
Viewing and Querying Objects in the Recycle Bin	7-8
Recycle Bin Capacity and Space Pressure	7-9
Understanding Space Pressure	7-9
How the Database Responds to Space Pressure.....	7-9
Recycle Bin Objects and Segments	7-9
Performing Flashback Drop on Tables in the Recycle Bin	7-10
Flashback Drop of Multiple Objects With the Same Original Name	7-10
Purging Objects from the Recycle Bin.....	7-11
PURGE TABLE: Purging a Table and Dependent Objects	7-11
PURGE INDEX: Freeing Space in the Recycle Bin.....	7-11
PURGE TABLESPACE: Purging All Dropped Objects from a Tablespace	7-12
PURGE RECYCLEBIN: Purging All Objects in a User's Recycle Bin.....	7-12
PURGE DBA_RECYCLEBIN: Purging All Recycle Bin Objects.....	7-12
Dropping a Tablespace, Cluster, User or Type and the Recycle Bin.....	7-12
Privileges and Security for Flashback Drop.....	7-12
Limitations and Restrictions on Flashback Drop	7-13
Reversing Database Changes with Flashback Database	7-13
Performing Flashback Database: Scenario.....	7-14
Options After a Successful Flashback Database Operation.....	7-15
Options After Flashback Database to the Wrong Time.....	7-16
Flashback Database and Ambiguous SCNs Across Incarnations.....	7-16
Performing Flashback Database to a Guaranteed Restore Point	7-17
Performing Flashback Database to Undo an OPEN RESETLOGS.....	7-17
Flashback Database Across OPEN RESETLOGS With Standby Databases	7-18
Flashback Database To The Right of Open Resetlogs: Example	7-18
Performing Database Point-in-Time Recovery	7-19
Requirements for Database Point-in-Time Recovery.....	7-19
Point-in-Time Recovery and Database Incarnations: Concepts	7-19
Understanding Parent, Ancestor and Sibling Database Incarnations.....	7-19
Incarnation History of a Database: Example	7-20
Sibling Incarnations, Ambiguous SCNs and RESET DATABASE INCARNATION	7-21
Database Incarnations and Orphaned Backups	7-21
Uses of Orphaned Backups	7-22
Preparing for Database Point-in-Time Recovery.....	7-22
Database Point-in-Time Recovery Within the Current Incarnation	7-22
Using a Time Expression for Database Point-in-Time Recovery	7-23
Options After Database Point-in-Time Recovery.....	7-24

8 Recovery Manager Maintenance Tasks

Managing the RMAN Repository Using Only the Control File	8-1
Backing Up and Restoring the Control File.....	8-1
Monitoring the Overwriting of Control File Records	8-2
Managing the Overwriting of Control File Records	8-2
Interaction of Flash Recovery Area and CONTROL_FILE_RECORD_KEEP_TIME	8-3
Using CROSSCHECK to Update the RMAN Repository	8-4
About RMAN Crosschecks.....	8-4
Basic Use of CROSSCHECK with Backup Sets and Image Copies	8-5
Crosschecking Specific Backup Sets and Copies	8-5
Crosschecking Backups of Specific Database Files	8-6
Limiting RMAN CROSSCHECK to a Backups Since a Specific Time.....	8-6
Deleting Backups	8-6
Deleting Specified Backups	8-6
Deleting Expired RMAN Backups after CROSSCHECK	8-7
Using DELETE FORCE With RMAN Backups.....	8-8
Deleting Obsolete RMAN Backups Based on Retention Policies.....	8-8
DELETE OBSOLETE Behavior When KEEP UNTIL Time Expires.....	8-9
Using Multiple RMAN Channels for Maintenance Operations	8-9
About Allocating Multiple RMAN Channels for Maintenance Commands.....	8-9
How RMAN Crosschecks and Deletes on Multiple Channels.....	8-9
Crosschecking Disk and Tape Channels with One Command: Example.....	8-10
Crosschecking on Multiple Oracle Real Application Cluster Nodes: Example.....	8-11
Deleting on Disk and Tape Channels with One DELETE Command: Example.....	8-11
Releasing Multiple Channels: Example	8-12
Deleting a Database with RMAN	8-12
Changing the Status of a Backup Record	8-13
Marking a Backup AVAILABLE or UNAVAILABLE	8-13
Exempting a Long-Term Backup from the Retention Policy.....	8-13
Cataloging Archived Logs and User-Managed Copies	8-14
About Cataloging Archived Logs and User-Managed Copies.....	8-14
Cataloging User-Managed Datafile Copies.....	8-15
Cataloging Backup Pieces	8-16
Cataloging All Files in a Disk Location.....	8-16
Cataloging Flash Recovery Area Contents.....	8-17
Uncataloging RMAN Records	8-17
About Uncataloging RMAN Records	8-17
Removing Records for Files Deleted with Operating System Utilities	8-17
Flash Recovery Area Maintenance	8-18
Resolving a Full Flash Recovery Area.....	8-18
Changing the Flash Recovery Area to a New Location.....	8-19
Flash Recovery Area Behavior When Instance Crashes During File Creation	8-19
Backing Up to the Flash Recovery Area: Basic Scenarios	A-1
Scripting Disk-Only Backups	A-1
Backup Scripts When Few Data Blocks Change.....	A-2

Initial Setup.....	A-2
Daily Script	A-2
Backup Scripts When Blocks Change Frequently	A-5
Backup Scripts When a Moderate Number of Blocks Change Weekly	A-5
Initial Setup.....	A-6
Weekly Script	A-6
Backing Up to the Flash Recovery Area and to Tape: Basic Scenarios.....	A-7
Configuring the RMAN Environment for Disk and Tape Backups	A-8
Writing Backup Scripts for Disk and Tape Scenarios	A-8
Backup Scripts When Few Data Blocks Change.....	A-8
Initial Setup.....	A-8
Daily Script	A-8
Backup Scripts When Many Blocks Change	A-9
Initial Setup.....	A-10
Weekly Scripts.....	A-10
Daily Script	A-10
Backup Scripts When Blocks Change Moderately	A-10
Initial Setup.....	A-11
Weekly Script	A-11
Daily Script	A-11
Backup Scripts When Not Enough Disk Space for a Database Backup	A-13
Weekly Script	A-13
Daily Script	A-13

Glossary

Index

Preface

This guide provides a basic conceptual overview of Oracle database backup and recovery.

This preface contains these topics:

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

Audience

This manual is intended for database administrators who perform backup and recovery of an Oracle database server using Recovery Manager (RMAN).

To use this document, you need to know the following:

- Relational database concepts and basic database administration as described in *Oracle Database Concepts* and *Oracle Database Administrator's Guide*
- The operating system environment under which you are running the Oracle database

The conceptual material in [Chapter 1, "Backup and Recovery Overview"](#) is of interest for all users responsible for backup and recovery, not just those using RMAN. The remainder of the book covers techniques for backup, recovery and maintenance using Recovery Manager. Users planning to manage backup and recovery without RMAN should review the conceptual material in [Chapter 1](#) and then turn to *Oracle Database Backup and Recovery Advanced User's Guide* for more conceptual material in Part I, and several chapters on user-managed backup and recovery techniques in Part IV.

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Database Backup and Recovery Quick Start Guide*
- *Oracle Database Backup and Recovery Advanced User's Guide*
- *Oracle Database Backup and Recovery Reference*
- *Oracle Database SQL Reference*
- *Oracle Database Utilities*

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

Conventions

The following text conventions are used in this document:

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

Backup and Recovery Overview

This chapter provides a general overview of backup and recovery concepts, the files in an Oracle database related to backup and recovery, and the tools available for making backups of your database, recovering from data loss or other error, and maintaining records of your backups.

This chapter includes the following topics:

- [What is Backup and Recovery?](#)
- [Backup and Recovery: Basic Concepts](#)
- [The Database Recovery Process: Basic Concepts](#)
- [Forms of Data Recovery](#)
- [Backup and Recovery with RMAN](#)
- [Automatic Disk-Based Backup and Recovery: The Flash Recovery Area](#)
- [Oracle Flashback Technology: Alternatives to Point-in-Time Recovery](#)
- [Matching Failures to Backup and Recovery Techniques](#)
- [System Requirements for Backup and Recovery Methods](#)
- [Feature Comparison of Backup Methods](#)

What is Backup and Recovery?

In general, **backup and recovery** refers to the various strategies and procedures involved in protecting your database against data loss and reconstructing the database after any kind of data loss.

Physical Backups and Logical Backups

A **backup** is a copy of data from your database that can be used to reconstruct that data. Backups can be divided into **physical backups** and **logical backups**.

Physical backups are backups of the physical files used in storing and recovering your database, such as datafiles, control files, and archived redo logs. Ultimately, every physical backup is a copy of files storing database information to some other location, whether on disk or some offline storage such as tape.

Logical backups contain logical data (for example, tables or stored procedures) exported from a database with an Oracle export utility and stored in a binary file, for later re-importing into a database using the corresponding Oracle import utility.

See also: *Oracle Database Utilities* for more details about using Oracle export and import utilities for logical backups

Physical backups are the foundation of any sound backup and recovery strategy. Logical backups are a useful supplement to physical backups in many circumstances but are not sufficient protection against data loss without physical backups.

Unless otherwise specified, the term "backup" as used in the backup and recovery documentation refers to physical backups, and to **back up** part or all of your database is to take some kind of physical backup. The focus in the backup and recovery documentation set will be almost exclusively on physical backups.

Errors and Failures Requiring Recovery from Backup

While there are several types of problem that can halt the normal operation of an Oracle database or affect database I/O operations, only two typically require DBA intervention and media recovery: media failure, and user errors.

Other failures may require DBA intervention to restart the database (after an instance failure) or allocate more disk space (after statement failure due to, for instance, a full datafile) but these situations will not generally cause data loss or require recovery from backup.

Understanding User Error

User errors occur when, either due to an error in application logic or a manual mis-step, data in your database is changed or deleted incorrectly. Data loss due to user error includes such missteps as dropping important tables or deleting or changing the contents of a table. While user training and careful management of privileges can prevent most user errors, your backup strategy determines how gracefully you recover the lost data when user error does cause data loss.

Understanding Media Failure

A **media failure** is the failure of a read or write of a disk file required to run the database, due to a physical problem with the disk such as a head crash. Any database file can be vulnerable to a media failure.

The appropriate recovery technique following a media failure depends on the files affected and the types of backup available.

Oracle Backup and Recovery Solutions: RMAN and User-Managed Backup

For performing backup and recovery based on physical backups, you have two solutions available:

- **Recovery Manager**, a tool (with command-line client and Enterprise Manager GUI interfaces) that integrates with sessions running on the Oracle server to perform a range of backup and recovery activities, as well as maintaining a repository of historical data about your backups
- The traditional **user-managed backup and recovery**, where you directly manage the files that make up your database with a mixture of host operating system commands and SQL*Plus backup and recovery-related capabilities

Both methods are supported by Oracle Corporation and are fully documented. Recovery Manager is, however, the preferred solution for database backup and recovery. It can perform the same types of backup and recovery available through user-managed methods more easily, provides a common interface for backup tasks

across different host operating systems, and offers a number of backup techniques not available through user-managed methods.

Most of the backup and recovery documentation set will focus on RMAN-based backup and recovery. User-managed backup and recovery techniques are covered in the later chapters of *Oracle Database Backup and Recovery Advanced User's Guide*.

Whether you use RMAN or user-managed methods, you can supplement your physical backups with logical backups of schema objects made using data export utilities. Data thus saved can later be imported to re-create this data after restore and recovery. However, logical backups are for the most part beyond the scope of the backup and recovery documentation.

Backup and Recovery: Basic Concepts

The physical structures of the database and the role each plays in the database recovery process determine the forms of backup and recovery available through user-managed techniques and through RMAN.

Physical Database Structures Used in Recovering Data

The files and other structures that make up an Oracle database store data and safeguard it against possible failures. This discussion introduces each of the physical structures that make up an Oracle database and their role in the reconstruction of a database from backup. This section contains these topics:

- [Datafiles and Data Blocks](#)
- [Redo Logs](#)
- [Undo Segments](#)
- [Control Files](#)

Datafiles and Data Blocks

An Oracle database consists of one or more logical storage units called tablespaces. Each tablespace in an Oracle database consists of one or more files called datafiles, physical files under the host operating system which collectively contain the data stored in the tablespace. The simplest Oracle database would have one tablespace, stored in one datafile.

The database manages the storage space in the datafiles of a database in units called data blocks. Data blocks are the smallest units of storage that the database can use or allocate.

Modified or new data is not written to datafiles immediately. Updates are buffered in memory and written to datafiles at intervals. If a database has not gone through a normal shutdown (that is, if it is open, or exited abnormally, as in an instance failure or a SHUTDOWN ABORT) then there are typically changes in memory that have not been written to the datafiles. Datafiles that were restored from backup, or were not closed during a **consistent shutdown**, are typically not completely up to date.

Copies of the datafiles of a database are a critical part of any backup.

See also: *Oracle Database Concepts* for more detail about the structure and contents of datafiles and data blocks.

Redo Logs

Redo logs record all changes made to a database's data files. Each time data is changed in the database, that change is recorded in the **online redo log** first, before it is applied to the datafiles.

An Oracle database requires at least two **online redo log groups**, and in each group there is at least one **online redo log member**, an individual redo log file where the changes are recorded.

At intervals, the database rotates through the online redo log groups, storing changes in the **current online redo log**.

Because the redo log contains a record of all changes to the datafiles, if a backup copy of a datafile from some point in time and a complete set of redo logs from that time forward are available, the database can reapply changes recorded in the redo logs, in order to re-construct the datafile contents at any point between the backup time and the end of the last redo log. However, this is only possible if the redo log has been preserved.

Therefore, preserving the redo logs is a major part of most backup strategies. The first level of preserving the redo log is through a process called **archiving**. The database can copy online redo log groups that are not currently in use to one or more **archive locations** on disk, where they are collectively called the **archived redo log**. Individual files are referred to as **archived redo log files**. After a redo log file is archived, it can be backed up to other locations on disk or on tape, for long term storage and use in future recovery operations.

Without archived redo logs, your database backup and recovery options are severely limited. Your database must be taken offline before it can be backed up, and if you must restore your database from backup, the database contents are only available as of the time of the backup. Reconstructing the state of the database at a point in time between backups is impossible without the archived log.

See also: *Oracle Database Administrator's Guide* for more details about the online redo logs, *Oracle Database Administrator's Guide* for more details about archived redo logs, and "[Deciding Between ARCHIVELOG and NOARCHIVELOG Mode](#)" on page 2-7 for a discussion of the implications of archiving or discarding your redo log files.

Control Files

The control file contains the record of the physical structures of the database and their status. Several types of information stored in the control file are related to backup and recovery:

- Database information (RESETLOGS SCN and time stamp)
- Tablespace and datafile records (filenames, datafile checkpoints, read/write status, offline ranges)
- Information about redo threads (current online redo log)
- Log records (log sequence numbers, SCN range in each log)
- A record of past RMAN backups
- Information about corrupt datafile blocks

The recovery process for datafiles is in part guided by status information in the control file, such as the database checkpoints, current online redo log file, and the **datafile**

header checkpoints for the datafiles. Loss of the control file makes recovery from a data loss much more difficult.

See also: *Oracle Database Concepts* for more information about control files.

Undo Segments

In general, when data in a datafile is updated, "before images" of that data are written into **undo segments**. If a transaction is rolled back, this undo information can be used to restore the original datafile contents.

In the context of recovery, the undo information is used to undo the effects of uncommitted transactions, once all the datafile changes from the redo logs have been applied to the datafiles. The database is actually opened before the undo is applied.

You should not have to concern yourself with undo segments or manage them directly as part of your backup and recovery process.

See also: *Oracle Database Concepts* for detailed information about undo segments.

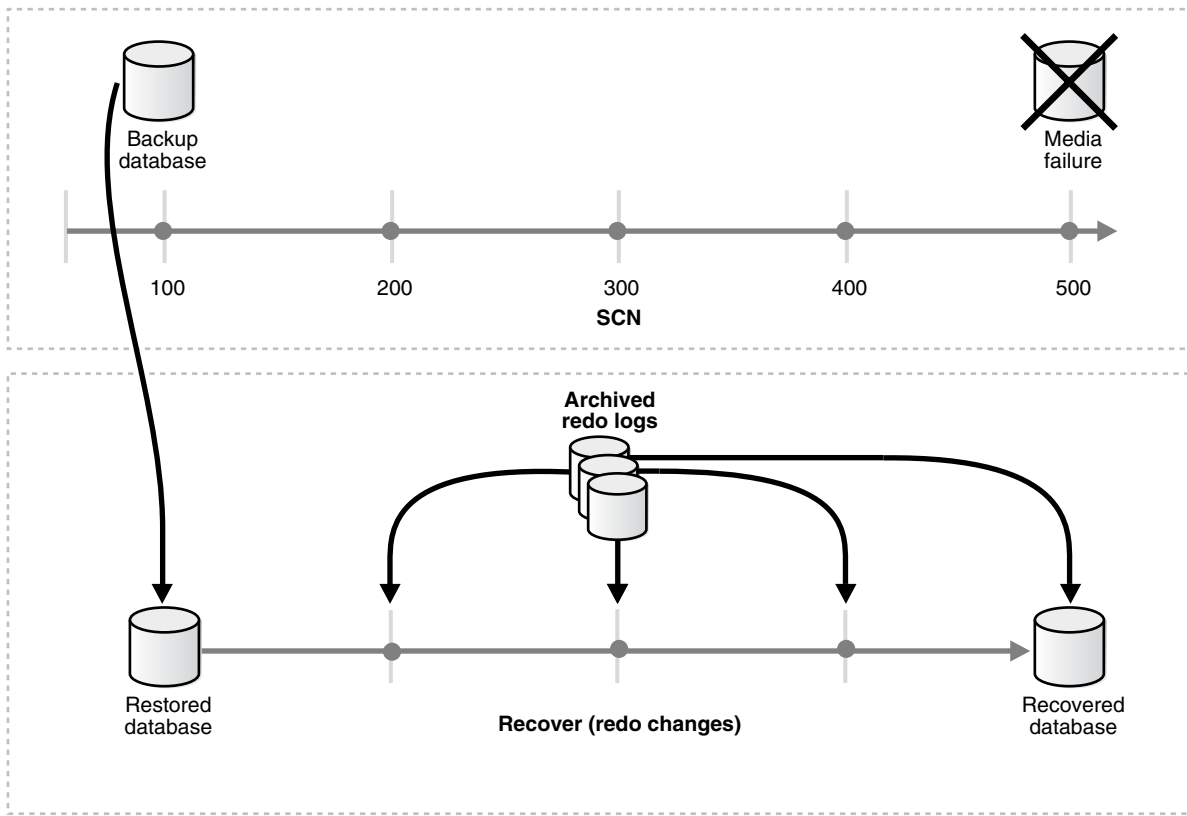
The Database Recovery Process: Basic Concepts

Reconstructing the contents of all or part of a database from a backup typically involves two phases: retrieving a copy of the datafile from a backup, and reapplying changes to the file since the backup from the archived and online redo logs, to bring the database to a desired SCN since the backup (usually, the present).

To **restore** a datafile or control file from backup is to retrieve the file onto disk from a backup location on tape, disk or other media, and make it available to the database server.

To **recover** a datafile (also called **performing recovery** on a datafile), is to take a restored copy of the datafile and apply to it changes recorded in the database's redo logs. To recover a whole database is to perform recovery on each of its datafiles.

[Figure 1-1](#) illustrates the basic principle of backing up, restoring, and recovering a database. Most of the data recovery procedures supported by the Oracle database are variations on the process described here.

Figure 1–1 Restoring and Recovering a Database

In this example a full backup of a database (copies of its datafiles and control file) is taken at SCN 100. Redo logs generated during the operation of the database capture all changes that occur between SCN 100 and SCN 500. Along the way, some logs fill and are archived. At SCN 500, the datafiles of the database are lost due to a media failure. The database is then returned to its transaction-consistent state at SCN 500, by restoring the datafiles from the backup taken at SCN 100, then applying the transactions captured in the archived and online redo logs and undoing the uncommitted transactions.

Forms of Data Recovery

The preceding scenario outlined the basics of the restore-and-recovery process. Several variants on this scenario are important to your backup and recovery work.

This section contains the following topics:

- [Datafile Media Recovery: Restore Datafiles, Apply Redo](#)
- [Complete, Incomplete and Point-In-Time Recovery](#)
- [Automatic Recovery After Instance Failure: Crash Recovery](#)

Datafile Media Recovery: Restore Datafiles, Apply Redo

Datafile media recovery (often simply called **media recovery**) is the most basic form of user-initiated data recovery. It can be used to recover from a lost or damaged current datafile, SPFILE or control file. It can also recover changes that were recorded in the redo logs but not in the datafiles for a tablespace that went offline without the

OFFLINE NORMAL option. Datafile media recovery can be performed whether you use Recovery Manager or user-managed backup and recovery. (For user-managed backup and recovery, it is in fact the main option available.)

The need to restore a datafile from backup is not detected automatically. The first step in performing media recovery is to manually restore the datafile by copying it from a backup. Once a datafile has been restored from backup, however, the database does automatically detect that this datafile is out of date and must undergo media recovery.

Several situations force you to perform media recovery:

- You restore a backup of a datafile.
- You restore a backup control file (even if all datafiles are current).
- A datafile is taken offline (either by you or automatically by the database) without the OFFLINE NORMAL option.

For a datafile to be available for media recovery, one of two things must be true:

- The database that the datafile belongs to must not be open;

or

- The specific datafile needing recovery must be offline, if the database is open.

A datafile that needs media recovery cannot be brought online until media recovery has been completed. A database cannot be opened if any of the online datafiles needs media recovery.

You can manage the expected duration of media recovery as part of your backup and recovery strategy. It is affected by, for example, the frequency of backups and parallel recovery parameters.

Complete, Incomplete and Point-In-Time Recovery

Complete recovery is recovering a database to the most recent point in time, without the loss of any committed transactions. Generally, the term "recovery" refers to complete recovery.

Occasionally, however, you need to return a database to its state at a past point in time. For example, to undo the effect of a user error, such as dropping or deleting the contents of a table, you may want to return the database to its contents before the delete occurred. In **incomplete recovery**, also known as **point-in-time recovery**, the goal is to restore the database to its state at some previous target SCN or time. Point-in-time recovery is one possible response to a data loss caused by, for instance, a user error or logical corruption that goes unnoticed for some time.

Point-in-time recovery is also your only option if you have to perform a recovery and discover that you are missing an archived log covering time between the backup you are restoring from and the target SCN for the recovery. Without the missing log, you have no record of the updates to your datafiles during that period. Your only choice is to recover the database from the point in time of the restored backup, as far as the unbroken series of archived logs permits, then perform an OPEN RESETLOGS and abandon all changes in or after the missing log. (If you discover that you have lost archived logs and your database is still up, you should do a full backup immediately.)

Note: If only one tablespace is affected by the data loss, you have the option of performing point-in-time recovery on that tablespace instead of the entire database. Tablespace point-in-time recovery (often abbreviated TSPITR) is an advanced technique documented in *Oracle Database Backup and Recovery Advanced User's Guide*.

Automatic Recovery After Instance Failure: Crash Recovery

The **crash recovery** process is a special form of recovery, which happens the first time an Oracle database instance is started after a crash (or `SHUTDOWN ABORT`). In crash recovery, the goal is to bring the datafiles to a transaction-consistent state, preserving all committed changes up to the point when the instance failed.

Like crash recovery, datafile media recovery is intended to restore database integrity. However, there are a number of important differences between the two:

- Media recovery must be explicitly invoked by a user. The database will not run media recovery on its own.
- Media recovery applies needed changes to datafiles that have been restored from backup, not to online datafiles left over after a crash.
- Media recovery must use archived logs as well as the online logs, to find changes reaching back to the time of the datafile backup.

Unlike the forms of recovery performed manually after a data loss, crash recovery uses only the online redo log files and current online datafiles, as left on disk after the instance failure. Archived logs are never used during crash recovery, and datafiles are never restored from backup.

The database applies any pending updates in the online redo logs to the online datafiles of your database. The result is that, whenever the database is restarted after a crash, the datafiles reflect all committed changes up to the moment when the haven't said failure occurred. (After the database opens, any changes that were part of uncommitted transactions at the time of the crash are rolled back.)

The duration of crash recovery is a function of the number of instances needing recovery, amount of redo generated in the redo threads of crashed instances since the last checkpoint, and user-configurable factors such as the number and size of redo log files, checkpoint frequency, and the parallel recovery setting. You can set parameters in the database server that can tune the duration of crash recovery. You can also tune checkpointing to optimize recovery time.

Note: Crash recovery in a Real Application Cluster (RAC) database takes place when all instances in the cluster have failed. The related process of **instance recovery** takes place when some but not all instances fail. For more information on crash and instance recovery in the context of RAC, refer to *Oracle Real Application Clusters Quick Start*.

Backup and Recovery with RMAN

As noted earlier, using RMAN gives you access to several data backup and recovery techniques and features not available at all with user-managed backup and recovery. The most noteworthy are:

- **Incremental backups**, which provide more compact backups (storing only changed blocks) and faster datafile media recovery (reducing the need to apply redo during datafile media recovery)
- **Block media recovery**, in which a datafile with only a small number of corrupt data blocks can be repaired without being taken offline or restored from backup
- **Unused block compression**, where RMAN can in some cases skip unused datafile blocks during backups
- **Binary compression**, which uses a compression mechanism integrated into the Oracle database server to reduce the size of backups
- **Encrypted backups**, which uses encryption capabilities integrated into the Oracle database to store backups in an encrypted format

A complete list of feature differences between RMAN and user-managed backup and recovery can be found in "[Feature Comparison of Backup Methods](#)" on page 1-15.

RMAN also reduces the administration work associated with your backup strategy. RMAN keeps an extensive record of metadata about backups, archived logs, and its own activities, known as the **RMAN repository**. In restore operations, RMAN can use this information to eliminate the need for you to identify backup files for use in restores in most situations. You can also generate reports of backup activity using the information in the repository.

Primary storage for RMAN repository information is in the control file of the production database. You can also set up an independent **recovery catalog**, a schema that stores RMAN repository information for one or many databases in a separate **recovery catalog database**.

The remainder of this book, *Oracle Database Backup and Recovery Basics*, focuses on using RMAN to implement your backup and recovery strategy.

Files That RMAN Can Back Up

RMAN can back up all database files needed for efficient recovery in the event of a failure. RMAN supports backing up the following types of files:

- Datafiles, and image copies of datafiles
- Control files, and image copies of control files
- Archived redo logs
- The current server parameter file
- Backup pieces, containing other backups created by RMAN

Note: Although the database depends on other types of files for operation, such as network configuration files, password files, and the contents of the Oracle home, these files cannot be backed up with RMAN. Likewise, some features of Oracle, such as external tables or the BFILE datatype, store data in files other than those listed here. RMAN cannot back up those files. You must use some non-RMAN backup solution for any files not in the preceding list.

RMAN Backup Destinations: Disk and Media Managers

RMAN can create and manage backups on disk and on tape, back up backups originally created on disk to tape, and restore database files from backups on disk or tape.

Devices used for tape backup are often referred to as SBT (System Backup to Tape) devices. RMAN interacts with SBT devices through software known as a media management layer, or media manager.

Types of Oracle Database Backup under RMAN

There are several ways of distinguishing among physical backups, according to the state the database was in when the backup was created, what parts of the database were actually backed up, and how the resulting backup was stored.

About Consistent and Inconsistent Backups

Physical backups can also be divided into **consistent** and **inconsistent** backups. Consistent backups are those created when the database is in a consistent state, that is, when all changes in the redo log have been applied to the datafiles. A database restored from a consistent backup can be opened immediately, without undergoing media recovery. However, a consistent backup can only be created after a consistent shutdown, that is, not after a crash or a `SHUTDOWN ABORT`.

For reasons of availability, the Oracle database is designed to work equally well with an inconsistent backup, a backup taken while the database is open. However, when a database is restored from an inconsistent backup, it must undergo media recovery, so that the database can apply any pending changes from the online and archived redo log before the database is opened again. Because archived logs are required for media recovery, using inconsistent backups requires that your database be run in `ARCHIVELOG` mode.

About Full and Incremental Backups

Full backups are backups which include datafiles in their entirety. Full backups can be created with Recovery Manager or with operating system-level file copy commands. Incremental backups are based on the idea of making copies only of changed data blocks in a data file. In recovery, extracting entire changed blocks from an incremental backup can substitute for application of redo for individual datafile updates during the time covered by the backup, shortening recovery times considerably. Incremental backups can only be created with RMAN.

See Also: ["About RMAN Full and Incremental Datafile Backups"](#) on page 4-3 for more details about the different ways to back up datafiles.

About Image Copies, Backup Sets and Backup Pieces

The results of an Oracle database backup created through RMAN can be either image copies or backup sets. An **image copy** is a bit-for-bit identical copy of a database file. RMAN can create image copy backups, although in the process, RMAN will check the contents for corruption, something that native operating-system file copy utilities cannot do. RMAN records image copies it creates in the RMAN repository, so that it can use them when restoring your database. Image copies can also be created using operating system commands such as `cp` in Unix or `COPY` in Windows.

Note: If you create image copies outside of RMAN, you must use the CATALOG command to record them in the RMAN repository before RMAN can make use of them.

RMAN can also store its backups in an RMAN-specific format called a **backup set**. A backup set is a collection of files called **backup pieces**, each of which may contain the backup of one or several database files. A backup task performed in RMAN can create one or more backup sets, which are recorded in the RMAN repository. Backup sets are also the only form in which RMAN can write backups to media manager devices like tape libraries. Backup sets are only created and accessed through Recovery Manager.

See Also: ["About RMAN Backup Formats: Image Copies and Backup Sets"](#) on page 4-2 for more details about image copies and backup sets.

Automatic Disk-Based Backup and Recovery: The Flash Recovery Area

The components that create different backup and recovery-related files have no knowledge of each other or of the size of the file systems where they store their data. With Automatic Disk-Based Backup and Recovery, you can create a flash recovery area, which automates management of backup-related files. Choose a location on disk and an upper bound for storage space, and set a retention policy that governs how long backup files are needed for recovery, and the database manages the storage used for backups, archived redo logs, and other recovery-related files for your database within that space. Files no longer needed are eligible for deletion when RMAN needs to reclaim space for new files.

Using a flash recovery area minimizes the need to manually manage disk space for your backup-related files and balance the use of space among the different types of files. Oracle recommends that you enable a flash recovery area to simplify your backup management.

Oracle Flashback Technology: Alternatives to Point-in-Time Recovery

Oracle Flashback Technology provides a set of features that provide useful alternatives to support viewing past states of data, and winding data back and forth in time, without requiring you to restore large portions of your database from backup or perform point-in-time recovery. The flashback features of Oracle are more efficient and less disruptive than media recovery in most circumstances in which they are applicable.

Most of the flashback features of Oracle operate at the logical level, viewing and manipulating database objects, as follows:

- **Oracle Flashback Query** lets you specify a target time and then run queries against your database, viewing results as they would have appeared at that time. To recover from an unwanted change like an erroneous update to a table, a user could choose a target time before the error and run a query to retrieve the contents of lost or changed rows.
- **Oracle Flashback Version Query** lets you view all the versions of all the rows that ever existed in one or more tables in a specified time interval, as updates were applied to the tables. You can also retrieve metadata about the differing versions of the rows, including start time, end time, operation, and transaction ID of the

transaction that created the version. This feature can be used both to recover lost data values and to audit changes to the tables queried.

- **Oracle Flashback Transaction Query** lets you view changes made by a single transaction, or by all the transactions during a period of time.
- **Oracle Flashback Table** returns a table to its state at a previous point in time. You can restore table data while the database is online, undoing changes only to the specified table.
- **Oracle Flashback Drop** reverses the effects of a `DROP TABLE` statement.

Flashback Table, Flashback Query, Flashback Transaction Query and Flashback Version Query all rely on **undo data**, records of the effects of each update to an Oracle database and values overwritten in the update. Used primarily for such purposes as providing read consistency for SQL queries and rolling back transactions, these undo records contain the information required to reconstruct data as it stood at a past time and all changes since that time.

Flashback Drop is built around a mechanism called the **Recycle Bin**, which Oracle uses to manage dropped database objects until the space they occupied is needed to store new data.

Note: The logical-level flashback features of Oracle are not dependent upon RMAN, and are available whether or not RMAN is part of your backup strategy.

At the physical level, **Oracle Flashback Database** provides a more efficient direct alternative to database point-in-time recovery. If you have datafiles which merely have unwanted changes, then you can use Flashback Database to cause your current datafiles revert to their contents at a past time. The end product is much like the result of a point-in-time recovery, but is generally much faster because it does not require restoring datafiles from backup, and requires only limited application of redo compared to media recovery.

Flashback Database uses **flashback logs** to access past versions of data blocks, as well as some information from the archived redo log. Flashback Database requires that you configure a flash recovery area for your database, because the flashback logs can only be stored there. Flashback logging is not enabled by default. Space used for flashback logs is managed automatically by the database, and balanced against space required for other files in the flash recovery area.

Note:

- Flashback Database is integrated with RMAN, which can automatically retrieve from backup any archived logs needed during Flashback Database. It can also be used from within SQL*Plus without the use of RMAN, but if used in this mode you must make all required archived logs available on disk yourself.
 - If insufficient space is allocated for the flash recovery area, then flashback logs may be deleted to make room for backups and archived log files. Database point-in-time recovery can generally deliver the same result as flashback database, returning your database to its contents at a past point in time,
-
-

About Restore Points

The Oracle database also supports **restore points** in conjunction with Flashback Database and restore and recovery features.

A **normal restore point** is an alias corresponding to an SCN, which you can create at any time if you anticipate needing to return part or all of your database to its contents at that time. Future point-in-time recovery, Flashback Table and Flashback Database operations are simplified because you do not need to research or record the target SCN for the operation.

Creating a **guaranteed restore point** ensures that you can use Flashback Database to return your database to the time of the restore point.

See Also:

- ["Using Normal and Guaranteed Restore Points"](#) on page 5-6 for more information about the use of normal and guaranteed restore points
- [Chapter 7, "Performing Flashback and Database Point-in-Time Recovery"](#) for more information about the use of the flashback features of Oracle in a data recovery context
- *Oracle Database Concepts* and *Oracle Database Administrator's Guide* for more information on undo data and automatic undo management
- *Oracle Database Application Developer's Guide - Fundamentals* for more information on Flashback Query, Flashback Transaction Query and Flashback Version Query

Matching Failures to Backup and Recovery Techniques

In planning your database backup and recovery strategy, you must try to anticipate the errors that will arise, and put in place the backups needed to recover from them. While there are several types of problem that can halt the normal operation of an Oracle database or affect database I/O operations, only two typically require DBA intervention and media recovery: media failure, and user errors. Instance failures, network failures, failures of Oracle database background processes and failure of a statement to execute due to, for instance, exhaustion of some resource such as space in a datafile may require DBA intervention, and might even crash a database instance, but will not generally cause data loss or the need to recover from backup.

This section contains these topics:

- [Responding to Media Failure](#)
- [Responding to User Error](#)

Responding to Media Failure

Database operation after a media failure of online redo log files or control files depends on whether the online redo log or control file is protected by **multiplexing**, as recommended. When an online redo log or control file is multiplexed, multiple copies of the file are maintained on the system. Multiplexed files should be stored on separate disks.

If a media failure damages a disk containing one copy of a multiplexed online redo log, then the database can usually continue to operate without significant interruption.

Damage to a nonmultiplexed online redo log causes database operation to halt and may cause permanent loss of data.

Damage to any control file, whether it is multiplexed or not, halts database operation when the database attempts to read or write to the damaged control file (which happens frequently, for example at every checkpoint and log switch).

Media failures are either **read errors** or **write errors**. In a read error, the instance cannot read a datafile and an operating system error is returned to the application, along with an error indicating that the file cannot be found, cannot be opened, or cannot be read. The database continues to run, but the error is returned each time an unsuccessful read occurs. At the next checkpoint, a write error will occur when the database attempts to write the file header as part of the standard checkpoint process.

The effect of a datafile write error depends upon which tablespace the datafile is in.

If the instance cannot write to a datafile in the **SYSTEM tablespace**, an undo tablespace (if the database is in **automatic undo management mode**, which is the preferred choice in Release 10g), or a datafile in a tablespace containing active rollback segments (if in **manual undo management mode**), then the database issues an error and shuts down the instance. All files in the **SYSTEM** tablespace and all datafiles containing rollback segments must be online in order for the database to operate properly.

If the instance cannot write to a datafile other than those in the preceding list, then the result depends on whether the database is running in ARCHIVELOG mode or not.

In ARCHIVELOG mode, the database records an error in the database writer trace file and takes the affected datafile offline. (All other datafiles in the tablespace containing this datafile remain online.) You can then rectify the underlying problem and restore and recover the affected tablespace.

In NOARCHIVELOG mode, the database writer background process DBWR fails, and the instance fails, the cause of the problem determines the required response. If the problem is temporary (for example, a disk controller was powered off), then crash recovery usually can be performed using the online redo log files. In such situations, the instance can be restarted without resorting to media recovery. However, if a datafile is damaged, then you must restore a **consistent backup** of the entire database.

Responding to User Error

Typically, a user error like dropping a table or deleting rows from a table requires one of the following responses:

- Re-importing the dropped object, if a suitable export file exists or if the object is still available at a standby database
- Performing **tablespace point-in-time recovery (TSPITR)** of one or more tablespaces
- Re-entering the lost data manually, if a record of them exists
- Returning the database to a past state using database point-in-time recovery
- Using one of the flashback features of the Oracle database to recover from logical corruption by returning affected objects to a past state

The recovery options available to you will be a function of your backup strategy. For example, if you are running in NOARCHIVELOG mode then you have limited point-in-time recovery options.

See Also:

- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to perform point-in-time recovery for an entire database
- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to perform tablespace point-in-time recovery
- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to use the flashback features of the Oracle database

System Requirements for Backup and Recovery Methods

When choosing a backup and recovery solution, find one that is appropriate for the database environment. For example, if you manage only databases of release 8.0 or higher, then you can use RMAN to manage your backup and recovery requirements. Releases older than 8.0 will have to be managed using some method besides RMAN.

[Table 1–1](#) describes the version and system requirements for different database backup and recovery methods.

Table 1–1 Requirements for Different Backup Methods

Backup Method	Type	Version Available	Requirements
Recovery Manager (RMAN)	Physical	Oracle version 8.0 and higher	Third-party media manager (only if backing up to tape)
Operating System	Physical	All versions	Operating system backup utility (for example, UNIX <code>dd</code> command)
Export	Logical	All versions	N/A

Feature Comparison of Backup Methods

Besides being limited by system requirements, the backup and recovery solution you choose should be driven by the features that you want. [Table 1–2](#) compares the features of the different backup methods.

Table 1–2 Feature Comparison of Backup Methods

Feature	Recovery Manager	User-Managed	Export
Closed database backups	Supported. Requires instance to be mounted.	Supported.	Not supported.
Open database backups	Supported. No need to use <code>BEGIN/END BACKUP</code> statements.	Supported. Must use <code>BEGIN/END BACKUP</code> statements.	Requires rollback or undo segments to generate consistent backups.
Incremental backups	Supported.	Not supported.	Not supported.
Corrupt block detection	Supported. Identifies corrupt blocks and logs in <code>V\$DATABASE_BLOCK_CORRUPTION</code> .	Not supported.	Supported. Identifies corrupt blocks in the export log.
Automatic record keeping of files in backups	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.

Table 1–2 (Cont.) Feature Comparison of Backup Methods

Feature	Recovery Manager	User-Managed	Export
Recovery catalogs	Supported. Backups are recorded in the RMAN repository, which is contained in the control file and optionally in the recovery catalog database.	Not supported. DBA must keep own records of backups.	Not supported.
Backups to media manager	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 controlled by a media manager.	Supported.
Backs up initialization parameter file	Supported.	Supported.	Not supported.
Backs up password and networking files	Not supported.	Supported.	Not supported.
Platform-independent language for backups	Supported.	Not supported.	Supported.

Backup and Recovery Strategies

This chapter offers guidelines and considerations for developing an effective backup and recovery strategy.

This chapter includes the following sections:

- [Data Recovery Strategy Determines Backup Strategy](#)
- [Planning Data Recovery Strategy](#)
- [Planning Backup Strategy](#)
- [Validating Your Data Recovery Strategy](#)

Data Recovery Strategy Determines Backup Strategy

To decide on backup strategies, start with your data recovery requirements and your data recovery strategy. Each type of data recovery will require that you take certain types of backup.

Failures can run the gamut from user error, datafile block corruption and media failure to situations like the complete loss of a data center. How quickly you can resume normal operation of your database is a function of what kinds of restore and recovery techniques you include in your planning. Each restore and recovery technique will impose requirements on your backup strategy, including which features of the Oracle database you use to take, store and manage your backups.

When thinking about recovery strategies, ask yourself questions like these:

- If a disk failed and destroyed some of the database files, such as datafiles or redo logs, how would you recover the lost files? As described in "[Planning a Response to Media Failure: Restore and Media Recovery](#)" on page 2-4, you should be able to handle the loss of datafiles, control files, and online redo logs.
- If a logic error in an application or a user error caused the loss of important data from one or several tables or tablespaces, how could you recover that data, and what would happen to database updates since the error? Could you determine the cause of the error, to prevent it from happening again? As described in "[Planning Responses to User Error: Point-in-Time Recovery and Flashback Features](#)" on page 2-3, techniques available to you include point-in-time recovery of the whole database or one or more tablespaces, importing data from earlier logical exports with one of the data import utilities, and using the Oracle database's flashback features.
- If the instance alert log indicates that one or more tables contains corrupt blocks, how can you repair the corruption? Does the tablespace have to remain available during the repair? As described in "[Planning a Response to Datafile Block](#)

"Corruption: Block Media Recovery" on page 2-5, the RMAN `BLOCKRECOVER` command can help you in this situation. Also, troubleshoot recovery with the `SQL*Plus RECOVER . . . TEST` command.

- If the entire data center is destroyed, can you perform disaster recovery? Assume that all you have is an archive tape containing backups. How would you recover the database? How long would that recovery take?
- If you were not available to recover your database, could someone else recover it in your absence? Are your recovery procedures sufficiently automated and documented?

With these needs in mind, decide how you can take advantage of features related to backup and recovery, and look at how each feature meets some requirement of your backup strategy. For example:

- Using Recovery Manager simplifies most backup and recovery operations compared to user-managed backup and recovery. It automates management of most backup files, including the deletion of backups and archived redo logs from disk or tape when no longer needed to meet recovery goals. It provides detailed reporting on backup activities, can verify that your available backups can be used to recover your database. Finally, RMAN makes possible many recovery techniques not available if you are using user-managed backup and recovery, such as incremental backups.
- Flashback Database will help you restore a database to a previous time much faster than media recovery. However, you must decide in advance to keep flashback logs, and keeping flashback logs requires that you configure a flash recovery area.
- Block media recovery may be better than datafile media recovery if availability is critical. While block media recovery is possible even if you do not base your backup and recovery strategy on RMAN, RMAN-based block media recovery can be performed more quickly and with less effort.

Once you decide which features to use in your recovery strategy, you can plan your backup strategy, answering the following questions, among others:

- How and where will you store your recovery-related files? Will you use a flash recovery area? Will you use an ASM disk group to provide redundancy? Will you store backups on tape or other offline storage, or only on disk?
- At what intervals will you take scheduled backups? And what form of physical backups will you take in each situation?
- What situations require you to take a database backup outside of the regular schedule? Sometimes you must take an unscheduled backup to ensure that you can recover your data, such as after an `OPEN RESETLOGS` or after changes to your database such as `NOLOGGING` operations that do not appear in the redo log. You may also have business requirements that require backups for auditing purposes or other reasons not related to database recovery.
- How can you validate your backups, to ensure that you can recover your database when necessary?
- How do you manage records of your backups? Do you use RMAN with a recovery catalog?
- Do you have detailed recovery plans that cover each type of failure? How do your DBAs can execute these plans in a crisis? Can scripts be written to automate execution of these plans in a crisis?

- Can you apply Oracle database availability technologies, such as Data Guard or Real Application Clusters, to improve availability during a database failure? How does using these availability technologies affect your backup and recovery strategy?

These are of course only a few of the considerations you should take into account. Available resources (hardware, media, staff, budget, and so on) will also be factors in your decision.

Planning Data Recovery Strategy

Your data recovery strategy should include responses to any number of database failure scenarios. The key to an effective, efficient strategy is envisioning failure modes, matching Oracle database recovery techniques and tools to the failure modes in which they are useful, and then making sure you incorporate the necessary backup types to support those recovery techniques.

To help match failure modes to recovery techniques that can help resolve them, refer to the following sections:

- [Planning Responses to User Error: Point-in-Time Recovery and Flashback Features](#)
- [Planning a Response to Media Failure: Restore and Media Recovery](#)
- [Planning a Response to Datafile Block Corruption: Block Media Recovery](#)

Planning Responses to User Error: Point-in-Time Recovery and Flashback Features

A user or application may make unwanted changes to your database, such as erroneous updates, deleting the contents of a table, or dropping database objects. An adequate backup and recovery strategy uses the many features of the Oracle database to let you return your database to the desired state, with the minimum possible impact upon database availability, and minimal DBA effort.

Depending on the situations you anticipate, consider incorporating each of the following options into your strategy for responding to data loss, and then set up your database to make these options possible.

Flashback Database

Using Flashback Database enables you to return your whole database to a previous state without restoring old copies of your datafiles from backup, as long as you have enabled logging for flashback database in advance.

You can turn on flashback logging to allow a return to an arbitrary SCN as far in the past as the available flashback logs permit, or you can create guaranteed restore points at specific SCNs, such as before major database updates, which ensure that Flashback Database can be used to return the database to its state at a specific moment.

You must have a flash recovery area configured for logging for flashback database or guaranteed restore points.

Creating Normal and Guaranteed Restore Points

As noted, guaranteed restore points ensure that you can return your database to a specific previous time using Flashback Database. Normal restore points do not provide protection for your data, but they are a convenience because by creating one, you can avoid having to record the SCN of the database before an operation from which you wish to recover using point-in-time recovery or Flashback Table, or having to investigate after the operation to determine the correct SCN.

No special setup is required for using normal restore points, though you must plan on creating restore points before they are needed.

Database Point-in-Time Recovery

You can perform point-in-time recovery, bringing one tablespace or the whole database back to its state before the time of the error. In either case, you need backups from before the time of the error, plus the redo logs from the time of the backup to the time of the error.

Importing Lost Objects from Logical Backup

If you have performed a logical backup by exporting the contents of the affected tables, sometimes you can import the data back into the table. This technique presumes that you are regularly exporting logical backups of your data, and that any changes between exports are unimportant.

Note: Oracle's Flashback Technology provides faster and less disruptive alternatives to media recovery in many circumstances.

- Oracle Flashback Database is a physical-level recovery mechanism similar to media recovery, but generally faster and not requiring the restore of data from backup.
- Oracle Flashback Table and Oracle Flashback Drop work at the logical level, undoing unwanted changes to tables, including reversing the effects of `DROP TABLE` statements.
- Oracle Flashback Query and Oracle Flashback Version Query are useful in viewing past contents of tables and investigating how and when logical corruptions affected your database.

Information about these features is collected in [Chapter 7, "Performing Flashback and Database Point-in-Time Recovery"](#). This document will allude to such features where they can be helpful and provide pointers for more information. Familiarize yourself with these features before planning your backup and recovery strategy, because you may find that they can be quite valuable and require limited advanced planning.

Planning a Response to Media Failure: Restore and Media Recovery

A media failure occurs when a problem external to the database prevents Oracle from reading from or writing to a file during database operations. Typical media failures include physical failures, such as head crashes, and the overwriting, deletion or corruption of a database file. Media failures are less common than user or application errors, but your backup and recovery strategy should prepare for them.

The type of media failure determines the recovery technique to use. For example, the strategy you use to recover from a corrupted datafile is different from the strategy for recovering from the loss of the control file.

Example: Online Redo Log Recovery

The method of recovery from loss of all members of an online log group depends on a number of factors, such as:

- The state of the database (open, crashed, closed consistently, and so on)

- Whether the lost redo log group was current
- Whether the lost redo log group was archived

For example:

- If you lose the current group, and the database is not closed consistently (either it is open, or it has crashed), then you will have to restore an old backup and perform point-in-time recovery, followed by `OPEN RESETLOGS`. You will lose all transactions that were in the lost log. You should take a new full database backup immediately after the `OPEN RESETLOGS`. Backups from before the `OPEN RESETLOGS` will not be recoverable because of the lost log.
- If you lose the current redo log group, and if the database is closed consistently, then you can perform `OPEN RESETLOGS` with no transaction loss. However, you should take a new full database backup. Backups from before the `OPEN RESETLOGS` will not be recoverable because of the lost log.
- If you lose a noncurrent redo log group, then you can use the `ALTER DATABASE CLEAR LOGFILE` statement to re-create all members in the group. No transactions are lost. If the lost redo log group was archived before it was lost, then nothing further is required. Otherwise, you should immediately take a new full backup of your database. Backups from before the log was lost will not be recoverable because of the lost log.

Planning a Response to Datafile Block Corruption: Block Media Recovery

If a small number of blocks within one or more datafiles are corrupt, you can perform **block media recovery** instead of restoring the datafiles from backup and performing complete media recovery of those files. The Recovery Manager `BLOCKRECOVER` command can be used to restore and recover specified data blocks while the database is open and the corrupted datafile is online.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to perform block media recovery with RMAN.

Planning Backup Strategy

Your plans for data recovery strategies are the basis of your plans for backup strategy. This discussion describes general guidelines that can help you decide when to perform database backups, which parts of a database you should back up, what tools Oracle provides for those backups, and how to configure your database to improve its robustness and make backup and recovery easier. Of course, the specifics of your strategy must balance the needs of your restore strategy with questions of cost, resources, personnel and other factors.

Protecting Your Redundancy Set

The set of files needed to recover an Oracle database from the failure of any of its files—a datafile, control file, or online redo log—is called the **redundancy set**. The redundancy set should contain:

- The last backup of the control file and all the datafiles
- All archived redo logs generated after the last backup was taken
- Duplicates of the current control file and online redo log files, generated by Oracle database multiplexing, operating system mirroring, or both

- Copies of configuration files such as the server parameter file, `tnsnames.ora`, and `listener.ora`

Caution: Do not store the redundancy set on the same disk that contains the datafiles, online logs and control files of the database. Otherwise, the disk becomes a single point of failure for the database. If this disk fails, you lose committed transactions.

A minimal production-level database thus requires at least two disk drives: one to hold the files in the redundancy set and one to hold the database files. Ideally, separate the redundancy set from the primary files in every way possible: on separate volumes, separate file systems, and separate RAID devices.

The simplest way to manage your redundancy set is to use a flash recovery area, on a separate device from the working set files. However, whether or not you use a flash recovery area, Oracle recommends the following practices:

- Multiplex the online redo log files and current control file at the database level. (For instance, configure the database to write its online logs to two or more destinations, so that each write is a separate operation carried out by the database, rather than by operating system-level or hardware-level redundancy.) If you multiplex at the database level, then an I/O failure or lost write should only corrupt one of the copies.

Ideally, the multiplexed files should be on different disks mounted under different disk controllers. The flash recovery area is an excellent location for one copy of these files.

You can also mirror the online redo logs and current control file at the operating system or hardware level, but this is not a substitute for multiplexing at the database level.

- If running in ARCHIVELOG mode, archive the redo logs to multiple locations, ideally on different disks. If you are using a flash recovery area, use it as one of the archiving locations.
- Use operating system or hardware mirroring for the control file. All copies of the control file multiplexed at the database level must be available at all times, or the instance will crash. If you use operating system or hardware mirroring for your control file, your database can continue to operate even if one copy of the control file mirrored at the operating system level is unavailable due to a disk failure.
- Use operating system or hardware mirroring for the primary datafiles if possible, to avoid having to perform media recovery for simple disk failures.
- Keep at least one copy of the entire redundancy set—including the most recent backup—on disk. The flash recovery area is the ideal location for the redundancy set.
- If the target database is stored on a RAID device, then store the redundancy set on a set of disks that are not in the same RAID device.
- If you store the redundancy set on tape, then maintain at least two copies of the data to protect against the risk of tape failure. Also, if you have more than one backup of the same data, then consider keeping backups from different points in time. In this way, if one backup or split mirror was created when the database was corrupted, you may still have a backup from a time when the database was not corrupted.

Deciding Whether to Use a Flash Recovery Area

It is recommended that you take advantage of the flash recovery area to store as many backup and recovery-related files as possible, including disk backups and archived redo logs.

Some features of Oracle database backup and recovery, such as Oracle Flashback Database and guaranteed restore points, require the use of a flash recovery area. However, having a flash recovery area for use by these features does not force you to use it to store all recovery-related files.

Even when its use is not required, however, the flash recovery area offers a number of advantages over other on-disk backup storage methods. Backups moved to tape from the flash recovery area are retained on disk until space is needed for other required files, reducing the need to restore backups from tape. At the same time, obsolete files no longer needed to meet your recoverability goals and files backed up to tape become eligible for deletion and are deleted when space is needed. The DBA no longer has to manually delete old backups, and, it is less likely that a DBA accidentally deletes redundancy set files.

See Also: ["Setting Up a Flash Recovery Area for RMAN"](#) on page 3-13 for more about the uses and benefits of the flash recovery area.

Deciding Between ARCHIVELOG and NOARCHIVELOG Mode

The redo logs of your database provide a complete record of changes to the datafiles of your database (with a few exceptions, such as direct path loads).

You can run your database in one of two modes: ARCHIVELOG mode or NOARCHIVELOG mode. In ARCHIVELOG mode, a used online redo log group must be copied to one or more archive destinations before it can be reused. Archiving the redo log preserves all transactions stored in that log, so that they can be used in recovery operations later. In NOARCHIVELOG mode, the online redo log groups are simply overwritten when the log is reused. All information about transactions recorded in that redo log group is lost.

Implications of Running in NOARCHIVELOG Mode

Running your database in NOARCHIVELOG mode imposes severe limitations on your backup and recovery strategy.

- You cannot perform online backups of your database. You must shut your database down cleanly before you can take a backup in NOARCHIVELOG mode.
- You cannot use any data recovery techniques that require the archived redo logs. These include complete and point-in-time media recovery, as described in ["Forms of Data Recovery"](#) on page 1-6, and more advanced recovery techniques such as point-in-time recovery of individual tablespaces and Flashback Database (described in *Oracle Database Backup and Recovery Advanced User's Guide*).

If you are running in NOARCHIVELOG mode and you must recover from damage to datafiles due to disk failure, you have two main options for recovery:

- Drop all objects that have any extents located in the affected files, and then drop the files. The remainder of the database is intact, but all data in the affected files is lost.

- Restore the entire database from the most recent backup, and lose all changes to the database since the backup. (Recovering changes since the backup would require performing media recovery, which uses the archived redo logs.)

Implications of Running in ARCHIVELOG Mode

For most applications, running in ARCHIVELOG mode is preferable to running in NOARCHIVELOG mode because you have more flexible recovery options after a data loss, such as point-in-time recovery of the database or some tablespaces. There are, however, associated costs of running in ARCHIVELOG mode:

- Space must be set aside for archiving destinations, locations on disk where the archived redo logs will be stored. These can become quite large in databases with large numbers of updates.
- The stored archived redo logs must be managed. To limit the disk space used by archived redo logs, archived redo logs can be moved to tape for longer-term storage, and older logs no longer needed to meet your recoverability goals should be deleted. (RMAN can automate most of the management of archived redo logs, by recording the location and contents of all archived redo logs, making it easy to move archived logs to tape, and identifying and deleting redo logs no longer required to meet your recoverability objectives.)
- Some performance overhead is associated with the background processes ARC0 through ARC*n* which copy filled online redo logs to the archiving destinations.

When performance requirements are extreme or disk space limitations are severe, it may be preferable to run in NOARCHIVELOG mode in spite of the limitations that this choice imposes upon your recovery options.

Deciding Whether to Use Oracle Flashback Features and Restore Points

Using the flashback features of Oracle improves the availability of your database when repairing the effects of unwanted database changes. The logical-level flashback features allow the objects not affected to remain available, and Flashback Database allows for faster rewind of the entire database than point-in-time recovery.

If you intend to take advantage of the logical-level flashback features of Oracle, you must take into account how the database manages undo data. See *Oracle Database Concepts* and *Oracle Database Administrator's Guide* for more information about undo data and automatic undo management.

Incorporating Flashback Database or guaranteed restore points into your strategy requires that you enable a flash recovery area, as well as creating guaranteed restore points at the right points in time, or configuring flashback logging. See [Chapter 5, "Data Protection with Restore Points and Flashback Database"](#) for more information on the role that these features can play in your data protection strategy and the requirements for using them separately or together.

Choosing a Backup Retention Policy

Your backup retention policy is the rule you set regarding which backups must be retained (whether on disk or other backup media) to meet your recovery and other requirements.

Backup retention policy can be based on **redundancy** or a **recovery window**. In a redundancy-based retention policy, you specify a number *n* such that you always keep at least *n* distinct backups of each file in your database. In a recovery window-based retention policy, you specify a time interval in the past (for example, one week, or one

month) and keep all backups required to let you perform point-in-time recovery to any point during that window.

A backup no longer needed to satisfy the backup retention policy is said to be **obsolete**.

Implementing Backup Retention Policy with RMAN

RMAN automates the implementation of a backup retention policy, using the following commands:

- `CONFIGURE RETENTION POLICY` command lets you set the retention policy that will apply to all of your database files by default.
- `REPORT OBSOLETE` command lets you list backups currently on disk that are obsolete under the retention policy. You can also specify parameters to see which files would be obsolete under different retention policies.
- `DELETE OBSOLETE` command deletes the files which `REPORT OBSOLETE` lists as obsolete.
- `CHANGE... KEEP` lets you set a separate retention policy for specific backups, such as **long-term backups** kept for archival purposes. You can specify that a given backup must be kept until a future time, or even specify that a backup be kept forever. `CHANGE... NOKEEP` is used to let the retention policy apply to a backup previously protected by `CHANGE... KEEP`.

If you use a flash recovery area to store your backups, the database will delete obsolete backups automatically as disk space is needed for newer backups, archived logs and other files. For backups stored on disk outside a flash recovery area and for backups stored on tape, you should periodically run the `DELETE OBSOLETE` command to remove obsolete backups.

Recovery Window-Based Backup Retention Policy

A recovery window-based retention policy lets you guarantee that you can perform point-in-time recovery to any point in the past, up to a number of days that you specify. The earliest point in time to which you can recover your database under your retention policy is known as the **point of recoverability**. All backups required for recovery or point-in-time recovery back to that time will be retained.

Use a recovery window-based backup retention policy if your business requirements dictate that any possible logical damage to the database must be repairable as long as it is discovered within a given period of time. Set the recovery window to that period of time.

Note that satisfying a recovery window-based retention policy will generally require that you keep backups older than the beginning of the recovery window. A point-in-time recovery to the beginning of the recovery window would require a restore from this backup, and then applying all changes between the backup time and the point of recoverability. For example, you might configure a recovery window of three days:

```
RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 3 DAYS;
```

If your last full database backup was six days ago, RMAN will keep the six-day-old backup, and all redo logs required to roll the database forward to the beginning of the recovery window three days ago, in addition to any backups and redo logs needed to recover the database to all points in time within the three day window.

A recovery window-based backup retention policy provides the most certain recoverability for your data. The disadvantage is that more careful disk space planning is required, since it may not be obvious how many backups of datafiles and archived logs must be retained to guarantee the recovery window.

Redundancy-Based Backup Retention Policy

A redundancy-based backup retention policy determines whether a backup is obsolete based on how many backups of a file are currently on disk. You might configure a redundancy level of 3:

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 3;
```

In this case, RMAN keeps three backups of each database file, and all redo logs required to recover all retained datafile backups to the current time. Any older backups will be considered obsolete.

For example, assume that you make backups of a datafile every day, starting on a Monday. On Thursday, you make your fourth backup of the datafile, and the backup from Monday becomes obsolete because you have the backups from Tuesday, Wednesday and Thursday. On Friday, the backup from Tuesday becomes obsolete, because you have the backups from Wednesday, Thursday and Friday.

Archiving Older Backups

There are several reasons to keep older backups of datafiles and archived logs:

- An older backup of datafiles and archived logs is necessary for performing point-in-time recovery to a time before your most recent backup.
- If your most recent backup is corrupt, you can still recover your database using an older backup and the complete set of archived logs since that older backup.
- You may want to keep a copy of the database for archival purposes.

To perform point-in-time recovery to a given target time earlier than your current point of recoverability, then you need a database backup that completed before the target time, as well as all of the archived logs created between the time the backup was started and the target time. For example, if you take full database backups starting at 1:00 AM on February 1 (at SCN 10000) and on February 14 (at SCN 20000), and if you decide on February 28 to use point-in-time recovery to bring your database to its state at 9:00AM February 7 (SCN 13500), then you must use the February 1 backup, plus all redo logs containing changes from between the beginning of the creation of the backup (SCN 10000) and 9:00AM February 7 (SCN 13500).

Determining Backup Frequency

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

The more frequently your database is updated, the more often you should perform database backups. The scenario in ["Backup Scripts When Blocks Change Frequently"](#) on page A-5 backs up the database every week.

If database updates are relatively infrequent, then you can make whole database backups infrequently and supplement them with incremental backups (which will be relatively small because few blocks have changed). The scenario in "[Backup Scripts When Few Data Blocks Change](#)" on page A-8 describes how to develop a backup strategy based on a single whole database backup.

See Also: [Appendix A, "RMAN-Based Disk and Tape Backup Strategies: Scenarios"](#) on page A-1

Performing Backups Before and After You Make Structural Changes

There are times when you will need to take a backup of your database independent of your regular backup schedule. If you make any of the following structural changes, then perform a backup of the appropriate portion of your database immediately before and after completing the following changes:

- 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.

If you are in NOARCHIVELOG mode, then you must shut down the database and perform a consistent whole database backup after any such change. If you are running in ARCHIVELOG mode, then you must make a control file backup after any such change, using either RMAN's `BACKUP CONTROLFILE` command or the `SQL ALTER DATABASE BACKUP CONTROLFILE` statement.

Scheduling Backups for Frequently-Updated Tablespaces

If you run in ARCHIVELOG mode, then you can back up an individual tablespace or even a single datafile. You might want to do this for one or more tablespaces that are updated much more often than the rest of your database, as is sometimes the case for the `SYSTEM` tablespace and automatic undo tablespaces.

More frequent backups of heavily-used datafiles can shorten recovery times in some situations. You may have a database where most updates are restricted to a small set of tablespaces. If you take a full database backup each Sunday, then recovery from a media failure affecting the frequently updated tablespaces on Friday requires re-applying large amounts of redo. Daily backups of the frequently-updated tablespaces reduces the amount of redo to apply without requiring a daily full database backup.

See Also: *Oracle Database Administrator's Guide* for information about managing undo tablespaces

Backing Up after NOLOGGING Operations

When a direct path load is performed to populate a database, no redo data is logged for those database changes. You cannot recover these changes after a restore from backup using conventional media recovery. Likewise, when tables and indexes are created as `NOLOGGING`, the database does not log redo data for these objects, which means that you cannot recover these objects from existing backups. Therefore, you should back up your datafiles after operations for which no redo data is logged.

Note: You can use either a full backup of your datafiles or an incremental backup. Either one will capture all changed blocks, including blocks changed by unrecoverable operations.

See Also: *Oracle Database SQL Reference* for information about the UNRECOVERABLE option of the CREATE TABLE . . . AS SELECT and CREATE INDEX statements.

Exporting Data for Added Protection and Flexibility

Oracle database import and export utilities are used to export database objects (tables, stored procedures, and so forth) from databases to be stored as files, and re-import objects from those files. An export provides a logical-level snapshot of the exported objects at the time of the export, as a binary file that can be imported back into the source database or some other database. Consider exporting portions or all of a database for supplemental protection and flexibility in a database's backup strategy.

While useful, database exports are not a substitute for whole database backups. They cannot provide the same complete recovery advantages of physical-level backups. For example, you cannot apply archived logs to logical backups in order to update lost changes.

See also: *Oracle Database Utilities* for more details about exporting and importing data for logical backup

Preventing the Backup of Online Redo Logs

Online redo logs, unlike archived logs, should never be backed up. The chief danger associated by having backups of online redo logs is that you may accidentally restore those backups without meaning to, and corrupt your database.

Online redo log backups are also not particularly useful, for the following reasons:

- If your database is in ARCHIVELOG mode, then the archiver is already archiving the filled redo logs automatically.
- If your database is in NOARCHIVELOG mode, then the only type of physical backups that you can 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 useful after a restore from backup.

The best method for protecting the online logs against media failure is to multiplex them, with multiple log members in each group, on different disks attached to different disk controllers.

Note: RMAN does not permit you to back up online redo logs. You must archive a redo log before backing it up.

Keeping Records of the Hardware and Software Configuration of the Server

During the stress of a recovery situation, it is important that you have all necessary information at your disposal. This is especially true if for some reason you need to contact Oracle Support because you run into a problem that you do not understand. You should have the following documentation about the hardware configuration:

- The name, make, and model of the machine that hosts the database

- The version and patch of the operating system
- The number of disks and disk controllers
- The disk capacity and free space
- The names of all datafiles
- The name and version of the media management software (if you use a third-party media manager)

You should also keep the following documentation about the software configuration:

- The name of the database instance (SID)
- The database identifier (DBID)
- The version and patch release of the Oracle database server
- The version and patch release of the networking software
- The method (RMAN or user-managed) and frequency of database backups
- The method of restore and recovery (RMAN or user-managed)

You should keep this information both in electronic and hardcopy form. For example, if you save this information in a text file on the network or in an email message, then if the entire system goes down, you may not have access to this data.

It is especially important to keep a record of the DBID. If you have to restore and recover your database including the loss of the SPFILE and control file, you will need the DBID during the recovery process. See "[Basic Database Restore and Recovery Scenarios](#)" on page 6-3 for details on how the DBID is used during recovery.

Validating Your Data Recovery Strategy

Practice backup and recovery techniques 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 RMAN, then one option is to run the `DUPLICATE` command to create a test database using backups of your production database. If you perform user-managed backup and recovery, then you can either create a new database, a standby database, or a copy of an existing database to test your backups.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* to learn about RMAN testing methods, troubleshooting SQL*Plus recovery, block media recovery, and RMAN disaster recovery

Using BACKUP... VALIDATE

Using the `RMAN BACKUP . . . VALIDATE` command causes RMAN to read all of the specified database files that would be input for a specific backup task, without actually producing any backups as output. For example, `BACKUP DATABASE VALIDATE` causes RMAN to read all files that would be backed up in backing up the entire database and ensure that they are intact and not corrupted. All of the data blocks in the input files are validated, exactly as they are when a real backup takes place. This process can provide a useful integrity check for your database.

Validating RMAN Backups: VALIDATE and RESTORE VALIDATE

The RMAN `VALIDATE` and `RESTORE VALIDATE` commands should be part of ongoing testing of your recovery plan. `VALIDATE` causes RMAN to read specified backups on disk or tape and report whether they are intact and usable in a restore operation. `RESTORE . . . VALIDATE` causes RMAN to check whether the set of available backups is sufficient to restore the specified database objects. For example, `RESTORE TABLESPACE TBS_1 VALIDATE` selects backups sufficient to restore the named tablespace, just as RMAN does in a real restore operation, and reads the backups to ensure that they are present and not corrupted.

See Also: ["Using RMAN to Validate Database Files"](#) on page 4-21 for more details on using `BACKUP VALIDATE` and `RESTORE VALIDATE`

Testing Your Database Restore and Recovery Procedures

You can also test your backups by performing a complete test of your restore and recovery strategy onto different hardware. Ideally, use a hardware and software configuration for the test that is as similar as possible to the environment available to you in a real disaster recovery scenario.

Setting Up and Configuring Backup and Recovery

This chapter describes how to start and interact with the RMAN client, and prepare the RMAN environment for implementation of your backup and recovery strategy.

This chapter includes the following topics:

- [Overview of Interacting With the RMAN Client](#)
- [Connecting the RMAN Client to Databases](#)
- [Setting Up a Database for RMAN Backup](#)
- [Setting Up a Flash Recovery Area for RMAN](#)

Note: While Oracle Flashback Database and guaranteed restore points are not true database backups, they can be an important part of your data protection strategy. These features may require some initial setup, depending upon how you incorporate them in your backup strategy. See [Chapter 5, "Data Protection with Restore Points and Flashback Database"](#) for more information about how to set up your database to use these features.

Overview of Interacting With the RMAN Client

This section describes basic interactions with the RMAN client, such as starting and exiting the RMAN client, entering commands at the command prompt, and using command line arguments. It contains the following sections:

- [Starting and Exiting RMAN](#)
- [Setting Globalization Support Environment Variables for RMAN](#)
- [Entering RMAN Commands at the Command Prompt](#)
- [Using Command Files with RMAN](#)
- [Checking Syntax of RMAN Commands and Command Files: CHECKSYNTAX](#)

Starting and Exiting RMAN

You have the following basic options for starting RMAN:

- Start the RMAN executable at the operating system command line without specifying any connection options, as in this example:

```
% rman
```

- Start the RMAN executable at the operating system command line while connecting to a target database and, possibly, a recovery catalog, as in these examples:

```
% rman TARGET /  
% rman TARGET SYS/oracle@trgt NOCATALOG  
% rman TARGET / CATALOG rman/cat@catdb
```

Note: Most RMAN commands require that RMAN connect to at least a target database to perform useful work. See "[Connecting the RMAN Client to Databases](#)" on page 3-5 for more details on connecting RMAN to different types of databases.

To quit RMAN and terminate the program, type `EXIT` or `QUIT` at the RMAN prompt. For example:

```
RMAN> EXIT
```

Setting Globalization Support Environment Variables for RMAN

Before invoking RMAN, it may be useful to 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'
```

If you are going to use RMAN to connect to an unmounted database and mount the database later while RMAN is still connected, then set the `NLS_LANG` environment 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 returns errors after the database is mounted. For example, if the character set is `WE8DEC`, you can set the `NLS_LANG` parameter as follows:

```
NLS_LANG=american_america.we8dec
```

`NLS_LANG` and `NLS_DATE_FORMAT` must be set for `NLS_DATE_FORMAT` to be used.

See Also:

- *Oracle Database Reference* for more information about the `NLS_LANG` and `NLS_DATE_FORMAT` parameters
- *Oracle Database Globalization Support Guide*

Entering RMAN Commands at the Command Prompt

When the RMAN client is ready for your commands, it displays the command prompt, as in this example:

```
RMAN>
```

Enter commands for RMAN to execute. For example:

```
RMAN> CONNECT TARGET /
```

```
RMAN> CONNECT CATALOG rman/rman@inst2
```

```
RMAN> BACKUP DATABASE ;
```

Most RMAN commands take a number of parameters and must end with a semicolon. (The few exceptions, such as `STARTUP`, `SHUTDOWN`, and `CONNECT`, can be used with or without a semicolon.)

When you enter a line of text that is not a complete command, RMAN prompts for continuation input with a line number. For example:

```
RMAN> BACKUP DATABASE
2> INCLUDE CURRENT
3> CONTROLFILE
4> ;
```

Using Command Files with RMAN

For repetitive tasks, you can create a text file containing RMAN commands, and start the RMAN client with the `@` argument, followed by a filename. For example, create a text file `cmdfile1` in the current directory contained one line of text as shown here:

```
BACKUP DATABASE INCLUDE CURRENT CONTROLFILE;
```

You can run this command file from the command line as shown in this example, and the command contained in it is executed:

```
% rman TARGET / @cmdfile1
```

After the command completes, RMAN exits.

You can also use the `@` command at the RMAN command prompt to execute the contents of a command file during an RMAN session. RMAN reads the file and executes the commands in it. For example:

```
RMAN> @cmdfile1
```

After the command file contents have been executed, RMAN displays the following message:

```
RMAN> **end-of-file**
```

Unlike the case where a command file is executed from the operating system command line, RMAN does not exit.

See also: *Oracle Database Backup and Recovery Reference* for more details about using `@` and command files

Checking Syntax of RMAN Commands and Command Files: CHECKSYNTAX

You may want to test some RMAN commands for syntactic correctness without executing them, either entering them at the command prompt or reading them in from a command file. Use the command-line argument `CHECKSYNTAX` to start the RMAN client in a mode in which it only parses the commands you enter and returns an RMAN-00558 error for commands that are not legal RMAN syntax.

Checking RMAN Syntax at the Command Line: Example

To test commands at the command line, start RMAN with the `CHECKSYNTAX` parameter, and then enter the commands to be tested, as shown in the following example:

```
% rman CHECKSYNTAX
Recovery Manager: Release 10.2.0.1.0 - Production on Sun Jun 12 02:41:03 2005

Copyright (c) 1982, 2005, Oracle. All rights reserved.
RMAN> run [ backup database; ]

RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-00558: error encountered while parsing input commands
RMAN-01006: error signalled during parse

RMAN-02001: unrecognized punctuation symbol "["

RMAN> run { backup database; }

The command has no syntax errors

RMAN>
```

Checking RMAN Syntax in Command Files: Example

To test commands in a command file, start RMAN with the `CHECKSYNTAX` parameter, and use the `@` argument to name the command file to be passed. For example, use a text editor to create a command file `/tmp/goodcmdfile` with the following contents:

```
#command file with legal syntax
restore database;
recover database;
```

Also create another command file, `/tmp/badcmdfile`, with the following contents:

```
#command file with bad syntax commands
restore database
recover database
```

When you run the command files through RMAN with `CHECKSYNTAX`, the output is:

```
% rman CHECKSYNTAX @/tmp/goodcmdfile
Recovery Manager: Release 10.2.0.1.0 - Production on Sun Jun 12 02:41:03 2005

Copyright (c) 1982, 2005, Oracle. All rights reserved.

RMAN> # command file with legal syntax
2> restore database;
3> recover database;
4>
The cmdfile has no syntax errors

Recovery Manager complete.

% rman checksyntax @/tmp/badcmdfile
Recovery Manager: Release 10.2.0.1.0 - Production on Sun Jun 12 02:41:03 2005

Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

```

RMAN> #command file with syntax error
2> restore database
3> recover
RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS=====
RMAN-00571: =====
RMAN-00558: error encountered while parsing input commands
RMAN-01005: syntax error: found "recover": expecting one of: "archivelog,
            channel, check, controlfile, clone, database, datafile, device,
            from, force, high, (, preview, ;, skip, spfile, standby, tablespace,
            until, validate"
RMAN-01007: at line 3 column 1 file: /tmp/badcmdfile

```

Using RMAN to Start Up and Shut Down Databases

When an RMAN procedure requires that your database be started, shut down or brought to MOUNT or NOMOUNT state, you can use the RMAN client to start up and shut down the target database. The following example uses the RMAN SHUTDOWN and STARTUP commands to shut the target database down cleanly, and then mount it in preparation for a backup:

```

% rman TARGET /
RMAN> SHUTDOWN IMMEDIATE # closes database consistently
RMAN> STARTUP MOUNT

```

To change the state of a target database that is in NOMOUNT or MOUNT state, you must either use SQL*Plus or use the RMAN SQL command to issue a SQL statement, as shown in these examples:

```

RMAN> SQL 'ALTER DATABASE OPEN';

RMAN> SQL 'ALTER DATABASE OPEN';

RMAN> SQL 'ALTER DATABASE OPEN';

```

Connecting the RMAN Client to Databases

This section describes connecting the RMAN client to target databases. It contains the following topics:

- [Types of Database Connections Used with RMAN](#)
- [Authentication for Database Connections](#)
- [Connecting to the Target Database from the Command Line](#)
- [Connecting to the Target Database from the RMAN Prompt](#)

In the examples in this section, the generic values used have the following meanings.

Value Used in Example	Meaning
SYS	User with SYSDBA privileges
oracle	The password for connecting as SYSDBA specified in the target database's orapwd file
trgt	The net service name for the target database

Types of Database Connections Used with RMAN

To perform useful work, the RMAN client must connect to a target database, the database to be backed up or recovered. Depending upon the task to be performed and your specific backup strategy, the RMAN client may also connect to two other databases:

- The recovery catalog database, which provides an optional backup store for the RMAN repository in addition to the control file
- An auxiliary database, which may be a standby database, or an instance created for performing a specific task such as duplicating a database, transporting tablespaces without taking the making database read-only, or performing tablespace point-in-time recovery.

Note: For many tasks that use an auxiliary database, RMAN creates an automatic auxiliary instance for use during the task, connects to it, performs the task, and then destroys it when the task is completed. You do not given any explicit command to connect to automatic auxiliary databases.

Authentication for Database Connections

When connecting to a target or auxiliary database, you must have the `SYSDBA` privilege.

You can connect as `SYSDBA` with a password file or with operating system authentication.

Note: Unlike SQL*Plus, RMAN does not require that you specify the `SYSDBA` privilege when connecting to a database. Because all RMAN database connections require `SYSDBA` privilege, RMAN always implicitly attempts to connect with this privilege.

If the target database uses password files, then 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 as `SYSDBA` with a net service name.

If you connect to the database using operating system authentication, you must set the environment variable specifying the Oracle SID. For example, to set the SID to `trgt` at the UNIX command line enter:

```
% ORACLE_SID=trgt; export ORACLE_SID
```

A `SYSDBA` privilege is not required when connecting to the recovery catalog. Note that you must grant the `RECOVERY_CATALOG_OWNER` role to the schema owner.

For automatic auxiliary instances, RMAN ensures that you have `SYSDBA` privilege when it sets up the instance.

See Also: *Oracle Database Administrator's Guide* to learn how to authenticate users on a database

Connecting to the Target Database from the Command Line

To connect to the target database from the operating system command line, enter the connection as in the following examples:

```
# example of operating system authentication
% rman TARGET / NOCATALOG
# example of Oracle Net authentication
% rman TARGET SYS/oracle@trgt NOCATALOG
```

You can also start RMAN without specifying NOCATALOG or CATALOG as follows:

```
# example of operating system authentication
% rman TARGET /
# example of Oracle Net authentication
% rman TARGET SYS/oracle@trgt
```

If you do not specify NOCATALOG on the command line, and if you do not specify CONNECT CATALOG after RMAN has started, then RMAN begins to work in NOCATALOG mode the first time that you run a command that requires the use of the RMAN repository.

Note: Once you have executed a command that uses the RMAN repository in NOCATALOG mode, you must exit and restart RMAN to be able to connect to a recovery catalog.

If you connect to the target database on the operating system command line, then you can begin executing commands after the RMAN prompt is displayed.

Connecting to the Target Database from the RMAN Prompt

If you start RMAN without connecting to the target database, then you must issue a CONNECT TARGET command at the RMAN prompt to connect to a target database and begin performing useful work. This example connects to a target database using operating system authentication:

```
% rman
RMAN> CONNECT TARGET /
```

This example connects to the target database with database-level credentials:

```
% rman
RMAN> connect target SYS/oracle@trgt
```

Setting Up a Database for RMAN Backup

Backing up a database with RMAN is meant to be simple. Therefore, for most of the parameters required for RMAN backup, there are sensible defaults which will let you do basic backup and recovery without extensive setup or configuration.

However, when implementing an RMAN-based backup strategy, you can use RMAN more effectively if you understand the more common options available to you. Many of these can be set in the RMAN environment on a persistent basis, so that you do not have to specify the same options every time you issue a command.

The following discussion presents how RMAN's default behaviors can be changed for your backup and recovery environment and strategies, and introduces the major settings available to you and their most common possible values.

This section includes the following topics:

- [Persistent Configuration Settings: Controlling RMAN Behavior](#)
- [Configuring the Default Backup Type for Disk Backups](#)

- [Configuring Disk Devices and Channels](#)
- [Configuring Tape Devices and Channels](#)
- [Configuring Compressed Backupsets as Default for Tape or Disk](#)
- [Configuring Control File and Server Parameter File Autobackup](#)

Persistent Configuration Settings: Controlling RMAN Behavior

To simplify ongoing use of RMAN for backup and recovery, the RMAN lets you set a number of persistent configuration settings for each target database. These settings control many aspects of RMAN's behavior when working with that database, such as backup retention policy, default destinations for backups to tape or disk, default backup device type (tape or disk), and so on.

The default values for these configuration settings let you use RMAN effectively without changing any of them. However, as you develop a more advanced backup and recovery strategy, you will have to change these settings to implement that strategy. Use the RMAN `SHOW` and `CONFIGURE` commands to view and change the RMAN configuration settings.

See Also: *Oracle Database Backup and Recovery Reference* for `CONFIGURE` syntax

Displaying Current RMAN Configuration Settings: SHOW

The `SHOW` command is used to display the current value of one or all of RMAN's configured settings, as well as whether those commands are currently set to their default value.

After you connect to the target database and recovery catalog (if you use one), run the `SHOW` command with the name of a setting you wish to view. For example:

```
RMAN> SHOW RETENTION POLICY;  
RMAN> SHOW DEFAULT DEVICE TYPE;
```

The `SHOW ALL` command displays the current settings of all parameters that you can set with the `CONFIGURE` command. The output includes both parameters you have changed and parameters that are still set to the default.

To view all configured settings, run the RMAN `SHOW ALL` command, as in this example:

```
RMAN> SHOW ALL;      # shows all CONFIGURE settings, both user-entered and default
```

Sample output for `SHOW ALL` follows:

```
RMAN configuration parameters are:  
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 3 DAYS;  
CONFIGURE BACKUP OPTIMIZATION ON;  
CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default  
CONFIGURE CONTROLFILE AUTOBACKUP ON;  
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE SBT_TAPE TO '%F'; #  
default  
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F'; # default  
CONFIGURE DEVICE TYPE 'SBT_TAPE' PARALLELISM 2 BACKUP TYPE TO COMPRESSED  
BACKUPSET;  
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default  
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE SBT_TAPE TO 1; # default  
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default  
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE SBT_TAPE TO 1; # default
```

```

CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE CHANNEL DEVICE TYPE 'SBT_TAPE' PARMS 'SBT_
LIBRARY=mylibrary.disksbt,ENV=(BACKUP_PARAM=value)';
CONFIGURE MAXSETSIZE TO UNLIMITED; # default
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/disk1/oracle/dbs/snapcf_ev.f'; # default

```

The configuration is displayed as the series of RMAN commands required to re-create the configuration. You can save the output of `SHOW ALL` into a text file and use that command file to re-create the configuration on the same or a different target database.

See Also: *Oracle Database Backup and Recovery Reference* for `SHOW` syntax

Restoring Default RMAN Configuration Settings: `CONFIGURE... CLEAR`

You can return any setting to its default value by using `CONFIGURE... CLEAR`, as in these examples:

```

RMAN> CONFIGURE BACKUP OPTIMIZATION CLEAR;
RMAN> CONFIGURE RETENTION POLICY CLEAR;
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK CLEAR;

```

Configuring the Default Device Type for Backups

By default, RMAN sends all backups to an operating system specific directory on disk. You can also configure RMAN to make backups to media such as tape.

After configuring an `sbt` (that is, tape or media management) device according to the instructions in your media management vendor documentation, you can make the media manager the default device:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

After configuring the default device type, the backups produced by any `BACKUP` command which does not specify the destination device type are directed to the configured default device type. For example, the following commands would produce a series of backups on tape:

```

CONFIGURE DEFAULT DEVICE TYPE TO sbt;
BACKUP DATABASE;
BACKUP DATAFILE 3;
BACKUP DATABASE PLUS ARCHIVELOG;

```

To configure disk as the default device for backups, you can either use `CONFIGURE... CLEAR` to restore the default setting, or explicitly configure the default device as shown:

```
CONFIGURE DEFAULT DEVICE TYPE TO DISK;
```

Note: You can always direct backups to a specific device type, DISK or SBT, using the DEVICE TYPE clause of the BACKUP command. For example:

```
BACKUP DEVICE TYPE sbt DATABASE;  
BACKUP DEVICE TYPE DISK DATABASE;
```

See *Oracle Database Backup and Recovery Reference* for more details on using the BACKUP command with the DEVICE TYPE clause.

Configuring the Default Backup Type for Disk Backups

You can configure backup sets or image copies as the default, using either of the following commands:

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY; # image copies  
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO BACKUPSET; # uncompressed
```

The default for backups to disk, as with tape, is backup set. Note that there is no analogous option for media manager devices, because RMAN can only write backups to media manager devices as backup sets.

Configuring Compressed Backupsets as Default for Tape or Disk

You can configure RMAN to use compressed backupsets by default on a particular device type, by using the CONFIGURE DEVICE TYPE command with the BACKUP TYPE TO COMPRESSED BACKUPSET option, as shown in the following examples.

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COMPRESSED BACKUPSET;  
RMAN> CONFIGURE DEVICE TYPE sbt BACKUP TYPE TO COMPRESSED BACKUPSET;
```

To disable compression, use the CONFIGURE DEVICE TYPE command with arguments specifying your other desired settings, but omitting the COMPRESSED keyword, as in the following examples:

```
RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO BACKUPSET;  
RMAN> CONFIGURE DEVICE TYPE sbt BACKUP TYPE TO BACKUPSET;
```

Configuring Disk Devices and Channels

RMAN **channels** (connections to server sessions on the target database) perform all RMAN tasks. By default, RMAN allocates one disk channel for all operations.

The following command configures RMAN to write disk backups to the /backup directory (refer to ["Backing Up Database Files and Archived Logs with RMAN"](#) on page 4-7):

```
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/backup/ora_df%t_s%s_s%p';
```

The format specifier %t is replaced with a four byte time stamp, %s with the backup set number, and %p with the backup piece number.

You can also configure an Automatic Storage Management disk group as your destination, as in the following example:

```
CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '+dgroup1';
```

Note: By configuring an explicit format for disk channels, you direct backups away from the flash recovery area, if you have configured one. You lose the disk space management capabilities of the flash recovery area.

Configuring Tape Devices and Channels

Some media managers require configuration settings that are passed by including a PARS string in the CONFIGURE command, as follows:

```
CONFIGURE CHANNEL DEVICE TYPE sbt PARS='ENV=mml_env_settings';
```

The contents of your PARS string depend on your media management library. Refer to your media management vendor's documentation for details.

You can configure parallelism settings, backup set compression and other options for the SBT device using CONFIGURE DEVICE TYPE SBT. (These settings for the device type are set independently of the channel configuration for your device set in the previous example.) The following command configures two sbt channels for use in RMAN jobs:

```
CONFIGURE DEVICE TYPE sbt PARALLELISM 2;
old RMAN configuration parameters:
CONFIGURE DEVICE TYPE 'SBT_TAPE' BACKUP TYPE TO COMPRESSED BACKUPSET PARALLELISM
1;
new RMAN configuration parameters:
CONFIGURE DEVICE TYPE 'SBT_TAPE' PARALLELISM 2 BACKUP TYPE TO COMPRESSED
BACKUPSET;
new RMAN configuration parameters are successfully stored

RMAN> configure device type sbt backup type to backupset;

old RMAN configuration parameters:
CONFIGURE DEVICE TYPE 'SBT_TAPE' PARALLELISM 2 BACKUP TYPE TO COMPRESSED
BACKUPSET;
new RMAN configuration parameters:
CONFIGURE DEVICE TYPE 'SBT_TAPE' BACKUP TYPE TO BACKUPSET PARALLELISM 2;
new RMAN configuration parameters are successfully stored
```

Note that the CONFIGURE commands used in this example to set parallelism and backup type do not affect the values of settings not specified, that is, the default backup type of compressed backupset was not changed by the CONFIGURE DEVICE TYPE SBT PARALLELISM 1 command.

Configuring Control File and Server Parameter File Autobackup

RMAN can be configured to automatically back up the control file and server parameter file whenever the database structure metadata in the control file changes and whenever a backup record is added. The autobackup enables RMAN to recover the database even if the current control file, catalog, and server parameter file are lost.

Because the filename for the autobackup uses a well-known format, RMAN can search for it without access to a repository, and then restore the server parameter file. After you have started the instance with the restored server parameter file, RMAN can restore the control file from an autobackup. After you mount the control file, the RMAN repository is available and RMAN can restore the datafiles and find the archived redo log.

You can enable the autobackup feature by running this command:

```
CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

You can disable the feature by running this command:

```
CONFIGURE CONTROLFILE AUTOBACKUP OFF;
```

See Also:

- *Oracle Database Backup and Recovery Advanced User's Guide* for a conceptual overview of control file autobackups
- *Oracle Database Backup and Recovery Reference* for CONFIGURE syntax

Configuring the Control File Autobackup Format

By default, the format of the autobackup file for all configured devices is the substitution variable %F. This variable format translates into *c-YYYYMMDD-QQ*, where:

- *YYYYMMDD* stands for the DBID.
- *YYYYMMDD* is a time stamp of the day the backup is generated
- *QQ* is the hex sequence that starts with 00 and has a maximum of FF

You can change the default format by using the following command, where *deviceSpecifier* is any valid device such as DISK or sbt, and *string* must contain the substitution variable %F (and no other substitution variables) and is a valid handle for the specified device:

```
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT  
FOR DEVICE TYPE deviceSpecifier TO 'string';
```

For example, you can run the following command:

```
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT  
FOR DEVICE TYPE DISK TO '?/oradata/cf_%F';
```

The following example configures the autobackup to write to an Automatic Storage Management disk group:

```
CONFIGURE CONTROLFILE AUTOBACKUP  
FOR DEVICE TYPE DISK TO '+dgroup1/%F';
```

To clear control file autobackup formats for a device, use the following commands:

```
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK CLEAR;  
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE sbt CLEAR;
```

If you have set up a flash recovery area for your database, you can direct control file autobackups to the flash recovery area by clearing the control file autobackup format for disk.

Overriding the Configured Control File Autobackup Format

The SET CONTROLFILE AUTOBACKUP FORMAT command, which you can specify either within a RUN block or at the RMAN prompt, overrides the configured autobackup format in the current session only.

The order of precedence is:

1. SET CONTROLFILE AUTOBACKUP FORMAT (within a RUN block)

2. SET CONTROLFILE AUTOBACKUP FORMAT (at RMAN prompt)
3. CONFIGURE CONTROLFILE AUTOBACKUP FORMAT

The following example shows how the two forms of SET CONTROLFILE AUTOBACKUP FORMAT interact:

```

RMAN> SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO 'controlfile_%F';
RMAN> BACKUP AS COPY DATABASE;
RMAN> RUN {
    SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/tmp/%F.bck';
    BACKUP AS BACKUPSET DEVICE TYPE DISK DATABASE;
}

```

The first SET CONTROLFILE AUTOBACKUP FORMAT controls the name of the control file autobackup until the RMAN client exits, overriding any configured control file autobackup format. The SET CONTROLFILE AUTOBACKUP FORMAT in the RUN block overrides the SET CONTROLFILE AUTOBACKUP FORMAT outside the RUN block for the duration of the RUN block.

Setting Up a Flash Recovery Area for RMAN

As explained in "[Automatic Disk-Based Backup and Recovery: The Flash Recovery Area](#)" on page 1-11, the flash recovery area feature lets you set up a location on disk where the database can create and manage a variety of backup and recovery-related files on your behalf.

Using a flash recovery area simplifies the ongoing administration of your database by automatically naming recovery-related files, retaining them as long as they are needed for restore and recovery activities, and deleting them when they are no longer needed to restore your database and space is needed for some other backup and recovery-related purpose.

Your long-term backup and recovery administration can be greatly simplified by using a flash recovery area. Use of the flash recovery area is strongly recommended. You may want to set up a flash recovery area as one of the first steps in implementing your backup strategy.

This section outlines the functions of the flash recovery area, identifies the files stored there, explains the rules for how files are managed there, and introduces the most important configuration options.

Choosing a Location for the Flash Recovery Area

When setting up a flash recovery area, you must choose a location (a directory or Automatic Storage Management disk group) to hold the files. The flash recovery area cannot be stored on a raw file system.

You must also determine a **disk quota** for the flash recovery area, the maximum space to be used for all files stored there. You must choose a location large enough to accommodate the required disk quota. When the disk space limit is approached, Oracle can delete nonessential files to make room for new files, subject to the limitations of the retention policy.

The flash recovery area should be on a separate disk from the **database area**, where active database files such as datafiles, control files, and online redo logs are stored. Keeping the flash recovery area on the same disk as the database area exposes you to loss of both your live database files and backups in the event of a media failure.

Note: There are special considerations for choosing a location for the flash recovery area in a RAC environment. The location must be on a cluster file system, ASM or a shared directory configured through NFS. The location and disk quota must be the same on all instances.

Flash Recovery Area, Automatic Storage Management, and Oracle Managed Files

The flash recovery area is closely related to and can be used in conjunction with two other Oracle features: Oracle Managed Files and Automatic Storage Management.

Oracle Managed Files (OMF) is a service that automates naming, location, creation and deletion of database files such as control files, redo log files, datafiles and others, based on a few initialization parameters. It can simplify many aspects of the DBA's work by eliminating the need to devise your own policies for such details.

The flash recovery area is built on top of OMF, so the flash recovery area can be stored anywhere Oracle-managed files can. Oracle Managed Files can be used on top of a traditional file system supported by the host operating system (for example, VxFS or ODM).

The flash recovery area can also be used with Oracle's Automatic Storage Management (ASM). ASM consolidates storage devices into easily managed disk groups and provides benefits such as mirroring and striping without requiring a third-party logical volume manager.

Even if you choose not to set up the flash recovery area in ASM storage, you can still use Oracle Managed Files to manage your backup files in an ASM disk group. You will lose one of the major benefits of the flash recovery area, the automatic deletion of files no longer needed to meet your recoverability goals as space is needed for more recent backups. However, the other automatic features of OMF will still function.

Note: When storing backup files, using OMF on top of Automatic Storage Management without using a flash recovery area is supported but discouraged. It is awkward to directly manipulate files under Automatic Storage Management.

Files That Can Be Stored in the Flash Recovery Area

The files in the flash recovery area can be classified as **permanent** or **transient**. The only permanent files (assuming these are configured to be stored in the flash recovery area) are multiplexed copies of the current control file and online redo logs. These cannot be deleted without causing the instance to fail. All other files are transient, because Oracle will generally eventually delete these files, at some point after they become obsolete under the retention policy or have been backed up to tape. Transient files include archived redo logs, datafile copies, control file copies, control file autobackups, and backup pieces.

Note: The Oracle Flashback Database feature, which provides an convenient alternative to point-in-time recovery, generates flashback logs, which are also considered transient files and must be stored in the flash recovery area. However, unlike other transient files, flashback logs cannot be backed up to other media. They are automatically deleted as space is needed for other files in the flash recovery area. See [Chapter 5, "Data Protection with Restore Points and Flashback Database"](#) for more details about Oracle Flashback Database.

Planning the Size of the Flash Recovery Area

The larger the flash recovery area is, the more useful it becomes. Ideally, the flash recovery area should be large enough to contain all of the following files:

- A copy of all datafiles
- Incremental backups, as used by your chosen backup strategy
- Online redo logs
- Archived redo logs not yet backed up to tape
- Control files
- Control file autobackups (which include copies of the control file and SPFILE)

If providing this much space is impractical, it is best to create an area large enough to keep a backup of the most important tablespaces and all the archived logs not yet copied to tape. At an absolute minimum, the flash recovery area must be large enough to contain the archived logs that have not been copied to tape.

To determine the disk quota and current disk usage in the flash recovery area, query the view `V$RECOVERY_FILE_DEST`.

Formulas for estimating a useful flash recovery area size depend upon several factors in your backup and recovery strategy:

- Whether your database has a small or large number of data blocks that change frequently;
- Whether you store backups only on disk, or on disk and tape;
- Whether you use a redundancy-based retention policy, or a recovery window-based retention policy;
- Whether you plan to use Flashback Database or guaranteed restore points as alternatives to point-in-time recovery to recover from logical errors.

Specific formulas are provided for many different backup scenarios in [Appendix A, "RMAN-Based Disk and Tape Backup Strategies: Scenarios"](#). If you want to use Flashback Database, you must add extra space to the flash recovery area, as discussed in ["Sizing the Flash Recovery Area to Include Flashback Logs"](#) on page 5-10.

Setting Initialization Parameters for Size and Location of the Flash Recovery Area

To enable the flash recovery area, you must set the two initialization parameters `DB_RECOVERY_FILE_DEST_SIZE` (which specifies the disk quota, or maximum space to use for flash recovery area files for this database) and `DB_RECOVERY_FILE_DEST` (which specifies the location of the flash recovery area).

Note:

- `DB_RECOVERY_FILE_DEST_SIZE` must be set *before* `DB_RECOVERY_FILE_DEST`.
 - In a RAC database, all instances must have the same values for these parameters.
-
-

Initialization parameters can be specified by any of the following means:

- Include them initialization parameter file of the target database
- Specify them with the SQL statement `ALTER SYSTEM SET`
- Use the Database Configuration Assistant to set them

See Also:

- *Oracle Database Administrator's Guide* for details on setting and changing database initialization parameters

To find out the current flash recovery area location, query `V$RECOVERY_FILE_DEST`.

See Also: *Oracle Database SQL Reference* for `ALTER SYSTEM` syntax

Flash Recovery Area Size: `DB_RECOVERY_FILE_DEST_SIZE`

This initialization parameter specifies the maximum storage in bytes of data to be used by the flash recovery area for this database.

Note: The value specified does not include certain kinds of disk overhead:

- Block 0 or the OS block header of each Oracle file is not included in this size. Allow an extra 10% for this data when computing the actual disk usage required for the flash recovery area.
 - `DB_RECOVERY_FILE_DEST_SIZE` does not indicate the real size occupied on disk when the underlying file system is mirrored, compressed, or in some other way affected by overhead not known to Oracle. For example, if you can configure the flash recovery area on a normal redundancy (2-way mirrored) ASM disk group, each file of X bytes occupies 2X bytes on the ASM disk group. In such a case, `DB_RECOVERY_FILE_DEST_SIZE` must be set to no more than 1/2 the size of the disks for the ASM disk group. Likewise, when using a high redundancy (three-way mirrored) ASM disk group, `DB_RECOVERY_FILE_DEST_SIZE` must be no more than 1/3 the size of the disks in the disk group, and so on.
-
-

Flash Recovery Area Location: Initialization Parameter `DB_RECOVERY_FILE_DEST`

This parameter specifies a valid disk location for file creation, which can be a directory on a file system, or Automatic Storage Management disk group.

Sharing a Flash Recovery Area Among Multiple Databases

Multiple databases can have the same value for `DB_RECOVERY_FILE_DEST`, if one of the following conditions is met:

- No two databases for which `DB_UNIQUE_NAME` is specified have the same value for `DB_UNIQUE_NAME`.
- For those databases where no `DB_UNIQUE_NAME` is provided, no two databases have the same value for `DB_NAME`.

When databases share a single flash recovery area in this fashion, the file system or ASM disk group holding the flash recovery area should be large enough to hold all of the recovery files for all of the databases. Add all the values for `DB_RECOVERY_FILE_DEST_SIZE` for the different databases, and then allow for any overhead such as mirroring or compression in allocating actual disk space, as described in ["Flash Recovery Area Size: DB_RECOVERY_FILE_DEST_SIZE"](#) on page 3-16.

Restrictions on Initialization Parameters When Using Flash Recovery Area

Using a flash recovery area has implications for some other initialization parameters:

- You cannot use the `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` parameters to specify redo log archive destinations. You must instead use the newer `LOG_ARCHIVE_DEST_n` parameters. See *Oracle Database Reference* for details on the semantics of the `LOG_ARCHIVE_DEST_n` parameters.
- `LOG_ARCHIVE_DEST_10` is implicitly set to `USE_DB_RECOVERY_FILE_DEST` (meaning that archived redo log files will be sent to the flash recovery area) if you create a recovery area and do not set any other local archiving destinations.
- Oracle Corporation recommends that `DB_RECOVERY_FILE_DEST` not be the same as `DB_CREATE_FILE_DEST` or any of the `DB_CREATE_ONLINE_LOG_DEST_n` parameters. A warning will appear in the alert log if `DB_RECOVERY_FILE_DEST` is the same as any of the other parameters listed here.

Adding a Flash Recovery Area to an Existing Database

To create a flash recovery area, you can set the necessary parameters in the initialization parameter file (PFILE) and restart the database. You can also set the `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE` initialization parameters using `ALTER SYSTEM`, to add a flash recovery area to an open database, as shown in this example.

Note: `DB_RECOVERY_FILE_DEST_SIZE` must be set before `DB_RECOVERY_FILE_DEST`.

1. After you start SQL*Plus and connect to the database, set the size of the flash recovery area. For example, set it to 10 GB:

```
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST_SIZE = 10G SCOPE=BOTH SID='*';
```

Set `SCOPE` to `BOTH` make the change both in memory and the server parameter file. (Setting `SID` to "*" has no effect in a single-instance database; in a RAC database it causes the change to take effect across all instances.)

2. Set the location of the flash recovery area. For example, if the location is the file system directory `/disk1/flash_recovery_area`, then you can do the following:

```
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST = '/disk1/flash_recovery_area'
SCOPE=BOTH SID='*';
```

If the flash recovery area location is an Automatic Storage Management disk group named `disk1`, for example, then you can do the following:

```
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST = '+disk1' SCOPE=BOTH SID='*';
```

Using V\$RECOVERY_FILE_DEST and V\$FLASH_RECOVERY_AREA_USAGE

The `V$RECOVERY_FILE_DEST` and `V$FLASH_RECOVERY_AREA_USAGE` views can help you determine whether you have allocated enough space for your flash recovery area.

Query the `V$RECOVERY_FILE_DEST` view to find out the current location, disk quota, space in use, space reclaimable by deleting files, and total number of files in the flash recovery area.

```
SQL> SELECT * FROM V$RECOVERY_FILE_DEST;
```

NAME	SPACE_LIMIT	SPACE_USED	SPACE_RECLAIMABLE	NUMBER_OF_FILES
/mydisk/rcva	5368709120	109240320	256000	28

Query the `V$FLASH_RECOVERY_AREA_USAGE` view to find out the percentage of the total disk quota used by different types of files, and how much space for each type of file can be reclaimed by deleting files that are obsolete, redundant, or already backed up to tape.

```
SQL> SELECT * FROM V$FLASH_RECOVERY_AREA_USAGE;
```

FILE_TYPE	PERCENT_SPACE_USED	PERCENT_SPACE_RECLAIMABLE	NUMBER_OF_FILES
CONTROLFILE	0	0	0
ONLINELOG	2	0	22
ARCHIVELOG	4.05	2.01	31
BACKUPPIECE	3.94	3.86	8
IMAGECOPY	15.64	10.43	66
FLASHBACKLOG	.08	0	1

See the *Oracle Database Reference* for more details on the `V$RECOVERY_FILE_DEST` and `V$FLASH_RECOVERY_AREA_USAGE` views.

Disabling the Flash Recovery Area

To disable the flash recovery area, set the `DB_RECOVERY_FILE_DEST` initialization parameter to a null string. For example, use this SQL*Plus statement to change the parameter on a running database:

```
ALTER SYSTEM SET DB_RECOVERY_FILE_DEST='' SCOPE=BOTH SID="*";
```

The database will no longer provide the space management features of the flash recovery area for the files stored in the old `DB_RECOVERY_FILE_DEST` location. The files will still be known to the RMAN repository, however, and available for backup and restore activities.

Configuring the Backup Retention Policy

The backup retention policy specifies which backups must be retained to meet your data recovery requirements. This policy can be based on a recovery window (the

maximum number of days into the past for which you can recover) or redundancy (how many copies of each backed-up file to keep).

Use the `CONFIGURE` command to set the retention policy.

See Also:

- *Oracle Database Backup and Recovery Reference* for `CONFIGURE` syntax

Configuring a Recovery Window-Based Retention Policy

The `RECOVERY WINDOW` parameter of the `CONFIGURE` command specifies the number of days between the current time and the earliest point of recoverability. RMAN does not consider any full or level 0 incremental backup as obsolete if it falls within the recovery window. Additionally, RMAN retains all archived logs and level 1 incremental backups that are needed to recover to a random point within the window.

Run the `CONFIGURE RETENTION POLICY` command at the RMAN prompt. This example ensures that you can recover the database to any point within the last week:

```
RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

RMAN does not automatically delete backups rendered obsolete by the recovery window. Instead, RMAN shows them as `OBSOLETE` in the `REPORT OBSOLETE` output and in the `OBSOLETE` column of `V$BACKUP_FILES`. RMAN deletes obsolete files if you run the `DELETE OBSOLETE` command.

Configuring a Redundancy-Based Retention Policy

The `REDUNDANCY` parameter of the `CONFIGURE RETENTION POLICY` command specifies how many backups of each datafile and control file that RMAN should keep. In other words, if the number of backups for a specific datafile or control file exceeds the `REDUNDANCY` setting, then RMAN considers the extra backups as obsolete. The default retention policy is `REDUNDANCY=1`.

As you produce more backups, RMAN keeps track of which ones to retain and which are obsolete. RMAN retains all archived logs and incremental backups that are needed to recover the nonobsolete backups.

Assume that you make a backup of datafile 7 on Monday, Tuesday, Wednesday, and Thursday. You now have four backups of the datafile. If `REDUNDANCY` is 2, then the Monday and Tuesday backups are obsolete. If you make another backup on Friday, then the Wednesday backup becomes obsolete.

Run the `CONFIGURE RETENTION POLICY` command at the RMAN prompt, as in the following example:

```
CONFIGURE RETENTION POLICY TO REDUNDANCY 3;
```

Showing the Current Retention Policy

You can view the currently configured retention policy with the `SHOW RETENTION POLICY` command. Sample output follows:

```
RMAN> SHOW RETENTION POLICY;
```

```
RMAN configuration parameters are:
CONFIGURE RETENTION POLICY TO REDUNDANCY 3;
```

Disabling the Retention Policy

When you disable the retention policy, RMAN does not consider any backup as obsolete.

To disable the retention policy, run this command:

```
CONFIGURE RETENTION POLICY TO NONE;
```

Configuring the retention policy to NONE is not the same as clearing it. Clearing it returns it to its default setting of REDUNDANCY=1, whereas NONE disables it completely.

If you disable the retention policy and run `REPORT OBSOLETE` or `DELETE OBSOLETE` without passing a retention policy option to the command, RMAN issues an error because no retention policy exists to determine which backups are obsolete.

Note: If you are using a flash recovery area, then you should not run your database with the retention policy disabled. If files are never considered obsolete, then a file can only be deleted from the flash recovery area if it has been backed up to some other disk location or to a tertiary storage device such as tape. It is quite likely that all of the space in your recovery area will be used. This interferes with the normal operation of your database as described in "[When Space is Not Available in the Flash Recovery Area](#)" on page 3-21.

How Oracle Manages Disk Space in the Flash Recovery Area

Oracle does not delete eligible files from the flash recovery area until the space must be reclaimed for some other purpose. The effect is that files recently moved to tape are often still available on disk for use in recovery. The recovery area can thus serve as a kind of cache for tape. Once the flash recovery area is full, Oracle automatically deletes eligible files to reclaim space in the flash recovery area as needed.

When Files are Eligible for Deletion from the Flash Recovery Area

There are relatively simple rules governing when files become eligible for deletion from the flash recovery area:

- Permanent files are never eligible for deletion.
- Files that are obsolete under the configured retention policy are eligible for deletion.
- Transient files that have been copied to tape are eligible for deletion.
- In a Data Guard environment, archived redo log deletion policy governs when archived redo log files can be deleted from the flash recovery area. See *Oracle Data Guard Concepts and Administration* for details on archived redo log deletion policy.

Note: Exactly which of the eligible files will be deleted to satisfy a space request is unpredictable. The rules governing the selection of specific files for deletion are likely to change between releases and are dependent upon your configuration. The safe and reliable way to control deletion of files from the flash recovery area is to change your retention policy. If you wish to increase the likelihood that files moved to tape are also retained on disk to minimize expected restore and recovery times, increase the flash recovery area quota.

When Space is Not Available in the Flash Recovery Area

If, for instance, the RMAN retention policy requires keeping a set of backups larger than the flash recovery area disk quota, or if the retention policy is set to NONE, then the flash recovery area can fill completely with no reclaimable space.

The database issues a warning alert when reclaimable space is less than 15% and a critical alert when reclaimable space is less than 3%. To warn the DBA of this condition, an entry is added to the alert log and to the `DBA_OUTSTANDING_ALERTS` table (used by Enterprise Manager). However, the database continues to consume space in the flash recovery area until there is no reclaimable space left.

When the recovery area is completely full, the error you will receive is:

```
ORA-19809: limit exceeded for recovery files
ORA-19804: cannot reclaim nnnnn bytes disk space from mmmmmm limit
```

where *nnnnn* is the number of bytes required and *mmmmmm* is the disk quota for the flash recovery area.

The database handles a flash recovery area with insufficient reclaimable space just as it handles a disk full condition. Often, the result is a database hang, but not always. For example, if the flash recovery area is one of your mandatory redo log archiving destinations, and the database cannot archive a new log because the recovery area is full, then the archiver may, depending on your configuration, retry archiving periodically until space is freed in the recovery area. For information on how a particular feature of Oracle responds to a disk full condition, see the documentation for that feature.

See Also:

- ["Resolving a Full Flash Recovery Area"](#) on page 8-18 for more on how to address situations where the flash recovery area frequently fills to capacity and there are no files eligible for deletion

Configure Flash Recovery Area for Disk-Based Backups: Example

In this example the database is configured as follows:

- Archived logs and RMAN backups are stored in the flash recovery area.
- The control file and online redo log copies are stored in directories in the file system outside the flash recovery area.
- Datafiles are expected to be no larger than 3GB in size. No more than 4GB of archived redo log files should be retained.

The backup strategy in this example is based on incremental backups. The control file will be automatically backed up to the flash recovery area.

The flash recovery area is sized to 10GB, room enough for control file autobackups, a whole database level 0 incremental backup (which consists of image copies of the 3GB of datafiles), plus several incremental level 1 backups.

To implement this strategy, the parameter file contains the following entries:

```
DB_NAME=sample
# set location for current datafiles:
DB_CREATE_FILE_DEST = '/u02/oradata/wrk_area'
# set location for control files and online redo logs:
DB_CREATE_ONLINE_LOG_DEST_1 = '/u03/oradata/wrk_area'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u04/oradata/wrk_area'
# set flash recovery area location and size
DB_RECOVERY_FILE_DEST = '/u01/oradata/rcv_area'
DB_RECOVERY_FILE_DEST_SIZE = 10G
```

Because the parameter file does not set any of the `LOG_ARCHIVE_DEST_n` parameters, Oracle sends archived logs to the flash recovery area only.

Once the target database is started, the following RMAN commands configure the retention policy, backup optimization, and the control file autobackup:

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 1;
RMAN> CONFIGURE BACKUP OPTIMIZATION ON;
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

Any disk-based backups are now directed to your flash recovery area.

See also: ["Scripting Disk-Only Backups"](#) on page A-1 for examples of backup jobs you could run in this environment.

Create a Database with Multiplexed Files in the Flash Recovery Area: Scenario

Assume that you want to create a database with the following properties:

- The control files, datafiles, and online redo logs are stored as Oracle managed files in a single file system directory.
- One multiplexed copy of the control file is kept in the flash recovery area
- Multiplexed copies of the online redo logs are kept in the flash recovery area
- Redo log files are archived both to the flash recovery area and another file system, separate from the work area
- RMAN backups are directed to the flash recovery area by default

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for more detailed information about file creation in the flash recovery area

To create a database with a flash recovery area:

1. Create a PFILE for the database, including the initialization parameters required to use a flash recovery area and direct online and archived logs, a copy of the control file, . Assume that you set the following:

```
# set DB_NAME
DB_NAME=sample
# set destination for OMF datafiles, control file and online redo logs
DB_CREATE_FILE_DEST = /u01/oradata/wrk_area/
# set log archiving destinations to a file system location
```

```
# and the flash recovery area
LOG_ARCHIVE_DEST_1 = 'LOCATION=/arc_dest1'
LOG_ARCHIVE_DEST_2 = 'LOCATION=USE_DB_RECOVERY_FILE_DEST'
# multiplexed copies of control file and online logs in flash recovery area
# rman backups also go here
DB_RECOVERY_FILE_DEST = 'LOCATION=/u01/oradata/rcv_area'
DB_RECOVERY_FILE_DEST_SIZE = 10G
```

The `DB_CREATE_FILE_DEST` parameter sets the default directory for all datafiles, online logs, and control files. Another copy of the control file and online logs is created in the flash recovery area.

2. After you set the initialization parameters, create the database. For example, start SQL*Plus and enter:

```
SQL> CREATE DATABASE sample;
```

The preceding statement has the following effects:

- Datafiles are created as Oracle managed files in `DB_CREATE_FILE_DEST`.
- Because no `LOGFILE` clause was included, online log groups are created by default. Each group has two members, one in `DB_CREATE_FILE_DEST`, the other in `DB_RECOVERY_FILE_DEST`.
- Because the `CONTROL_FILES` parameter was not set, Oracle creates a control file in `DB_CREATE_FILE_DEST` (primary) and `DB_RECOVERY_FILE_DEST` (multiplexed copy). On a Linux system, the filenames might look like the following:

```
/u02/oradata/wrk_area/SAMPLE/controlfile/o1_mf_3ajeikm_.ctl #primary
ctlfile
/u01/oradata/rcv_area/SAMPLE/controlfile/o1_mf_6adjkid_.ctl #ctl file copy
/u02/oradata/wrk_area/SAMPLE/logfile/o1_mf_0_orrm3lz_.log #log grp1, mem 1
/u01/oradata/rcv_area/SAMPLE/logfile/o1_mf_1_ixfvm8w9).log #log grp 1 mem 2
/u02/oradata/wrk_area/SAMPLE/logfile/o1_mf_2_2xyz16am_.log # log grp2, mem
1
/u01/oradata/rcv_area/SAMPLE/logfile/o1_mf_2_q89tmp28_.log #log grp 2, mem
2
```

- Oracle uses `LOG_ARCHIVE_DEST_1` and `LOG_ARCHIVE_DEST_2` as destinations for archiving the redo logs. Archived redo log files are created in the flash recovery area because `LOG_ARCHIVE_DEST_2` is configured to be the flash recovery area.

Because you enabled a local redo log archiving destination, `LOG_ARCHIVE_DEST_10` is *not* implicitly set to the flash recovery area.

The archived redo log files in the flash recovery area are given Oracle-managed filenames that are *not* based on the `LOG_ARCHIVE_FORMAT` parameter. For example, if you generate an archived log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

An archived log file is created in the primary archiving location, as well as the following flash recovery area subdirectory: `/u01/oradata/rcv_area/SAMPLE/archivelog/YYYY_MM_DD`

where `YYYY_MM_DD` is the creation date format.

3. You may want to create more online redo log groups for this database. To do so, use the `ALTER DATABASE ADD LOGFILE` statement in SQL*Plus. When no file name is specified, it creates another log file member in the destinations already

specified for online logs, including the flash recovery area. For example, the following statement creates a new log group with two members: one in DB_CREATE_FILE_DEST and another in DB_RECOVERY_FILE_DEST:

```
ALTER DATABASE ADD LOGFILE;
```

Creating a Database with Only Archived Logs in the Flash Recovery Area: Scenario

Assume that you want to create a database in which the control files, datafiles, and online redo logs are Oracle managed files in a single file system directory. Additionally, you want to do the following:

- Archive each redo log to the flash recovery area (and *only* to the flash recovery area)
- Send RMAN backups to the flash recovery area by default.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for more detailed information about file creation in the flash recovery area

To create a database with a flash recovery area:

1. Set the relevant initialization parameters. Assume that you set the following:

```
DB_NAME=sample
DB_CREATE_FILE_DEST = '/u02/oradata/wrk_area'
DB_RECOVERY_FILE_DEST = '/u01/oradata/rcv_area'
DB_RECOVERY_FILE_DEST_SIZE = 10G
# if you set the following parameters, then the online redo logs *and*
# current control file are located here
DB_CREATE_ONLINE_LOG_DEST_1 = '/u03/oradata/wrk_area'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u04/oradata/wrk_area'
```

The DB_CREATE_FILE_DEST parameter sets the default file system directory for the datafiles. The DB_CREATE_ONLINE_LOG_DEST_n parameter sets the default file system directories for the online redo logs and control files. DB_RECOVERY_FILE_DEST sets the file system directory for archived logs.

2. After you set the initialization parameters, create the database. For example, start SQL*Plus and enter:

```
CREATE DATABASE sample;
```

No multiplexed copies of the online redo logs or control files are created in the flash recovery area.

3. Because you enabled a flash recovery area, Oracle automatically sets LOG_ARCHIVE_DEST_10 to the flash recovery area. The filenames in the flash recovery area are given Oracle managed filenames that are *not* based on the LOG_ARCHIVE_FORMAT parameter. For example, generate an archived log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

On Linux, the preceding statement creates an archived log in a flash recovery area subdirectory: /u01/oradata/rcv_area/SAMPLE/archivelog/YYYY_MM_DD, where YYYY_MM_DD is the creation date format.

4. If you need to add more online redo log groups, execute the ALTER DATABASE ADD LOGFILE statement. When no file name is specified, it creates another log file

member in each `DB_CREATE_ONLINE_LOG_DEST_n` location (but not the flash recovery area). For example, enter the following:

```
ALTER DATABASE ADD LOGFILE;
```

On Linux, the preceding statement creates one member in `/u03/oradata/wrk_area/SAMPLE/logfile` and another member in `/u04/oradata/wrk_area/SAMPLE/logfile`. On other platforms, the specific file and directory names are platform-dependent.

Backing Up Databases Using RMAN

This chapter illustrates how to perform the most common backup tasks and implement backup strategies using Recovery Manager.

This chapter includes the following topics:

- [Overview of RMAN Backups](#)
- [Backing Up Database Files and Archived Logs with RMAN](#)
- [RMAN Incremental Backups](#)
- [Using RMAN to Validate Database Files](#)
- [Overview of Reporting on Backups and the RMAN Repository](#)
- [Listing RMAN Backups, Archived Logs, and Database Incarnations](#)
- [Reporting on Backups and Database Schema](#)

Note: Detailed scenarios and scripts for disk-only and disk-and-tape backup strategies are provided in [Appendix A, "RMAN-Based Disk and Tape Backup Strategies: Scenarios"](#). Strategies are categorized by levels of database update activity, disk space to be allocated for backups, and database recoverability requirements. RMAN commands for initial setup and daily and weekly backups are provided for each strategy. Once you have familiarized yourself with the different forms of backup supported by RMAN, including incrementally updated backups and the use of the flash recovery area, refer to the examples in [Appendix A](#) for help in planning your own backup strategy.

Many of the backup techniques described in this chapter are also used in the Oracle-provided backup strategy provided by Enterprise Manager and described in *Oracle Database 2 Day DBA*.

Restore Points and Flashback Database provide functionality that is complementary to the backup capabilities here, in that they allow you to prepare your database so that you can later reverse unwanted changes. Consider incorporating them into your backup and recovery strategy. See [Chapter 4, "Backing Up Databases Using RMAN"](#) for more information on these features.

Overview of RMAN Backups

Backing up all or part of your database is accomplished by using the `BACKUP` command from within the RMAN client.

RMAN uses the configured settings and channels for your database, the record of previous backups in the RMAN repository and the control file's record of the structure of your database to determine an efficient set of specific steps to perform in response to a BACKUP command, and then carries out those steps.

In many cases, once your database has been configured in accordance with your backup strategy, an RMAN backup of your entire database can often be performed with the following simple command:

```
RMAN> BACKUP DATABASE;
```

Files That RMAN Can Back Up

RMAN's BACKUP command supports backing up the following types of files:

- Database files, including datafiles, control files, and the server parameter file (SPFILE)
- Archived redo logs
- Other backups created by RMAN, including such as datafile and control file image copies, and backup sets containing SPFILEs, control files, datafiles and archived logs

Although the database depends on other types of files for operation, such as network configuration files, password files, and the contents of the Oracle home, these files cannot be backed up with RMAN. Likewise, some features of Oracle, such as external tables, may depend upon files other than the datafiles, control files, and redo log for storing information. RMAN cannot back up these files. Use some non-RMAN backup solution for any files not in the preceding list.

About RMAN Backup Formats: Image Copies and Backup Sets

RMAN backups can be stored in one of two formats: as **image copies** or as **backup sets**.

About Image Copies

An image copy is a bit-for-bit duplicate of a database file, identical to a copy made with an operating system command. (RMAN-created image copies are, however, recorded in the RMAN repository, unlike file copies created using operating system-level commands.)

Image copy backups can only be created on disk. RMAN can create image copies of datafiles and datafile copies, control files and control file copies, archived redo logs, and backup pieces. RMAN creates image copies when the AS COPY option is used with the BACKUP command.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for more detailed information about RMAN's handling of image copies

About Backup Sets

RMAN can also store backup information a logical structure called a **backup set**. A backup set contains the data from one or more datafiles, archived redo logs, or control files or SPFILE. (Datafiles and archive logs cannot be mixed together in the same backup set.) You can also back up existing backup sets into another backup set.

A backup set consists of one or more files in an RMAN-specific format, known as **backup pieces**. By default, a backup set consists of one backup piece. For example, you can back up ten datafiles into a single backup set consisting of a single backup piece (that is, one backup piece will be produced as output, the backup set consists of the single file containing the backup piece, and the backup piece and the backup set that contains it will be recorded in the RMAN repository). A file cannot be split across backup sets.

Note: The backup set is the smallest unit of a backup. RMAN only records backup sets in the repository that complete successfully. There is no such thing as a partial backup set. You cannot usefully manipulate individual backup pieces.

Only RMAN can create or restore from backup sets. When multiple files are backed up into the same backup set, they are read concurrently and their data is multiplexed together.

Backup sets are the only type of backup that RMAN supports on media manager devices such as tapes. Backup sets can also be created on disk. By default, RMAN creates backups both on both disk and on tape as backup sets.

When backing up datafiles to backup sets, RMAN is able to skip some datafile blocks that do not currently contain data, reducing the size of backup sets and the time required to create them. This behavior, known as **unused block compression**, means that backups of datafiles as backup sets are generally smaller than image copy backups and take less time to write. This behavior is fundamental to how RMAN writes datafiles into backup pieces, and cannot be disabled. For more information about how and when unused blocks are skipped, see *Oracle Database Backup and Recovery Reference*.

RMAN also supports binary compression of backup sets, where the backup set contents are compressed before being written to disk using a compression algorithm tuned for compression of datafiles and archived log files. The use of binary compression is described in "[Using Compressed Backupsets for RMAN Backup](#)" on page 4-6.

About RMAN Full and Incremental Datafile Backups

RMAN backups of datafiles can be either full datafile backups, or incremental backups.

A full backup of a datafile is a backup that includes every used data block in the file. If a full datafile backup is created as an image copy, the entire file contents are reproduced exactly. (If backing the datafile up to a backup set, then unused blocks may be skipped, as described in "[About Backup Sets](#)" on page 4-2)

An incremental backup of a datafile captures images of blocks in the datafile changed since a specific point in time, usually the time of a previous incremental backup. Incremental backups are always stored as backup sets. The resulting backup sets are generally smaller than full datafile backups, unless every block in the datafile is changed. RMAN can only create incremental backups of datafiles, not of archived redo log files or other files.

During media recovery, RMAN uses the block images from incremental backups, to update changed blocks to their contents at the SCN where the block was created in a single step. Without incremental backups, all changes must be applied one at a time from the archived redo logs. Recovery using incremental backups is therefore much

faster than applying changes one at a time from the archived redo logs. Incremental backups also capture changes to data blocks made by `NOLOGGING` operations, which are not recorded in the redo log. For these reasons, whenever incremental backups are available for use in media recovery, RMAN uses incremental backups instead of archived logs.

Specifying Options Affecting Output of the RMAN BACKUP Command

If you specify only the minimum required options for an RMAN command such as `BACKUP DATABASE`, RMAN determines the destination device, locations for backup output, and tag for the backup automatically based on your configured environment and built-in RMAN defaults. However, you can provide arguments to `BACKUP` to override these defaults and configured settings. The most common are described in the following sections:

- [Specifying Output Device Type for RMAN BACKUP](#)
- [Specifying Image Copy or Backup Set Output for RMAN BACKUP to Disk](#)
- [Specifying Output File Locations for RMAN BACKUP](#)
- [Specifying Tags for RMAN BACKUP](#)

Specifying Output Device Type for RMAN BACKUP

The `BACKUP` command takes a `DEVICE TYPE` clause that specifies whether to back up to disk or to an SBT device, as shown in this example:

```
BACKUP DATABASE DEVICE TYPE DISK;
```

When `BACKUP` is used without a `DEVICE TYPE` clause, the backup is stored on the configured default device (`DISK` or `sbt`) as set using `CONFIGURE DEFAULT DEVICE TYPE` as described in ["Configuring the Default Device Type for Backups"](#) on page 3-9.

Specifying Image Copy or Backup Set Output for RMAN BACKUP to Disk

As noted, RMAN can create backups on disk as image copies or as backup sets. Setting the default output type is described in ["Configuring the Default Backup Type for Disk Backups"](#) on page 3-10. You can override this default with the `AS COPY` or `AS BACKUPSET` clauses to the `BACKUP` command. To back up your data to disk as image copies, use `BACKUP AS COPY`:

```
BACKUP AS COPY DEVICE TYPE DISK DATABASE;
```

To back up your data into backup sets, use the `BACKUP AS BACKUPSET` clause. You can let backup sets be created on the configured default device, or direct them specifically to disk or tape, as shown here:

```
BACKUP AS BACKUPSET DATABASE;  
BACKUP AS BACKUPSET DEVICE TYPE DISK DATABASE;  
BACKUP AS BACKUPSET DEVICE TYPE SBT DATABASE;
```

Specifying Output File Locations for RMAN BACKUP

RMAN generates unique filenames for the output from the backup based on the following set of rules, listed in order of priority:

- You can specify a `FORMAT` clause with the individual `BACKUP` command to direct the output to a specific location, as shown here:

```
BACKUP DATABASE FORMAT='/tmp/backup_%U';
```

Backups in this case are stored with generated unique filenames in the location `/tmp/backups/`. Note that the `%U`, used to generate a unique string at that point in the filename, is required.

You can also use the `FORMAT` clause to name an ASM disk group as backup destination, as shown in this example:

```
RMAN> BACKUP DATABASE FORMAT '+dgroup1'; # sets an ASM disk group
```

No `%U` is required in this instance, because ASM generates unique file names as needed.

- If a `FORMAT` setting is configured for the specific channel used for the backup, then this setting controls the generated filename.
- If a `FORMAT` setting is configured for the device type used for the backup, then this setting controls the generated filename.
- If the backup is a disk backup and a flash recovery area is configured, then the backup is stored under an automatically generated name in the flash recovery area.
- If none of the other conditions in this list apply, then the default location and filename format are platform-specific.

Specifying Tags for RMAN BACKUP

RMAN attaches an identifier called a **tag** to every backup it creates, as a way of identifying that backup that can be used with later RMAN commands.

When you use the `BACKUP` command, you can specify the to be assigned to the backups taken with a particular command. You could use this tag to identify backups taken at a given time, such as `2003_year_end`. You can also use a tag repeatedly for a series of backups created over time. For example, you might label a series of weekly incremental backups with a tag such as `weekly_incremental`.

In practice, tags are often used to distinguish a series of backups created as part of a single strategy, such as an incremental backup strategy. Many forms of the `BACKUP` command let you associate a tag with a backup, and many `RESTORE` and `RECOVER` commands let you specify a tag to restrict which backups to use in the `RESTORE` or `RECOVER` operation.

If you do not specify a tag with a `BACKUP` command, then RMAN generates a unique tag and assigns it to the backups created by the command.

To specify a tag to identify a group of backups, use the `TAG` clause of the `BACKUP` command. For example, enter:

```
RMAN> BACKUP DATABASE TAG = 'weekly_backup'; # gives the backup a tag identifier
```

If a backup is not assigned a tag using the `TAG` clause to the `BACKUP` command, then RMAN generates a tag automatically and assigns it to the backup.

Note: The characters used in a tag must be limited to the characters that are legal in filenames on the target file system. For example, ASM does not support the use of the - character in the filenames it uses internally, so a tag including a - (such as `weekly-incremental`) is not a legal tag name if you are storing backups in ASM disk groups.

See Also: *Oracle Database Backup and Recovery Reference* for the default format description in `BACKUP . . . TAG`

Using Compressed Backupsets for RMAN Backup

For any use of the `BACKUP` command that creates backupsets, you can take advantage of RMAN's support for binary compression of backupsets, by using the `AS COMPRESSED BACKUPSET` option to the `BACKUP` command. The resulting backupsets are compressed using an algorithm optimized for efficient compression of Oracle database files. No extra uncompression steps are required during recovery if you use RMAN's integrated compression.

The primary disadvantage of using RMAN binary compression is performance overhead during backups and restores. Backup performance while creating compressed backupsets is CPU bound. If you have more than one CPU, you can use increased parallelism to run jobs on multiple CPUs and thus improve performance.

Note: If you are backing up to tape and your tape device performs its own compression, you should not use both RMAN backupset compression and the media manager vendor's compression. In most instances you will get better results using the media manager's compression. See the discussion of tuning RMAN's tape backup performance in *Oracle Database Backup and Recovery Advanced User's Guide* for details.

This example backs up the entire database and archived logs to the configured default backup destination (disk or tape), producing compressed backupsets:

```
BACKUP AS COMPRESSED BACKUPSET DATABASE PLUS ARCHIVELOG;
```

This example backs up several datafiles to the default device, using binary compression:

```
BACKUP AS COMPRESSED BACKUPSET DATAFILE 1,2,4;
```

Predictably, creating compressed backupsets imposes significant extra CPU overhead during backup and restore, which can slow the backup process. The performance penalty may be worth paying, however, in some circumstances:

- If you are using disk-based backups and disk space in your flash recovery area or other disk-based backup destination is limited
- If you are performing your backups to some device over a network and reduced network bandwidth is more important than CPU usage
- If you are using some archival backup media such as CD or DVD, where reducing backup sizes saves on media costs and archival storage

For more information on performance considerations when using binary compression of backup sets, see the description of the `AS COMPRESSED BACKUPSET` option of the `BACKUP` command, in *Oracle Database Backup and Recovery Reference*.

Backing Up Database Files and Archived Logs with RMAN

This section contains these topics:

- [Making Consistent and Inconsistent Backups with RMAN](#)
- [Specifying Options Affecting Output of the RMAN BACKUP Command](#)
- [Making Whole Database Backups with RMAN](#)
- [Backing Up Individual Tablespaces with RMAN](#)
- [Backing Up Individual Datafiles and Datafile Copies with RMAN](#)
- [Backing Up Control Files with RMAN](#)
- [Backing Up Server Parameter Files with RMAN](#)
- [Backing Up Archived Redo Logs with RMAN](#)

Making Consistent and Inconsistent Backups with RMAN

A **consistent backup** of the database is one taken when the database is in a consistent state, that is, one taken after the database has been shut down normally (using `SHUTDOWN NORMAL`, `SHUTDOWN IMMEDIATE` or `SHUTDOWN TRANSACTIONAL`). At this point, all changes in the redo log have been applied to the datafiles. If you mount the database and take a backup at this point, then you can restore the database from this backup at a later date and open it without performing media recovery.

Any backup taken when the database has not been shut down normally is an **inconsistent backup**. When a database is restored from an inconsistent backup, Oracle must perform media recovery before the database can be opened, applying any pending changes from the redo logs.

As long as your database is running in `ARCHIVELOG` mode, and you back up your archived redo log files as well as your datafiles, inconsistent backups can be the foundation for a sound backup and recovery strategy. Inconsistent backups are an important part of the backup strategy for most databases, because they offer superior availability. For example, backups taken while the database is still open are inconsistent backups.

Note: When performing user-managed backups, taking online backups required that you place your datafiles into backup mode using the `ALTER DATABASE/TABLESPACE BEGIN BACKUP` statement. RMAN does not require the use of backup mode for the creation of online backups. Do not use `ALTER DATABASE/TABLESPACE BEGIN BACKUP` before an RMAN backup.

Making Whole Database Backups with RMAN

You can perform whole database backups with the database mounted or open. To perform a whole database backup, from the RMAN prompt, use the `BACKUP`

DATABASE command. The simplest form of the command requires no parameters, as shown in this example:

```
RMAN> BACKUP DATABASE;
```

This example shows the procedure for taking a whole database backup to the default destination:

```
RMAN> BACKUP DATABASE; # uses automatic channels to make backup
RMAN> SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT'; # switches logs and archives all
logs
```

By archiving the logs immediately after the backup, you ensure that you have a full set of archived logs through the time of the backup. This guarantees that you can perform media recovery after restoring this backup.

Backing Up Individual Tablespaces with RMAN

You can backup one or more individual tablespaces with the `BACKUP TABLESPACE` command. You can use this command when the database is mounted or open.

To back up a tablespace:

After starting RMAN, run the `BACKUP TABLESPACE` command at the RMAN prompt. This example backs up the `users` and `tools` tablespaces to tape, using the `MAXSETSIZE` parameter to specify that no backup set should be greater than 10 MB:

```
BACKUP DEVICE TYPE sbt MAXSETSIZE = 10M TABLESPACE users, tools;
```

Oracle translates the tablespace name internally into a list of datafiles.

Backing Up Individual Datafiles and Datafile Copies with RMAN

You can back up datafiles and datafile copies when the database is mounted or open.

Backing Up Datafiles

With RMAN connected to the target database, use the `BACKUP DATAFILE` command to back up individual datafiles. You can specify the datafiles by name or number.

This example uses an `sbt` channel to back up datafiles 1 through 4 and a datafile copy stored at `/tmp/system01.dbf` to tape:

```
BACKUP DEVICE TYPE sbt
  DATAFILE 1,2,3,4
  DATAFILECOPY '/tmp/system01.dbf';
```

If `CONFIGURE CONTROLFILE AUTOBACKUP` is ON, then RMAN writes the current control file and SPFILE to a separate autobackup piece. Otherwise, these files are automatically included in the backup set that contains datafile 1.

Backing Up Datafile Copies

Use the `BACKUP DATAFILECOPY` command to back up datafile copies. Datafile copies exist on disk only.

To back up a datafile copy:

While connected to the target database, run the `BACKUP DATAFILECOPY` command at the RMAN prompt. This example backs up datafile `/tmp/system01.dbf` to tape:

```
BACKUP DEVICE TYPE sbt DATAFILECOPY '/tmp/system01.dbf';
```

Backing Up Control Files with RMAN

You can back up the control file when the database is mounted or open. RMAN uses a snapshot control file to ensure a read-consistent version. If `CONFIGURE CONTROLFILE AUTOBACKUP` is ON (by default it is OFF), then RMAN automatically backs up the control file and server parameter file after every backup and after database structural changes. The control file autobackup contains metadata about the previous backup, which is crucial for disaster recovery.

If the autobackup feature is not set, then you must manually back up the control file in one of the following ways:

- Run `BACKUP CURRENT CONTROLFILE`
- Include a backup of the control file within any backup by using the `INCLUDE CURRENT CONTROLFILE` option of the `BACKUP` command
- Back up datafile 1, because RMAN automatically includes the control file and SPFILE in backups of datafile 1

Note: If the control file block size is not the same as the block size for datafile 1, then the control file cannot be written into the same backup set as the datafile. RMAN writes the control file into a backup set by itself if the block size is different.

A manual backup of the control file is not the same as a control file autobackup. In manual backups, only RMAN repository data for backups within the current RMAN session is in the control file backup, and a manually backed-up control file cannot be automatically restored.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* to learn more about control file autobackups

Including the Current Control File in a Backup of Other Files

To include the current control file in a backup, specify the `INCLUDE CURRENT CONTROLFILE` option after specifying the backup object. In this example, the default configured channel is to an `sbt` device. This command backs up tablespace `users` to tape and includes the current control file in the backup:

```
BACKUP DEVICE TYPE sbt TABLESPACE users INCLUDE CURRENT CONTROLFILE;
```

If the autobackup feature is enabled, then RMAN also creates an autobackup of the control file after the `BACKUP TABLESPACE` command completes, so that the control file autobackup contains the record of the backup that was taken.

Backing Up the Current Control File Manually

After starting RMAN, run the `BACKUP CURRENT CONTROLFILE` command. This example backs up the current control file to the default disk device and assigns a tag:

```
BACKUP CURRENT CONTROLFILE TAG = mondaypmbackup;
```

If the control file autobackup feature is enabled, then RMAN makes two control file backups in this example: the explicit control file backup (`BACKUP CURRENT CONTROLFILE`) and the autobackup of the control file and server parameter file.

Backing Up a Control File Copy

This example creates a control file backup with the `BACKUP CONTROLFILECOPY` command.

To back up a control file copy:

After starting RMAN, run the `BACKUP CONTROLFILECOPY` command at the RMAN prompt. This example creates the control file copy `'/tmp/control01.ct1'` on disk and then backs it up to tape:

```
BACKUP AS COPY CURRENT CONTROLFILE FORMAT '/tmp/control01.ct1';
BACKUP DEVICE TYPE sbt CONTROLFILECOPY '/tmp/control01.ct1';
```

Backing Up Server Parameter Files with RMAN

As explained in "[Backing Up Control Files with RMAN](#)" on page 4-9, RMAN automatically backs up the current server parameter file in certain cases. The `BACKUP SPFILE` command backs up the parameter file explicitly. For example:

```
BACKUP DEVICE TYPE sbt SPFILE;
```

The SPFILE that is backed up is the one currently in use by the instance. If the instance is started with a client-side initialization parameter file, then RMAN does not back up anything when this command is used.

Backing Up Archived Redo Logs with RMAN

Archived redo logs are the key to successful media recovery. Back them up regularly. You can back up logs with `BACKUP ARCHIVELOG`, or back up logs while backing up datafiles and control files by specifying `BACKUP . . . PLUS ARCHIVELOG`.

Backing Up Archived Redo Log Files with `BACKUP ARCHIVELOG`

To back up archived redo logs, use the `BACKUP ARCHIVELOG` command at the RMAN prompt. This example uses a configured disk or `sbt` channel to back up one copy of each log sequence number for all archived redo logs:

```
BACKUP ARCHIVELOG ALL;
```

Even if your redo logs are being archived to multiple destinations and you use RMAN to back up archived redo logs, RMAN selects only one copy of the archived redo log file to include in the backup set. (Since logs with the same log sequence number are identical, there is no need to include more than one copy.)

You can also specify a range of archived redo logs by time, SCN, or log sequence number, as in the following example:

```
BACKUP ARCHIVELOG
  FROM TIME 'SYSDATE-30' UNTIL TIME 'SYSDATE-7';
```

Automatic Online Redo Log Switches During Backups of Archived Logs When taking a backup of archived redo logs that includes the most recent log (that is, a `BACKUP ... ARCHIVELOG` command is run without the `UNTIL` or `SEQUENCE` option) if the database is open, then before beginning the backup, RMAN will switch out of the current online redo log group, and all online redo logs that have not yet been archived, up to and including the redo log group that was current when the command was issued. This ensures that the backup contains all redo that was generated prior to the start of the command.

Using BACKUP ARCHIVELOG with DELETE INPUT or DELETE ALL INPUT You can specify the `DELETE INPUT` or `DELETE ALL INPUT` clauses for the `BACKUP ARCHIVELOG` command to delete archived logs after they are backed up, eliminating the separate step of manually deleting the archived redo logs. With `DELETE INPUT`, RMAN only deletes the specific copy of the archived redo log chosen for the backup set. With `DELETE ALL INPUT`, RMAN will delete each backed-up archived redo log file from all log archiving destinations.

For example, assume that you archive to `/arc_dest1`, `/arc_dest2`, and `/arc_dest3`, and you run the following command:

```
BACKUP DEVICE TYPE sbt
  ARCHIVELOG ALL
  DELETE ALL INPUT;
```

In this case RMAN backs up only one copy of each log sequence number in these directories, and then deletes *all* copies of any log that it backed up from the archiving destinations. If you had specified `DELETE INPUT` rather than `DELETE ALL INPUT`, then RMAN would only delete the specific archived redo log files that it backed up (for example, it would delete the archived redo log files in `/arc_dest1` if those were the files used as the source of the backup, but it would leave the contents of the `/arc_dest2` and `/arc_dest3` intact).

If you issue `BACKUP ARCHIVELOG ALL` or `BACKUP ARCHIVELOG LIKE '...'`, and there are no archived redo log files to back up, then RMAN does not signal an error.

Backing Up Logs with BACKUP ... PLUS ARCHIVELOG

You can add archived redo logs to a backup of other files by using the `BACKUP ... PLUS ARCHIVELOG` clause. Adding `BACKUP ... PLUS ARCHIVELOG` causes RMAN to do the following:

1. Runs the `ALTER SYSTEM ARCHIVE LOG CURRENT` command.
2. Runs `BACKUP ARCHIVELOG ALL`. Note that if backup optimization is enabled, then RMAN skips logs that it has already backed up to the specified device.
3. Backs up the rest of the files specified in `BACKUP` command.
4. Runs the `ALTER SYSTEM ARCHIVE LOG CURRENT` command.
5. Backs up any remaining archived logs generated during the backup.

This guarantees that datafile backups taken during the command are recoverable to a consistent state.

To back up archived redo logs with BACKUP ... PLUS ARCHIVELOG:

After starting RMAN, run the `BACKUP ... PLUS ARCHIVELOG` command at the RMAN prompt. This example backs up the database and all archived logs:

```
BACKUP DEVICE TYPE sbt
  DATABASE PLUS ARCHIVELOG;
```

Note: If backup optimization is enabled, then RMAN skips backups of archived logs that have already been backed up to the specified device.

RMAN Incremental Backups

RMAN incremental backups back up only datafile blocks that have changed since a specified previous backup. You can make incremental backups of databases, individual tablespaces or datafiles.

The goal of an incremental backup is to back up only those data blocks that have changed since a previous backup.

The primary reasons for making incremental backups part of your strategy are:

- For use in a strategy based on incrementally updated backups, where these incremental backups are used to periodically roll forward an image copy of the database
- To reduce the amount of time needed for daily backups
- To save network bandwidth when backing up over a network
- To get adequate backup performance when the aggregate tape bandwidth available for tape write I/Os is much less than the aggregate disk bandwidth for disk read I/Os
- To be able to recover changes to objects created with the `NOLOGGING` option. For example, direct load inserts do not create redo log entries and their changes cannot be reproduced with media recovery. They do, however, change data blocks and so are captured by incremental backups.
- To reduce backup sizes for `NOARCHIVELOG` databases. Instead of making a whole database backup every time, you can make incremental backups.

As with full backups, if you are in `ARCHIVELOG` mode, you can make incremental backups if the database is open; if the database is in `NOARCHIVELOG` mode, then you can only make incremental backups after a consistent shutdown.

See Also: *Oracle Database Concepts* for more information about `NOLOGGING` mode

One effective strategy is to make incremental backups to disk, and then back up the resulting backup sets to a media manager with `BACKUP AS BACKUPSET`. The incremental backups are generally smaller than full backups, which limits the space required to store them until they are moved to tape. Then, when the incremental backups on disk are backed up to tape, it is more likely that tape streaming can be sustained because all blocks of the incremental backup are copied to tape. There is no possibility of delay due to time required for RMAN to locate changed blocks in the datafiles.

Incremental Backup Algorithm

Each data block in a datafile contains a system change number (SCN), which is the SCN at which the most recent 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. If the SCN in the input data block is greater than or equal to the checkpoint SCN of the parent, then RMAN copies the block.

Note that if you enable the block change tracking feature, RMAN can refer to the change tracking file to identify changed blocks in datafiles without scanning the full contents of the datafile. Once enabled, block change tracking does not alter how you take or use incremental backups, other than offering increased performance. See

"Improving Incremental Backup Performance: Change Tracking" on page 4-19 for more details about enabling block change tracking.

Level 0 and Level 1 Incremental Backups

Incremental backups can be either level 0 or level 1. A level 0 incremental backup, which is the base for subsequent incremental backups, copies all blocks containing data, backing the datafile up into a backup set just as a full backup would. The only difference between a level 0 incremental backup and a full backup is that a full backup is never included in an incremental strategy.

A level 1 incremental backup can be either of the following types:

- A **differential backup**, which backs up all blocks changed after the most recent incremental backup at level 1 or 0
- A **cumulative backup**, which backs up all blocks changed after the most recent incremental backup at level 0

Incremental backups are differential by default.

Note: Cumulative backups are preferable to differential backups when recovery time is more important than disk space, because during recovery each differential backup must be applied in succession. Use cumulative incremental backups instead of differential, if enough disk space is available to store cumulative incremental backups.

The size of the backup file depends solely upon the number of blocks modified and the incremental backup level.

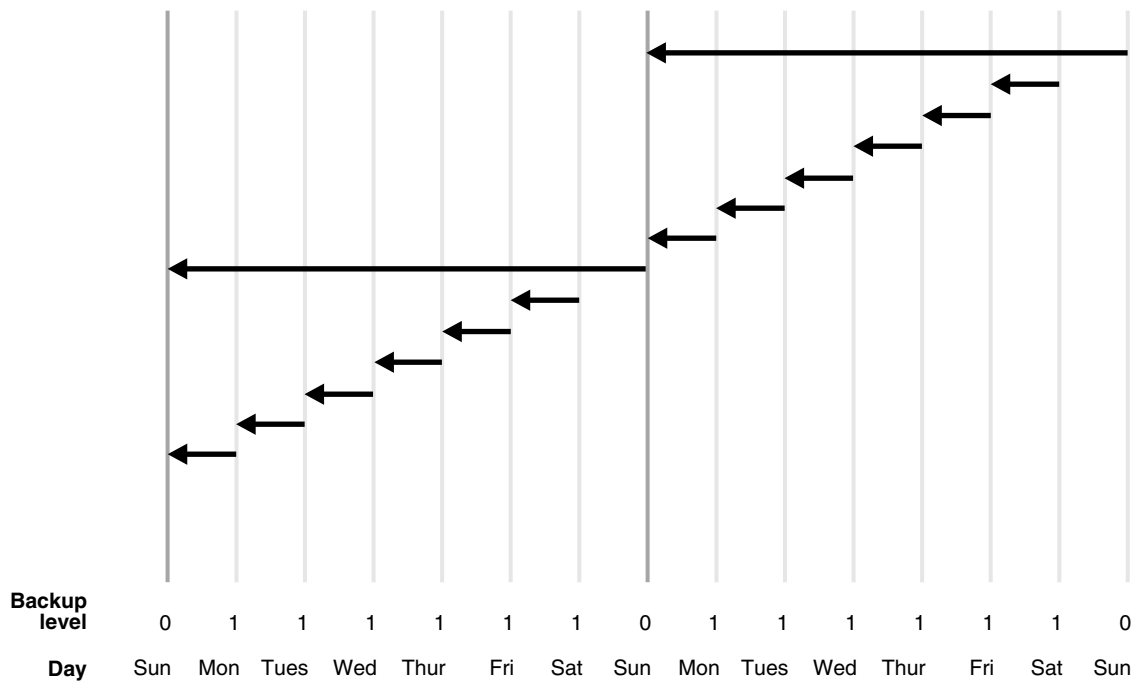
Differential Incremental Backups

In a differential level 1 backup, RMAN backs up all blocks that have changed since the most recent cumulative or differential incremental backup, whether at level 1 or level 0. RMAN determines which level 1 backup occurred most recently and backs up all blocks modified after that backup. If no level 1 is available, RMAN copies all blocks changed since the level 0 backup.

The following command performs a level 1 differential incremental backup of the database:

```
RMAN> BACKUP INCREMENTAL LEVEL 1 DATABASE;
```

If no level 0 backup is available, then the behavior depends upon the compatibility mode setting. If compatibility is $\geq 10.0.0$, RMAN copies all blocks changed since the file was created, and stores the results as a level 1 backup. In other words, the SCN at the time the incremental backup is taken is the file creation SCN. If compatibility $< 10.0.0$, RMAN generates a level 0 backup of the file contents at the time of the backup, to be consistent with the behavior in previous releases.

Figure 4-1 Differential Incremental Backups (Default)

In the example shown in [Figure 4-1](#), the following occurs:

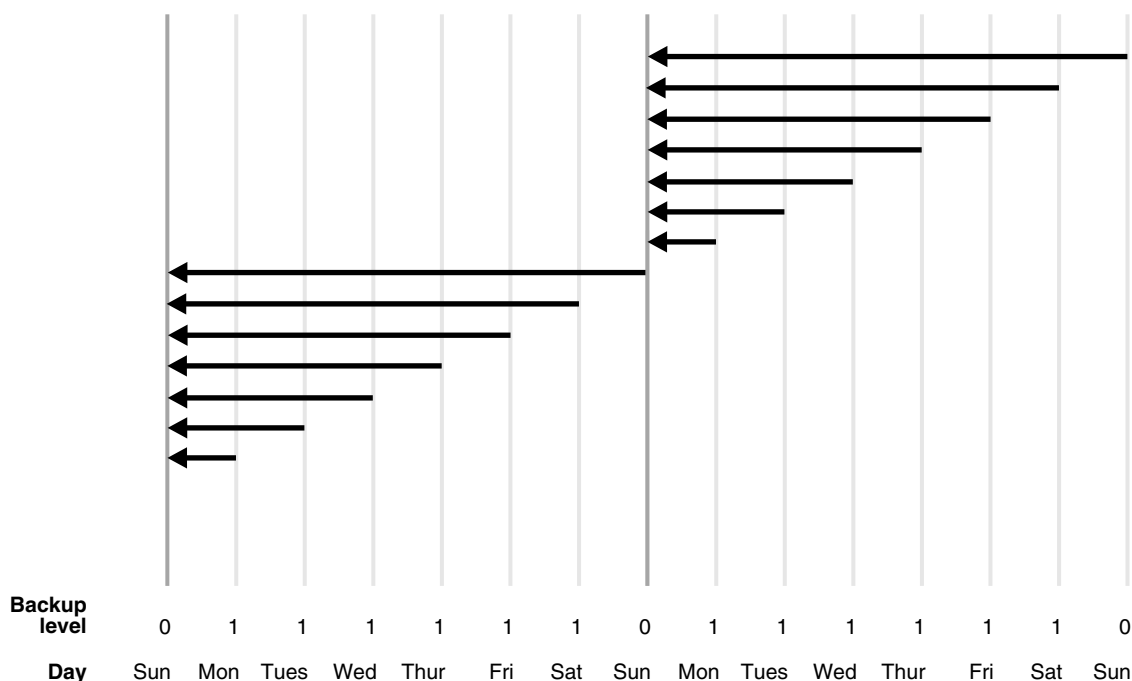
- Sunday
 - An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.
- Monday - Saturday
 - On each day from Monday through Saturday, a differential incremental level 1 backup backs up all blocks that have changed since the most recent incremental backup at level 1 or 0. So, the Monday backup copies blocks changed since Sunday level 0 backup, the Tuesday backup copies blocks changed since the Monday level 1 backup, and so forth.
- The cycle is repeated for the next week.

Cumulative Incremental Backups

In a cumulative level 1 backup, RMAN backs up all the blocks used since the most recent level 0 incremental 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.

The following command performs a cumulative level 1 incremental backup of the database:

```
BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE; # blocks changed since level 0
```

Figure 4–2 Cumulative Incremental Backups

In the example shown in [Figure 4–2](#), the following occurs:

- Sunday
 - An incremental level 0 backup backs up *all* blocks that have ever been in use in this database.
- Monday - Saturday
 - A cumulative incremental level 1 backup copies all blocks changed since the most recent level 0 backup. Because the most recent level 0 backup was created on Sunday, the level 1 backup on each day Monday through Saturday backs up all blocks changed since the Sunday backup.
- The cycle is repeated for the next week.

Basic Incremental Backup Strategy

Choose a backup scheme according to an acceptable MTTR (mean time to recover). For example, you can implement a three-level backup scheme so that a full or level 0 backup is taken monthly, a cumulative level 1 is taken weekly, and a differential level 1 is taken daily. In this scheme, you never have to apply more than a day's worth of redo for complete recovery.

When deciding how often to take full or 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 differential or cumulative backups to a 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.

Making Incremental Backups: BACKUP INCREMENTAL

After starting RMAN, run the `BACKUP INCREMENTAL` command at the RMAN prompt. This example makes a level 0 incremental backup of the database:

```
BACKUP INCREMENTAL LEVEL 0 DATABASE;
```

This example makes a differential level 1 backup of the `SYSTEM` tablespace and datafile `tools01.dbf`. It will only back up those data blocks changed since the most recent level 1 or level 0 backup:

```
BACKUP INCREMENTAL LEVEL 1
  TABLESPACE SYSTEM
  DATAFILE 'ora_home/oradata/trgt/tools01.dbf';
```

This example makes a cumulative level 1 backup of the tablespace `users`, backing up all blocks changed since the most recent level 0 backup.

```
BACKUP INCREMENTAL LEVEL = 1 CUMULATIVE
  TABLESPACE users;
```

Incrementally Updated Backups: Rolling Forward Image Copy Backups

Oracle's Incrementally Updated Backups feature lets you avoid the overhead of taking full image copy backups of datafiles, while providing the same recovery advantages as image copy backups.

At the beginning of a backup strategy, RMAN creates an image copy backup of the datafile. Then, at regular intervals, such as daily, level 1 incremental backups are taken, and applied to the image copy backup, rolling it forward to the point in time when the level 1 incremental was created.

During restore and recovery of the database, RMAN can restore from this incrementally updated copy and then apply changes from the redo log, with the same results as restoring the database from a full backup taken at the SCN of the most recently applied incremental level 1 backup.

A backup strategy based on incrementally updated backups can help minimize time required for media recovery of your database. For example, if you run scripts to implement this strategy daily, then at recovery time, you never have more than one day of redo to apply.

Incrementally Updated Backups: A Basic Example

To create incremental backups for use in an incrementally updated backups strategy, you must use the `BACKUP... FOR RECOVER OF COPY WITH TAG` form of the `BACKUP` command. How the command works is best understood in the context of an example script that would implement the strategy.

This script, run on a regular basis, is all that is required to implement a strategy based on incrementally updated backups:

```
RUN {
  RECOVER COPY OF DATABASE WITH TAG 'incr_update';
```

```

BACKUP INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG 'incr_update'
DATABASE;
}

```

The syntax used in the script does not, however, make it clear how the strategy works. To understand the script and the strategy, it is necessary to understand the effects of these two commands when no datafile copies or incremental backups exist.

- The `BACKUP INCREMENTAL LEVEL 1... FOR RECOVER OF COPY WITH TAG...` command does not actually always create a level 1 incremental backup. If there is no level 0 image copy backup of an particular datafile, then executing this command creates an image copy backup of the datafile on disk with the specified tag instead of creating the level 1 backup.

Note: Even when the `BACKUP INCREMENTAL LEVEL 1 ... FOR RECOVER OF COPY` command is used with `DEVICE TYPE SBT` to create a backup on tape, the first time it is used it creates the image copy on disk, and does not write any backup on tape. Subsequent incremental level 1 backups can be created on tape once the image copy is on disk.

Thus, the first time the script runs, it creates the image copy of the datafile needed to begin the cycle of incremental updates. In the second run and all subsequent runs, it produces level 1 incremental backups of the datafile.

- The `RECOVER COPY OF DATABASE WITH TAG...` command causes RMAN to apply any available incremental level 1 backups to a set of datafile copies with the specified tag.

If there is no incremental backup or no datafile copy, the command generates a message but does not generate an error.

The first time the script runs, this command has no effect, because there is neither a datafile copy nor a level 1 incremental backup.

The second time the script runs, there is a datafile copy (created by the first `BACKUP` command), but no incremental level 1 backup, so again, the command has no effect.

On the third run and all subsequent runs, there is a datafile copy and a level 1 incremental from the previous run, so the level 1 incremental is applied to the datafile copy, bringing the datafile copy up to the checkpoint SCN of the level 1 incremental.

Note also the following details about how this example works:

- Each time a datafile is added to the database, an image copy of the new datafile is created the next time the script runs. The time after that, the first level 1 incremental for that datafile is created, and on all subsequent runs the new datafile is processed like any other datafile.
- Tags must be used to identify the incremental level 0 datafile copies created for use in this strategy, so that they do not interfere with other backup strategies you implement. If you have multiple incremental backup strategies in effect, RMAN cannot unambiguously create incremental level 1 backups unless you tag level 0 backups.

The incremental level 1 backups to apply to those image copies are selected based upon the checkpoint SCNs of the image copy datafiles and the available

incremental level 1 backups. (The tag used on the image copy being recovered is not a factor in the selection of the incremental level backups.)

In practice, you would schedule the example script to run once each day, possibly at midnight. On a typical night (that is, after the first two nights), when the script completed the following files would be available for a point-in-time recovery:

- An image copy of the database, as of the checkpoint SCN of the preceding run of the script, 24 hours earlier
- An incremental backup for the changes since the checkpoint SCN of preceding run
- Archived redo logs including all changes between the checkpoint SCN of the image copy and the current time

If, at some point during the following 24 hours, you need to restore and recover your database from this backup, for either complete or point-in-time recovery, you can restore the datafiles from the incrementally updated datafile copies, and apply changes from the most recent incremental level 1 and the redo logs to reach the desired SCN. At most, you will have 24 hours of redo to apply, which limits how long point-in-time recovery will take.

See Also: *Oracle Database 2 Day DBA* to see how this technique is used in the Oracle-suggested backup strategy in Enterprise Manager.

Incrementally Updated Backups: A One Week Example

The basic example can be extended to provide fast recoverability to a window greater than 24 hours. Alter the `RECOVER COPY . . . WITH TAG` to perform incomplete recovery of the datafile copies to the point in time in the past where you want your window of recoverability to begin. This example shows how to maintain a seven day window:

```
RUN {
  RECOVER COPY OF DATABASE WITH TAG 'incr_update'
    UNTIL TIME 'SYSDATE - 7';
  BACKUP INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG 'incr_update'
    DATABASE;
}
```

The effect of the script is as follows:

- On the first night the `RECOVER COPY . . . UNTIL TIME` statement has no effect, and the `BACKUP INCREMENTAL . . . FOR RECOVER OF COPY` statement creates the incremental level 0 copy.
- On the second through seventh nights, the `RECOVER COPY . . . UNTIL TIME` statement has no effect because `TIME 'SYSDATE - 7'` is still a time in the future. The `BACKUP INCREMENTAL . . . FOR RECOVER OF COPY` statement creates differetial incremental level 1 backups containing the block changes for the previous day.
- On the eighth and all subsequent nights night, the `RECOVER COPY . . . UNTIL TIME` statement applies the level 1 incremental from seven days ago to the copy of the database. The `BACKUP INCREMENTAL . . . FOR RECOVER OF COPY` statement creates an incremental backup containing the changes for the previous day.

As with the basic example, you have fast recoverability to any point in time between the SCN of the datafile copies and the present, using block changes from the

incremental backups and individual changes from the redo logs. Because you have the daily level 1 incrementals, you still never need to apply more than one day of redo.

Improving Incremental Backup Performance: Change Tracking

RMAN's change tracking feature for incremental backups improves incremental backup performance by recording changed blocks in each datafile in a change tracking file. If change tracking is enabled, RMAN uses the change tracking file to identify changed blocks for incremental backup, thus avoiding the need to scan every block in the datafile.

After enabling change tracking, the first level 0 incremental backup still has to scan the entire datafile, as the change tracking file does not yet reflect the status of the blocks. Subsequent incremental backup that use this level 0 as parent will take advantage of the change tracking file.

Using change tracking in no way changes the commands used to perform incremental backups, and the change tracking files themselves generally require little maintenance after initial configuration.

Change tracking is disabled by default, because it does introduce some minimal performance overhead on your database during normal operations. However, the benefits of avoiding full datafile scans during backup are considerable, especially if only a small percentage of data blocks are changed between backups. If your backup strategy involves incremental backups, then you should enable change tracking.

One change tracking file is created for the whole database. By default, the change tracking file is created as an Oracle managed file in `DB_CREATE_FILE_DEST`. You can also specify the name of the block change tracking file, placing it in any location you choose.

Note: In a Real Applications Clusters (RAC) environment, the change tracking file must be located on shared storage accessible from all nodes in the cluster.

Oracle saves enough change-tracking information to enable incremental backups to be taken using any of the 8 most recent incremental backups as its parent.

Although RMAN does not support backup and recovery of the change-tracking file itself, if the whole database or a subset needs to be restored and recovered, then recovery has no user-visible effect on change tracking. After the restore and recovery, the change tracking file is cleared, and starts recording block changes again. The next incremental backup after any recovery is able to use change-tracking data.

Enabling and Disabling Change Tracking

You can enable or disable change tracking when the database is either open or mounted. To alter the change tracking setting, you must use SQL*Plus to connect to the target database with administrator privileges.

To store the change tracking file in the database area, set `DB_CREATE_FILE_DEST` in the target database. Then issue the following SQL statement to enable change tracking:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING;
```

You can also create the change tracking file in a location you choose yourself, using the following SQL statement:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING
```

```
USING FILE '/mydir/rman_change_track.f' REUSE;
```

The REUSE option tells Oracle to overwrite any existing file with the specified name.

To disable change tracking, use this SQL statement:

```
SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
```

If the change tracking file was stored in the database area, then it is deleted when you disable change tracking.

Checking Whether Change Tracking is Enabled

From SQL*Plus, you can query `V$BLOCK_CHANGE_TRACKING.STATUS` to determine whether change tracking is enabled, and if it is, query `V$BLOCK_CHANGE_TRACKING.FILENAME` to display the filename.

Moving the Change Tracking File

If you need to move the change tracking file, the `ALTER DATABASE RENAME FILE` command updates the control file to refer to the new location. The process outlined in this section describes how to change the location of the change tracking file while preserving its contents.

To relocate the change tracking file:

1. If necessary, determine the current name of the change tracking file:

```
SELECT filename
FROM V$BLOCK_CHANGE_TRACKING;
```

2. Shut down the database. For example:

```
SHUTDOWN IMMEDIATE
```

3. Using host operating system commands, move the change tracking file to its new location.

4. Mount the database and move the change tracking file to a location that has more space. For example:

```
ALTER DATABASE RENAME FILE
'ora_home/dbs/change_trk.f' TO '/new_disk/change_trk.f';
```

5. Open the database:

```
ALTER DATABASE OPEN;
```

If you cannot shut down the database, then you must disable change tracking and re-enable it at the new location, as in the following example:

```
ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
ALTER DATABASE ENABLE BLOCK CHANGE TRACKING USING FILE 'new_location';
```

If you choose this method, you will lose the contents of the change tracking file. Until the next time you complete a level 0 incremental backup, RMAN will have to scan the entire file.

Estimating Size of the Change Tracking File on Disk

The size of the change tracking file is proportional to the size of the database and the number of enabled threads of redo. The size is not related to the frequency of updates to the database. Typically, the space required for block change tracking is

approximately 1/30,000 the size of the data blocks to be tracked. Note, however, the following two factors that may cause the file to be larger than this estimate suggests:

- To avoid overhead of allocating space as your database grows, the change tracking file size starts at 10MB, and new space is allocated in 10MB increments. Thus, for any database up to approximately 300GB the file size is no smaller than 10MB, for up to approximately 600GB the file size is no smaller than 20MB, and so on.
- For each datafile, a minimum of 320K of space is allocated in the change tracking file, regardless of the size of the file. Thus, if you have a large number of relatively small datafiles, the change tracking file is larger than for databases with a smaller number of larger datafiles containing the same data.

Using RMAN to Validate Database Files

You can use the `VALIDATE` option of the `BACKUP` command to verify that database files exist and are in the correct locations, and have no physical or logical corruptions that would prevent RMAN from creating backups of them. When performing a `BACKUP... VALIDATE`, RMAN reads the files to be backed up in their entirety, as it would during a real backup. It does not, however, actually produce any backup sets or image copies.

If the backup validation discovers corrupt blocks, then RMAN updates the `V$DATABASE_BLOCK_CORRUPTION` view with rows describing the corruptions. You can repair corruptions using block media recovery, documented in *Oracle Database Backup and Recovery Advanced User's Guide*. After a corrupt block is repaired, the row identifying this block is deleted from the view.

For example, you can validate that all database files and archived logs can be backed up by running a command as follows:

```
BACKUP VALIDATE DATABASE ARCHIVELOG ALL;
```

The RMAN client displays the same output that it would if it were really backing up the files. If RMAN cannot validate the backup of one or more of the files, then it issues an error message. For example, RMAN may show output similar to the following:

```
RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03002: failure of backup command at 08/29/2002 14:33:47
ORA-19625: error identifying file /oracle/oradata/trgt/arch/archive1_6.dbf
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

You cannot use the `MAXCORRUPT` or `PROXY` parameters with the `VALIDATE` option.

See Also:

- *Oracle Database Backup and Recovery Reference* for `BACKUP` syntax
- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to repair corrupt blocks discovered by `BACKUP . . . VALIDATE`

Overview of Reporting on Backups and the RMAN Repository

You can obtain information from the RMAN repository in several different ways:

- The RMAN `LIST` and `REPORT` commands provide extensive information on available backups and how they can be used to restore and recover your database. `LIST` is described in "[Listing RMAN Backups, Archived Logs, and Database Incarnations](#)" on page 4-22 and `REPORT` is described in "[Reporting on Backups and Database Schema](#)" on page 4-27.
- When the database is open, a number of V\$ views provide direct access to RMAN repository records in the control file. If your database is registered in a recovery catalog, RC_ views that mostly correspond to the V\$ views provide direct access to the RMAN repository data stored in the recovery catalog. You can query views from either group in SQL*Plus. Some V\$ views such as `V$DATAFILE_HEADER`, `V$PROCESS`, and `V$SESSION`, contain information not found in the recovery catalog views. The V\$ views are documented in *Oracle Database Reference* and the RC_ views are documented in *Oracle Database Backup and Recovery Reference*.
- The `RESTORE` command supports a `PREVIEW` clause, which provides detailed information about how RMAN can restore and recover all or part of your database, given the set of backups available. `RESTORE... PREVIEW` is documented in "[Previewing Backups Used in Restore Operations: RESTORE PREVIEW](#)" on page 6-9.

Note: If backups have been manipulated outside of the control of RMAN (for example, if some disk-based backups have been deleted, or tape backups are temporarily unavailable or permanently lost) then see [Chapter 8, "Recovery Manager Maintenance Tasks"](#) for details on how to update the RMAN repository record of backups to reflect the actual set of available backups, using commands such as `CHANGE`, `CROSSCHECK`, and `DELETE`. Otherwise, the output of the commands and views listed here may be misleading, and RMAN may not be able to find the backups to restore and recover your database until you update the repository.

See Also:

- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to keep the RMAN repository up to date
- *Oracle Database Backup and Recovery Reference* for `LIST` syntax
- *Oracle Database Backup and Recovery Reference* for `REPORT` syntax
- *Oracle Database Backup and Recovery Reference* for `RESTORE PREVIEW` syntax

Listing RMAN Backups, Archived Logs, and Database Incarnations

The `LIST` command uses the information in the RMAN repository to provide lists of backups, archived logs, and database incarnations. You can use the output of `LIST` to identify specific backups you wish to use with other RMAN commands.

This section contains these topics:

- [About RMAN Reports Generated by the LIST Command](#)
- [Listing Backups](#)
- [Listing Backups by File](#)
- [Listing Backups in Summary Mode](#)

- [Listing Selected Backups](#)
- [Listing Database Incarnations](#)

About RMAN Reports Generated by the LIST Command

You can control how the output is displayed by using the `BY BACKUP` and `BY FILE` options of the `LIST` command and choosing between the `SUMMARY` and `VERBOSE` options.

The primary purpose of the `LIST` command is to determine which backups are available. For example, you can list:

- Backups and proxy copies of a database, tablespace, datafile, archived redo log, or control file
- Backups that have expired
- Backups restricted by time, path name, device type, tag, or recoverability
- Incarnations of a database

Note that the `V$BACKUP_FILES` also contains list information for backups.

Listing Backups

By default, RMAN lists backups by backup, which means that it serially lists each backup or proxy copy and then identifies the files included in the backup. You can also list backups by file.

By default, RMAN lists in verbose mode. You can also list backups in a summary mode if the verbose mode generates too much output.

Listing Backups by Backup

To list backups by backup, connect to the target database and recovery catalog (if you use one), and then execute the `LIST BACKUP` command. Specify the desired objects with the `listObjList` clause. For example, you can enter:

```
LIST BACKUP;           # lists backup sets, image copies, and proxy copies
LIST BACKUPSET;       # lists only backup sets and proxy copies
LIST COPY;            # lists only disk copies
```

Optionally, specify `EXPIRED` to identify backups not found during a crosscheck:

```
LIST EXPIRED BACKUP;
```

Examine the output (refer to *Oracle Database Backup and Recovery Reference* for an explanation of the various column headings in the `LIST` output). Sample output of `LIST BACKUP` follows:

```
List of CHANGE, CROSSCHECK, and DELETE Backup Sets
=====
```

BS Key	Size	Device Type	Elapsed Time	Completion Time
7	136M	DISK	00:00:20	04-NOV-03
BP Key: 7 Status: AVAILABLE Compressed: NO Tag: TAG20031104T200759				
Piece Name: /oracle/work/RDBMS/backupset/2003_11_04/ol_mf_annnn_TAG20031104T200759_ztjxx3k8_.bkp				

```
List of Archived Logs in backup set 7
```

Thrd	Seq	Low SCN	Low Time	Next SCN	Next Time
1	1	173832	21-OCT-03 17:47:50	174750	21-OCT-03

1	2	174750	21-OCT-03	174755	21-OCT-03
1	3	174755	21-OCT-03	174758	21-OCT-03
1	37	533321	01-NOV-03	575472	03-NOV-03
1	38	575472	03-NOV-03	617944	04-NOV-03
1	39	617944	04-NOV-03	631495	04-NOV-03

```

BS Key  Type LV Size      Device Type Elapsed Time Completion Time
-----
8      Full  2M      DISK        00:00:01    04-NOV-03
      BP Key: 8  Status: AVAILABLE Compressed: NO  Tag: TAG20031104T200829
      Piece Name: /ade/lashdown_rdbms/oracle/dbs/c-774627068-20031104-01
Controlfile Included: Ckp SCN: 631510      Ckp time: 04-NOV-03
SPFILE Included: Modification time: 21-OCT-03
    
```

Sample output of LIST COPY follows:

```

List of Archived Log Copies
Key      Thrd Seq      S Low Time Name
-----
37      1      37      A 01-NOV-03 /oracle/work/RDBMS/archivelog/2003_11_03/o1_mf_1_37_ztd4h15d_.arc
38      1      38      A 03-NOV-03 /oracle/work/RDBMS/archivelog/2003_11_04/o1_mf_1_38_zthvgl68_.arc
39      1      39      A 04-NOV-03 /oracle/work/RDBMS/archivelog/2003_11_04/o1_mf_1_39_ztjxwxwy_.arc
    
```

Listing Backups by File

Specify the desired objects with the *listObjList* or *recordSpec* clause (refer to *Oracle Database Backup and Recovery Reference*). If you do not specify an object, then RMAN displays copies of all database files and archived logs. By default, RMAN lists in verbose mode, which means that it provides extensive, multiline information.

To list backups by file, connect the RMAN client to the target database and recovery catalog (if you use one), and then execute LIST with the BY FILE option, specifying the desired objects to list and options. For example, you can enter:

```

LIST BACKUP BY FILE; # shows backup sets, proxy copies, and image copies
LIST COPY BY FILE;  # shows only disk copies
    
```

For another example, you could specify the EXPIRED option to identify backups not found during a crosscheck:

```
LIST EXPIRED BACKUP BY FILE;
```

Examine the output (refer to *Oracle Database Backup and Recovery Reference* for an explanation of the various column headings in the LIST output). Sample output follows:

```

List of Datafile Backups
=====
File Key      TY LV S Ckp SCN      Ckp Time  #Pieces #Copies Compressed Tag
-----
1      5      B F A 631092    04-NOV-03 1      1      YES      TAG20031104T195949
      2      B F A 175337    21-OCT-03 1      1      NO       TAG20031021T094513
2      5      B F A 631092    04-NOV-03 1      1      YES      TAG20031104T195949
      2      B F A 175337    21-OCT-03 1      1      NO       TAG20031021T094513
    
```

... some rows omitted

```

List of Archived Log Backups
=====
Thrd Seq      Low SCN      Low Time    BS Key  S #Pieces #Copies Compressed Tag
-----
    
```

```

1 1 173832 21-OCT-03 7 A 1 1 NO TAG20031104T200759
1 1 173832 21-OCT-03 7 A 1 1 NO TAG20031021T094505
1 2 174750 21-OCT-03 7 A 1 1 NO TAG20031104T200759
1 2 174750 21-OCT-03 7 A 1 1 NO TAG20031021T094505
... some rows omitted
1 38 575472 03-NOV-03 7 A 1 1 NO TAG20031104T200759
1 39 617944 04-NOV-03 7 A 1 1 NO TAG20031104T200759

```

List of Controlfile Backups

```

=====
CF Ckp SCN Ckp Time BS Key S #Pieces #Copies Compressed Tag
-----
631510 04-NOV-03 8 A 1 1 NO TAG20031104T200829
631205 04-NOV-03 6 A 1 1 NO TAG20031104T200432
175380 21-OCT-03 4 A 1 1 NO TAG20031021T094639

```

List of SPFILE Backups

```

=====
Modification Time BS Key S #Pieces #Copies Compressed Tag
-----
21-OCT-03 8 A 1 1 NO TAG20031104T200829
21-OCT-03 6 A 1 1 NO TAG20031104T200432

```

Listing Backups in Summary Mode

By default the `LIST` output is detailed, but you can also specify that RMAN display the output in summarized form. Specify the desired objects with the `listObjectList` or `recordSpec` clause. If you do not specify an object, then `LIST BACKUP` displays all backups.

After connecting to the target database and recovery catalog (if you use one), execute `LIST BACKUP`, specifying the desired objects and options. For example:

```
LIST BACKUP SUMMARY; # lists backup sets, proxy copies, and disk copies
```

You can also specify the `EXPIRED` keyword to identify those backups that were not found during a crosscheck:

```
LIST EXPIRED BACKUP SUMMARY;
```

Sample output follows:

```

List of Backups
=====
Key TY LV S Device Type Completion Time #Pieces #Copies Compressed Tag
-----
1 B A A SBT_TAPE 21-OCT-03 1 1 NO TAG20031021T094505
2 B F A SBT_TAPE 21-OCT-03 1 1 NO TAG20031021T094513
3 B A A SBT_TAPE 21-OCT-03 1 1 NO TAG20031021T094624
4 B F A SBT_TAPE 21-OCT-03 1 1 NO TAG20031021T094639
5 B F A DISK 04-NOV-03 1 1 YES TAG20031104T195949
6 B F A DISK 04-NOV-03 1 1 NO TAG20031104T200432
7 B A A DISK 04-NOV-03 1 1 NO TAG20031104T200759
8 B F A DISK 04-NOV-03 1 1 NO TAG20031104T200829

```

Refer to *Oracle Database Backup and Recovery Reference* for an explanation of the various column headings in the `LIST` output.

Listing Selected Backups

You can specify several different conditions to narrow your `LIST` output.

After connecting to the target database and recovery catalog (if you use one), execute LIST COPY or LIST BACKUP with the *listObjList* or *recordSpec* clause. For example, enter any of the following commands:

```
# lists backups of all files in database
LIST BACKUP OF DATABASE;
# lists copy of specified datafile
LIST COPY OF DATAFILE 'ora_home/oradata/trgt/system01.dbf';
# lists specified backup set
LIST BACKUPSET 213;
# lists datafile copy
LIST DATAFILECOPY '/tmp/tools01.dbf';
```

You can also restrict the search by specifying the *maintQualifier* or RECOVERABLE clause. For example, enter:

```
# specify a backup set by tag
LIST BACKUPSET TAG 'weekly_full_db_backup';
# specify a backup or copy by device type
LIST COPY OF DATAFILE 'ora_home/oradata/trgt/system01.dbf' DEVICE TYPE sbt;
# specify a backup by directory or path
LIST BACKUP LIKE '/tmp/%';
# specify a backup or copy by a range of completion dates
LIST COPY OF DATAFILE 2 COMPLETED BETWEEN '10-DEC-2002' AND '17-DEC-2002';
# specify logs backed up at least twice to tape
LIST ARCHIVELOG ALL BACKED UP 2 TIMES TO DEVICE TYPE sbt;
```

The output depends upon the options you pass to the LIST command. For example, the following lists copies of datafile 1:

```
RMAN> list backup of datafile 1;
```

```
List of Backup Sets
```

```
=====
```

BS Key	Type	LV	Size	Device Type	Elapsed Time	Completion Time
2	Full		230M	SBT_TAPE	00:00:49	21-OCT-03
	BP Key: 2 Status: AVAILABLE Compressed: NO Tag: TAG20031021T094513					
	Handle: 02f4eatc_1_1 Media: /smrdir					

```
List of Datafiles in backup set 2
```

File	LV	Type	Ckp SCN	Ckp Time	Name
1		Full	175337	21-OCT-03	/oracle/dbs/tbs_01.f

BS Key	Type	LV	Size	Device Type	Elapsed Time	Completion Time
5	Full		233M	DISK	00:04:30	04-NOV-03
	BP Key: 5 Status: AVAILABLE Compressed: NO Tag: TAG20031104T195949					
	Piece Name: /oracle/work/RDBMS/backupset/2003_11_04/o1_mf_nnndf_TAG20031104T195949_					

```
ztjxfvgz_.bkp
```

File	LV	Type	Ckp SCN	Ckp Time	Name
1		Full	631092	04-NOV-03	/ade/lashdown_rdbms/oracle/dbs/tbs_01.f

```
List of Datafiles in backup set 5
```

File	LV	Type	Ckp SCN	Ckp Time	Name
1		Full	631092	04-NOV-03	/ade/lashdown_rdbms/oracle/dbs/tbs_01.f

See Also:

- *Oracle Database Backup and Recovery Reference* for `listObjList` and `recordSpec` syntax
- *Oracle Database Backup and Recovery Reference* for an explanation of the various columns in the LIST output

Listing Database Incarnations

Each time an OPEN RESETLOGS operation is performed on a database, this operation creates a new incarnation of the database. Database incarnations and their effect upon database recovery with RMAN are explained in more detail in *Oracle Database Backup and Recovery Advanced User's Guide*.

When performing incremental backups, RMAN can use a backup from a previous incarnation or the current incarnation as a basis for subsequent incremental backups. When performing restore and recovery, RMAN can use backups from a previous incarnation in restore and recovery operations just as it would use backups from the current incarnation, as long as all archived logs are available.

Use the LIST INCARNATION command to see the incarnations of your database.

To list database incarnations:

After connecting to the target database and the recovery catalog if applicable, run LIST INCARNATION:

```
RMAN> LIST INCARNATION;
```

If you are using a recovery catalog, and if you register multiple target databases in the same catalog, then you can distinguish them by using the OF DATABASE option:

```
RMAN> LIST INCARNATION OF DATABASE prod3;
```

Refer to *Oracle Database Backup and Recovery Reference* for an explanation of the various column headings in the LIST output). Sample output follows:

```
RMAN> LIST INCARNATION OF DATABASE;
```

```
List of Database Incarnations
DB Key  Inc Key DB Name  DB ID          STATUS  Reset SCN  Reset Time
-----  -
1       1       RDBMS    774627068     PARENT  1          21-OCT-03
2       2       RDBMS    774627068     CURRENT 173832     21-OCT-03
```

The preceding output indicates that a RESETLOGS was performed on database `trgt` at SCN 164378, resulting in a new incarnation. The incarnation is distinguished by incarnation key (represented in the `Inc Key` column).

Reporting on Backups and Database Schema

The RMAN REPORT command analyzes the available backups and your database to answer important questions, such as:

- Which files need a backup?
- Which files have had unrecoverable operations performed on them?
- Which backups are obsolete and can be deleted?
- What was the physical schema of the database at some previous time?

- Which files have not been backed up recently?

Note: The results of `REPORT` are only correct if the RMAN repository has an accurate record of available backups and the state of your database. For example, if backups have been deleted from disk or tape outside of RMAN, reports generated by RMAN do not automatically reflect these changes.

If some backups recorded in the RMAN repository have been deleted, or if some are temporarily unavailable due to a storage device being offline or media being unavailable, you can use the `CROSSCHECK` commands to update the status of all backups, or the `CHANGE`, `CATALOG`, `UNCATALOG` and `DELETE` commands to directly set the status of individual backups. See [Chapter 8, "Recovery Manager Maintenance Tasks"](#) to learn how to update the RMAN repository to reflect your actual available backups.

This section contains the following topics:

- [About Reports of RMAN Backups](#)
- [Reporting on Files Needing a Backup Under a Retention Policy](#)
- [Reporting on Datafiles Affected by Unrecoverable Operations](#)
- [Reporting Obsolete Backups](#)
- [Reporting on the Database Schema](#)

About Reports of RMAN Backups

Reports enable you to confirm that your backup and recovery strategy is in fact meeting your requirements for database recoverability. The two major forms of `REPORT` used to determine whether your database is recoverable are:

- `REPORT NEED BACKUP`

Reports which database files need to be backed up to meet a configured or specified retention policy

- `REPORT UNRECOVERABLE`

Reports which database files require backup because they have been affected by some `NOLOGGING` operation such as a direct-path insert

Note: The `REPORT` command use the information in the RMAN repository as the basis for their output. If backups have been manipulated outside of the control of RMAN (for example, if you believe some disk-based backups have been deleted, or tape backups are temporarily unavailable or permanently lost) then see [Chapter 8, "Recovery Manager Maintenance Tasks"](#) for details on how to update the RMAN repository record to contain the actual set of available backups using the `CROSSCHECK`, `CHANGE`, `CATALOG`, `UNCATALOG` and `DELETE` commands. Otherwise, the output of `REPORT` may be misleading.

Reporting on Files Needing a Backup Under a Retention Policy

Use the `REPORT NEED BACKUP` command to determine which database files need backup under a specific retention policy.

With no arguments, `REPORT NEED BACKUP` reports which objects need backup under the currently configured retention policy. The output for a configured retention policy of `REDUNDANCY 1` is similar to this example:

```
REPORT NEED BACKUP;

RMAN retention policy will be applied to the command
RMAN retention policy is set to redundancy 1
Report of files with less than 1 redundant backups
File #bkps Name
-----
2      0      /oracle/oradata/trgt/undotbs01.dbf
```

Note: If you disable the retention policy using `CONFIGURE RETENTION POLICY TO NONE`, then `REPORT NEED BACKUP` returns an error message, because without a retention policy, RMAN cannot determine which files need to be backed up.

Using RMAN REPORT NEED BACKUP with Different Retention Policies

You can specify different criteria for `REPORT NEED BACKUP`, using one of the following forms of the command:

- `REPORT NEED BACKUP RECOVERY WINDOW OF n DAYS`
Displays objects requiring backup to satisfy a recovery window-based retention policy.
- `REPORT NEED BACKUP REDUNDANCY n`
Displays objects requiring backup to satisfy a redundancy-based retention policy.
- `REPORT NEED BACKUP DAYS = n`
Displays files that require more than *n* days' worth of archived redo log files for recovery.
- `REPORT NEED BACKUP INCREMENTAL n`
Displays files that require application of more than *n* incremental backups for recovery.

Using RMAN REPORT NEED BACKUP with Tablespaces and Datafiles

`REPORT NEED BACKUP` can check the entire database, skip specified tablespaces, or check only specific tablespaces or datafiles against different retention policies, as shown in the following examples:

```
RMAN> REPORT NEED BACKUP RECOVERY WINDOW OF 2 DAYS DATABASE SKIP TABLESPACE TBS_2;
RMAN> REPORT NEED BACKUP REDUNDANCY 2 DATAFILE 1;
RMAN> REPORT NEED BACKUP TABLESPACE TBS_3; # uses configured retention policy
RMAN> REPORT NEED BACKUP INCREMENTAL 2; # checks entire database
```

See Also: *Oracle Database Backup and Recovery Reference* for all possible options for `REPORT NEED BACKUP` and an explanation of the various column headings in the output

Using REPORT NEED BACKUP with Backups onTape or Disk Only

You can limit the backups tested by REPORT NEED BACKUP to disk-based or tape-based backups only, as shown in these examples:

```

RMAN> REPORT NEED BACKUP RECOVERY WINDOW OF 2 DAYS DATABASE DEVICE TYPE SBT;
RMAN> REPORT NEED BACKUP DEVICE TYPE DISK;
RMAN> REPORT NEED BACKUP TABLESPACE TBS_3 DEVICE TYPE SBT;

```

Reporting on Datafiles Affected by Unrecoverable Operations

When a datafile has been changed by an unrecoverable operation, such as a direct load insert, normal media recovery cannot be used to recover the file, because an unrecoverable operation does not generate redo. You must perform either a full or incremental backup of affected datafiles after such operations, to ensure that data blocks affected by the unrecoverable operation can be recovered using RMAN.

To identify datafiles affected by an unrecoverable operation and the type of backup required to ensure the datafile can be restored from backup, use the REPORT UNRECOVERABLE command, as shown in this example:

```

RMAN> REPORT UNRECOVERABLE;
Report of files that need backup due to unrecoverable operations
File Type of Backup Required Name
-----
1    full                               /oracle/oradata/trgt/system01.dbf

```

Reporting Obsolete Backups

You can report backup sets, backup pieces and datafile copies that are obsolete, that is, not needed to meet a specified retention policy, by specifying the OBSOLETE keyword. If you do not specify any other options, then REPORT OBSOLETE displays the backups that are obsolete according to the current retention policy, as shown in the following example:

```

RMAN> REPORT OBSOLETE;
Datafile Copy      44      08-FEB-05      /backup/ora_df549738566_s70_s1
Datafile Copy      45      08-FEB-05      /backup/ora_df549738567_s71_s1
Datafile Copy      46      08-FEB-05      /backup/ora_df549738568_s72_s1
Backup Set         26      08-FEB-05
  Backup Piece      26      08-FEB-05      /backup/ora_df549738682_s76_s1
.
.
.

```

You can also check which backups are obsolete under different recovery window-based or redundancy-based retention policies, by using REPORT OBSOLETE with RECOVERY WINDOW and REDUNDANCY options, as shown in these examples:

```

REPORT OBSOLETE RECOVERY WINDOW OF 3 DAYS;
REPORT OBSOLETE REDUNDANCY 1;

```

You can specify to only consider backups on disk or on tape when checking for obsolete files, by specifying a device type with the REPORT command, as in this example:

```

REPORT OBSOLETE RECOVERY WINDOW OF 3 DAYS DEVICE TYPE DISK;
REPORT OBSOLETE REDUNDANCY 1;

```

To report obsolete backups:

1. Connect to your target database and your recovery catalog (if you are using a recovery catalog.)
2. To make sure that your RMAN repository has current information about the status of different backups, you may want to issue `CROSSCHECK` commands to update the status of backups in the repository compared to their status on disk, or use the `CHANGE`, `CATALOG`, `UNCATALOG` and `DELETE` commands to directly specify the status of individual backups.

In the simplest case, you could crosscheck all backups on disk, tape or both, using any one of the following commands:

```
RMAN> CROSSCHECK BACKUP DEVICE TYPE DISK;
RMAN> CROSSCHECK BACKUP DEVICE TYPE SBT;
RMAN> CROSSCHECK BACKUP; # crosschecks all backups on all devices
```

See [Chapter 8, "Recovery Manager Maintenance Tasks"](#) for more details on how to update the RMAN repository record to contain the actual set of available backups.

3. Run `REPORT OBSOLETE` to identify which backups are obsolete because they are no longer needed for recovery. Here are several examples:

```
# lists backups that not needed to recover the database to within last week
REPORT OBSOLETE RECOVERY WINDOW OF 7 DAYS;
```

```
# lists backups not needed for redundancy-based retention policy, considering
only backups stored on tape
REPORT OBSOLETE REDUNDANCY = 2 DEVICE TYPE sbt;
```

See Also:

- ["Configuring the Backup Retention Policy"](#) on page 3-18 for a conceptual overview of RMAN backup retention policy
- ["Deleting Expired RMAN Backups after CROSSCHECK"](#) on page 8-7 for information on deleting RMAN backups and deleting records of RMAN backups from the RMAN repository

Reporting on the Database Schema

The `REPORT SCHEMA` command lists and displays information about the database files.

After connecting RMAN to the target database and recovery catalog (if you use one), issue `REPORT SCHEMA` as shown in this example:

```
RMAN> REPORT SCHEMA;
List of Permanent Datafiles
=====
File Size(MB) Tablespace          RB segs Datafile Name
-----
1    450    SYSTEM                ***    /oracle/oradata/tbs_01.f
2    50     SYSAUX                ***    /oracle/oradata/tbs_ax1.f
3    2      SYSTEM                ***    /oracle/oradata/tbs_02.f
4    2      TBS_1                 ***    /oracle/oradata/tbs_11.f
.
.
.
21   2      TBS_4                 ***    /oracle/oradata/tbs_43.f
22   2      TBS_5                 ***    /oracle/oradata/tbs_53.f
```

List of Temporary Files

```
=====
File Size(MB) Tablespace           Maxsize(MB) Tempfile Name
-----
1      40      TEMP                32767      /oracle/oradata/tbs_tmp1.f
```

Note: If you use a recovery catalog, then you can use the *atClause* to specify a past time, SCN, or log sequence number, as shown in these examples of the command:

```
REPORT SCHEMA AT TIME 'SYSDATE-14';      # schema 14 days ago
REPORT SCHEMA AT SCN 1000;               # schema at scn 1000
REPORT SCHEMA AT SEQUENCE 100 THREAD 1; # schema at sequence 100
```

Data Protection with Restore Points and Flashback Database

This chapter explains the concepts of Flashback Database and restore points, and explains setup, ongoing monitoring and maintenance associated with these features when incorporated into your data protection strategy.

This chapter contains the following sections:

- [Restore Points and Flashback Database: Concepts](#)
- [Using Normal and Guaranteed Restore Points](#)
- [Setup and Maintenance for Oracle Flashback Database](#)

Note: Detailed information on recovery scenarios that use Flashback Database and normal and guaranteed restore points can be found in [Chapter 7, "Performing Flashback and Database Point-in-Time Recovery"](#).

Restore Points and Flashback Database: Concepts

Flashback Database and Restore Points are two related data protection features of Oracle that provide more efficient alternatives to point-in-time recovery for reversing unwanted database changes. This section provides a conceptual overview of Flashback Database first, and then of Restore Points.

Flashback Database enables you to wind your entire database backward in time, reversing the effects of unwanted database changes within a given time window. The effects are similar to database point-in-time recovery.

Restore points provide capabilities related to Flashback Database as well as other recovery operations. Guaranteed restore points, in particular, provide a complementary capability to Flashback Database, allowing you to select an SCN and enforce the requirement that Flashback Database be usable to that SCN, though not necessarily to SCNs between the guaranteed restore point and the present SCN.

Restore points and Flashback Database can be used independently of each other or together. In both cases, the `RMAN FLASHBACK DATABASE` command or the `SQL*Plus FLASHBACK DATABASE` statement is used to actually return the database to a specified SCN, as in the following examples:

```
FLASHBACK DATABASE TO RESTORE POINT 'before_upgrade';  
FLASHBACK DATABASE TO SCN 202381;
```

It is easier, however, to explain the functionality of guaranteed restore points, the interactions of the two features and the tradeoffs in choosing to use either or both of them, after an overview of the more general Flashback Database functionality.

This section contains the following topics:

- [About Flashback Database](#)
- [About Normal Restore Points](#)
- [About Guaranteed Restore Points](#)
- [About Logging for Flashback Database and Guaranteed Restore Points](#)

About Flashback Database

Oracle Flashback Database, accessible from both RMAN (by means of the `FLASHBACK DATABASE` command) and SQL*Plus (by means of the `FLASHBACK DATABASE` statement), lets you quickly recover the entire database from logical data corruptions or user errors.

It is similar to conventional point in time recovery in its effects, allowing you to return a database to its state at a time in the recent past. Flashback Database is, however, much faster than point-in-time recovery, because it does not require restoring datafiles from backup and it requires applying fewer changes from the archived redo logs.

Flashback Database can be used to reverse most unwanted changes to a database, as long as the datafiles are intact. This includes returning a database to its state in previous incarnations, that is, undoing the effects of an `OPEN RESETLOGS` operation.

Note: Detailed information on the use of the `FLASHBACK DATABASE` command in reversing unwanted database changes can be found in "[Reversing Database Changes with Flashback Database](#)" on page 7-13.

Flashback Database uses its own logging mechanism, creating **flashback logs** which are stored in the flash recovery area. You can only use Flashback Database if flashback logs are available. Therefore, you must set up your database in advance to create flashback logs if you want to take advantage of this feature.

To enable Flashback Database, you set up a flash recovery area, and set a **flashback retention target**, to specify how far back into the past you want to be able to restore your database with Flashback Database.

From that time on, at regular intervals, the database copies images of each altered block in every datafile into the flashback logs. These block images can later be reused to reconstruct the datafile contents as of any moment at which logs were captured.

When a database is restored to its state at some past target time using Flashback Database, each block changed since that time is restored from the copy of the block in the flashback logs most immediately prior to the desired target time. The redo log is then used to re-apply changes since the time that block was copied to the flashback logs.

Note: Redo logs must be available for the entire time period spanned by the flashback logs, whether on tape or on disk. (In practice, however, redo logs are generally needed much longer than the flashback retention target to support point-in-time recovery.)

About the Flashback Database Window

The range of SCNs for which there is currently enough flashback log data to support the `FLASHBACK DATABASE` command is called the **flashback database window**. If space in the flash recovery area is low, then flashback logs may be deleted to free space for files required by the configured retention policy. The result is that the flashback database window can be shorter than the flashback retention target, depending upon the size of the flash recovery area, other backups that must be retained and how much flashback logging data is needed.

Note: The flashback retention target is a target, not an absolute guarantee that Flashback Database will be available.

If your flash recovery area is not large enough to hold both the flashback logs and files that must be retained to meet the retention policy, such as archived redo logs and other backups, then the flashback logs from the earliest SCNs may be deleted to make room in the flash recovery area for other files.

The flashback database window cannot extend further back than the earliest SCN in the available flashback logs. Flashback logs cannot be backed up outside the flash recovery area, so to increase the likelihood that enough logs are retained to meet the flashback database window, maximize the free space available in your flash recovery area. See ["Sizing the Flash Recovery Area to Include Flashback Logs"](#) on page 5-10 for details.

There are also a number of operations you can perform on your database, such as dropping a tablespace or shrinking a datafile, which cannot be reversed with Flashback Database. After such an operation, the flashback database window begins at the time immediately following that operation.

If `FLASHBACK DATABASE` fails because the flashback database window is not long enough, database point-in-time recovery (DBPITR) can be used in most cases to achieve a similar result. See ["Performing Database Point-In-Time Recovery"](#) on page 7-19 for details on DBPITR.

Using guaranteed restore points is the only way to ensure that you can use Flashback Database to return to a specific point in time or guarantee the size of the flashback window. See ["About Guaranteed Restore Points"](#) on page 5-4 for more information on guaranteed restore points and Flashback Database.

About Normal Restore Points

Creating a normal restore point assigns the restore point name to a specific point in time or SCN, as a kind of bookmark or alias you can use with commands that recognize a `RESTORE POINT` clause as a shorthand for specifying an SCN.

Before performing any operation that you may have to reverse, you can create a normal restore point. The name of the restore point and the SCN are recorded in the control file. Then, if you later need to use Flashback Database, Flashback Table, or point-in-time recovery, you can refer to the target time using the name of the restore point instead of a time expression or SCN. Defining a normal restore point before an operation to be reversed later eliminates the need to manually record an SCN in

advance, or investigate the correct SCN after the fact using features such as Flashback Query.

Normal restore points are very lightweight. The control file can maintain a record of thousands of normal restore points with no significant impact upon database performance. Normal restore points eventually age out of the control file if not manually deleted, so they require no ongoing maintenance.

Commands Supporting the Use of Restore Points

Restore points can be used to specify the target SCN in the following contexts:

- The `RECOVER DATABASE` and `FLASHBACK DATABASE` commands in RMAN
- The `FLASHBACK TABLE` statement in SQL*Plus

Note: In general, a guaranteed restore point can be used as an alias for an SCN with any command that works with a normal restore point. Except as noted, the information about where and how to use normal restore points applies to guaranteed restore points as well.

About Guaranteed Restore Points

Like normal restore points, guaranteed restore points can be used as aliases for SCNs in recovery operations. However, they also provide specific functionality related to the use of the Flashback Database feature.

Creating a guaranteed restore point at a particular SCN enforces the requirement that you can perform a Flashback Database operation to return your database to its state at that SCN, even if flashback logging is not enabled for your database. If flashback logging is enabled, creating a guaranteed restore point enforces the retention of flashback logs required for Flashback Database back to any point in time after the creation of the earliest guaranteed restore point.

A guaranteed restore point can be used to revert a whole database to a known good state days or weeks ago, as long as there is enough disk space in flash recovery area to store the needed logs. As with Flashback Database, even the effects of `NOLOGGING` operations like direct load inserts can be reversed using guaranteed restore points.

Note: Limitations that apply to Flashback Database also apply to guaranteed restore points. For example, shrinking a datafile or dropping a tablespace can prevent flashing back the affected datafiles to the guaranteed restore point.

Using Guaranteed Restore Points Instead of Storage Snapshots

In practice, guaranteed restore points provide a useful alternative to storage snapshots, which are often used to protect a database before risky operations like large-scale database updates, or application patching or upgrading. Rather than creating a snapshot or duplicate database against which to test the operation, you can create a guaranteed restore point on a primary or standby database, and then perform the risky operation, with the certainty that the required flashback logs are retained.

About Logging for Flashback Database and Guaranteed Restore Points

The logging for Flashback Database and guaranteed restore points is based upon capturing images of datafile blocks before changes are applied, so that these images

can be used to return the datafiles to their previous state when a `FLASHBACK DATABASE` command is executed.

The chief differences between normal flashback logging and logging for guaranteed restore points are related to when blocks are logged and whether the logs can be deleted in response to space pressure in the flash recovery area. These differences affect space usage for logs and database performance.

Whether to enable logging for flashback database, use guaranteed restore points, or both depends upon your recoverability goals, and the implications in performance and in space usage for these features, separately and when used together.

Guaranteed Restore Points and Flash Recovery Area Space Usage

When you create a guaranteed restore point, with or without enabling full flashback database logging, you must monitor the space available in your flash recovery area. No file in the flash recovery area is eligible for deletion if it is required to satisfy the guarantee. Thus, retention of flashback logs and other files required to satisfy the guarantee, as well as files required to satisfy your backup retention policy, can cause the flash recovery area to fill completely.

Caution: If no files are eligible for deletion from the flash recovery area because of the requirements imposed by your retention policy and the guaranteed restore point, then the database behaves as if it has encountered a disk full condition. In many circumstances, this causes your database to halt. See ["When Space is Not Available in the Flash Recovery Area"](#) on page 3-21 for more information on the effects of a full flash recovery area.

See also: See ["Managing Space For Flashback Logs in the Flash Recovery Area"](#) on page 5-11 for instructions on how to monitor flash recovery area disk space usage.

Logging for Guaranteed Restore Points With Flashback Logging Disabled

If a guaranteed restore point is created when logging for Flashback Database is disabled, then, the first time a datafile block is modified after the time of the guaranteed restore point, an image of the block before the modification is stored in the flashback logs. The flashback logs thus preserve the contents of every changed data block at the time that the guaranteed restore point was created. However, subsequent modifications to the same block do not cause the block contents to be logged again, unless another guaranteed restore point has been created since the block was last modified.

This method of logging has the following important consequences:

- The available block images can be used to re-create the datafile contents at the time of a guaranteed restore point using `FLASHBACK DATABASE`, but you cannot use `FLASHBACK DATABASE` to reach points in time between the guaranteed restore points and the current time, as is possible when logging for Flashback Database is enabled. If you need to return the database to an intermediate point in time, your only option is database point-in-time recovery.
- Because each block that changes is only logged once, disk space usage for logging for guaranteed restore points when flashback logging is disabled is generally considerably less than normal flashback logging. You could maintain a guaranteed restore point for days or even weeks without concern over the ongoing growth of

flashback logs that occurs if logging for Flashback Database is enabled. The performance overhead of logging for a guaranteed restore point without flashback database logging is generally lower as well.

If your primary need is to be able to return your database to the specific point in time at which the guaranteed restore point was created, then it is generally more efficient to turn off logging for Flashback Database and use only guaranteed restore points. For example, in a scenario in which you are performing an application upgrade on a production database server over a weekend, you could create a guaranteed restore point at the start of the upgrade process and, if the process failed at the end, reverse the changes using `FLASHBACK DATABASE` instead of restoring the database from backups.

Logging for Flashback Database With Guaranteed Restore Points Defined

If Flashback Database is enabled and one or more guaranteed restore points is defined, then the database performs normal flashback logging, which causes some performance overhead and, depending upon the pattern of activity on your database, can cause significant space pressure in the flash recovery area. However, unlike normal logging for Flashback Database, the flash recovery area always retains the flashback logs required to allow `FLASHBACK DATABASE` to any time as far back as the earliest currently defined guaranteed restore point. Flashback logs are not deleted in response to space pressure, if they are required to satisfy the guarantee.

In this scenario you have the full flexibility of `FLASHBACK DATABASE` to any point in the flashback window, and the certainty of the guaranteed restore point, but you must monitor space used in the flash recovery area to support the guarantee, as described in ["Monitoring Space Usage For Guaranteed Restore Points"](#) on page 5-8 and ["Managing Space For Flashback Logs in the Flash Recovery Area"](#) on page 5-11.

Using Normal and Guaranteed Restore Points

This section describes the commands used to manage normal and guaranteed restore points. This section contains the following topics:

- [Creating Normal and Guaranteed Restore Points](#)
- [Listing Restore Points](#)
- [Dropping Restore Points](#)

Requirements for Using Guaranteed Restore Points

To support the use of guaranteed restore points, the database must satisfy the following requirements:

- The `COMPATIBLE` initialization parameter must be set to 10.2 or greater.
- The database must be running in `ARCHIVELOG` mode. The `FLASHBACK DATABASE` operation used to return your database to a guaranteed restore point requires the use of archived redo logs from around the time of the restore point.
- A flash recovery area must be configured, as described in ["Setting Up a Flash Recovery Area for RMAN"](#) on page 3-13. Guaranteed restore points use a mechanism similar to flashback logging, and as with flashback logging, Oracle must store the required logs in the flash recovery area.
- If flashback database is not enabled, then the database must be mounted, not open, when creating the first guaranteed restore point (or if all previously created guaranteed restore points have been dropped).

Note: There are no special requirements for using normal restore points.

Creating Normal and Guaranteed Restore Points

To create normal or guaranteed restore points, use the `CREATE RESTORE POINT` statement in SQL*Plus, providing a name for the restore point and specifying whether it is to be a guaranteed restore point or a normal one (the default).

The database can be open or mounted when creating restore points. If it is mounted, then it must have been shut down cleanly (unless this is a physical standby database).

This example shows how to create a normal restore point:

```
SQL> CREATE RESTORE POINT before_upgrade;
Restore point created.
```

This example shows how to create a guaranteed restore point:

```
SQL> CREATE RESTORE POINT before_upgrade GUARANTEE FLASHBACK DATABASE;
Restore point created.
```

See also: *Oracle Database SQL Reference* for reference information about the `SQL CREATE RESTORE POINT` statement

Listing Restore Points

To see a list of the currently defined restore points, use the `V$RESTORE_POINT` control file view, by means of the following query:

```
SQL> SELECT NAME, SCN, TIME, DATABASE_INCARNATION#,
           GUARANTEE_FLASHBACK_DATABASE, STORAGE_SIZE
FROM V$RESTORE_POINT;
```

You can view the name of each restore point, the SCN, wall-clock time and database incarnation number at which the restore points were created, whether each restore point is a guaranteed restore point, and how much space in the flash recovery area is being used to support information needed for Flashback Database operations to that restore point.

You can also use the following query to view only the guaranteed restore points:

```
SQL> SELECT NAME, SCN, TIME, DATABASE_INCARNATION#,
           GUARANTEE_FLASHBACK_DATABASE, STORAGE_SIZE
FROM V$RESTORE_POINT
WHERE GUARANTEE_FLASHBACK_DATABASE='YES';
```

For normal restore points, `STORAGE_SIZE` is zero. For guaranteed restore points, `STORAGE_SIZE` indicates the amount of disk space in the flash recovery area used to retain logs required to guarantee `FLASHBACK DATABASE` to that restore point.

Full information about the columns of `V$RESTORE_POINT` can be found in the *Oracle Database Reference*.

Dropping Restore Points

When you are satisfied that you do not need an existing restore point, or when you want to create a new restore point with the name of an existing restore point, you can

drop the restore point, using the `DROP RESTORE POINT SQL*Plus` statement. For example:

```
SQL> DROP RESTORE POINT before_app_upgrade;
Restore point dropped.
```

The same statement is used to drop both normal and guaranteed restore points.

Note: Normal restore points eventually age out of the control file, even if not explicitly dropped. The rules governing retention of restore points in the control file are:

- The most recent 2048 restore points are always kept in the control file, regardless of their age.
- Any restore point more recent than the value of `CONTROL_FILE_RECORD_KEEP_TIME` is retained, regardless of how many restore points are defined.

Normal restore points that do not meet either of these conditions may age out of the control file.

Guaranteed restore points never age out of the control file. They remain until they are explicitly dropped.

See also: *Oracle Database SQL Reference* for reference information about the `SQL DROP RESTORE POINT` statement

Monitoring Space Usage For Guaranteed Restore Points

When guaranteed restore points are defined on your database, you should monitor the amount of space used in your flash recovery area for files required to meet the guarantee. Use the query for viewing guaranteed restore points in "[Listing Restore Points](#)" on page 5-7 and refer to the `STORAGE_SIZE` column to determine the space required for files related to each guaranteed restore point. To see the total usage of your flash recovery area, use the query provided in "[Using V\\$RECOVERY_FILE_DEST and V\\$FLASH_RECOVERY_AREA_USAGE](#)" on page 3-18.

Setup and Maintenance for Oracle Flashback Database

This section describes planning for Oracle Flashback Database, configuring your database to use it and useful maintenance, monitoring and tuning procedures.

This discussion contains the following topics:

- [Limitations of Flashback Database](#)
- [Requirements for Enabling Flashback Database](#)
- [Enabling Logging for Flashback Database](#)
- [Sizing the Flash Recovery Area to Include Flashback Logs](#)
- [Managing Space For Flashback Logs in the Flash Recovery Area](#)
- [Determining the Current Window for Flashback Database](#)
- [Performance Tuning for Flashback Database](#)
- [Monitoring Flashback Database Performance Impact](#)

Limitations of Flashback Database

Because Flashback Database works by undoing changes to the datafiles that exist at the moment that you run the command, it has the following limitations:

- Flashback Database can only undo changes to a datafile made by an Oracle database. It cannot be used to repair media failures, or to recover from accidental deletion of datafiles.
- You cannot use Flashback Database to undo a shrink datafile operation.
- If the database control file is restored from backup or re-created, all accumulated flashback log information is discarded. You cannot use `FLASHBACK DATABASE` to return to a point in time before the restore or re-creation of a control file.
- When using Flashback Database with a target time at which a `NOLOGGING` operation was in progress, block corruption is likely in the database objects and datafiles affected by the `NOLOGGING` operation. For example, if you perform a direct-path `INSERT` operation in `NOLOGGING` mode, and that operation runs from 9:00 to 9:15 on April 3, 2005, and you later need to use Flashback Database to return to the target time 09:07 on that date, the objects and datafiles updated by the direct-path `INSERT` may be left with block corruption after the Flashback Database operation completes.

If possible, avoid using Flashback Database with a target time or SCN that coincides with a `NOLOGGING` operation. Also, perform a full or incremental backup of the affected datafiles immediately after any `NOLOGGING` operation to ensure recoverability to points in time after the operation. If you expect to use Flashback Database to return to a point in time during an operation such as a direct-path `INSERT`, consider performing the operation in `LOGGING` mode.

See the discussion of `logging_clause` in *Oracle Database SQL Reference* for more information about operations that support `NOLOGGING` mode.

Requirements for Enabling Flashback Database

The requirements for enabling Flashback Database are:

- Your database must be running in `ARCHIVELOG` mode, because archived logs are used in the Flashback Database operation.
- You must have a flash recovery area enabled, because flashback logs can only be stored in the flash recovery area.
- For Real Application Clusters databases, the flash recovery area must be stored in a clustered file system or in ASM.

Enabling Logging for Flashback Database

To enable logging for Flashback Database, set the `DB_FLASHBACK_RETENTION_TARGET` initialization parameter and issue the `ALTER DATABASE FLASHBACK ON` statement. Follow the process outlined here:

1. Start SQL*Plus and ensure that the database is mounted, but not open. For example:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

2. Optionally, set the `DB_FLASHBACK_RETENTION_TARGET` to the length of the desired flashback window in minutes:

```
SQL> ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET=4320; # 3 days
```

By default `DB_FLASHBACK_RETENTION_TARGET` is set to one day (1440 minutes).

3. Enable the Flashback Database feature for the whole database:

```
SQL> ALTER DATABASE FLASHBACK ON;
```

By default, flashback logs are generated for all permanent tablespaces. If you wish, you can reduce overhead by disabling flashback logging for specific tablespaces:

```
SQL> ALTER TABLESPACE tbs_3 FLASHBACK OFF;
```

You can re-enable flashback logging for a tablespace later with this command:

```
SQL> ALTER TABLESPACE tbs_3 FLASHBACK ON;
```

Note that if you disable Flashback Database for a tablespace, then you must take its datafiles offline before running `FLASHBACK DATABASE`.

You can disable flashback logging for the entire database with this command:

```
SQL> ALTER DATABASE FLASHBACK OFF;
```

You can also enable flashback logging on a standby database. Enabling Flashback Database on a standby database enables you to perform Flashback Database on the standby database. Flashback Database of standby databases has a number of applications in the Data Guard environment. See *Oracle Data Guard Concepts and Administration* for details.

Sizing the Flash Recovery Area to Include Flashback Logs

When using Flashback Database, you must add extra space to the flash recovery area to hold the flashback logs, along with the other files stored in the flash recovery area.

The setting of the `DB_FLASHBACK_RETENTION_TARGET` initialization parameter determines, indirectly, how much flashback log data the database retains. The size of flashback logs generated by the database for a given time period can vary considerably, however, depending on the specific database workload. If more blocks are affected by database updates during a given interval, then more disk space is used by the flashback log data generated for that interval.

Estimating Disk Space Requirements for Flashback Database Logs

The `V$FLASHBACK_DATABASE_LOG` view can help you estimate how much space to add to your flash recovery area for flashback logs. After you have enabled logging for Flashback Database and set a flashback retention target, allow the database to run under a normal workload for a while, to generate a representative sample of flashback logs. Then run the following query:

```
SQL>
SELECT ESTIMATED_FLASHBACK_SIZE FROM V$FLASHBACK_DATABASE_LOG;
```

The result is an estimate of the disk space needed to meet the current flashback retention target, based on the database workload since Flashback Database was enabled. Add the amount of disk space specified in `$FLASHBACK_DATABASE_LOG.ESTIMATED_FLASHBACK_SIZE` to your flash recovery area size, to hold the expected database flashback logs.

Space usage in the flash recovery area is always balanced among backups and archived logs which must be kept according to the retention policy, and other files

which may be subject to deletion, such as flashback logs and backups already backed up to tape but still cached on disk. If you have not allocated enough space in your flash recovery area to store your flashback logs and still meet your other backup retention requirements, flashback logs may be deleted from the recovery area to make room.

Managing Space For Flashback Logs in the Flash Recovery Area

You cannot manage the flashback logs in the flash recovery area directly, other than by setting the flashback retention target or using guaranteed restore points. You can, however, manage flash recovery area space as a whole, in order to maximize space available for retention of flashback database logs. In this way you increase the likelihood of achieving the flashback retention target.

To make space for flashback logs, back up the other contents of your flash recovery area to tape, using commands such as `BACKUP RECOVERY AREA`, `BACKUP BACKUPSET` and so on. Then use the `DELETE` command to explicitly remove files from the flash recovery area once they have been backed up on tape. If this still does not create enough space to satisfy your backup retention policy and flashback retention target, allocate more space for your flash recovery area.

Note: `BACKUP RECOVERY AREA` does not include the flashback logs when backing up flash recovery area contents to tape.

Rules for Retention and Deletion of Flashback Logs

The following rules govern the flash recovery area's creation, retention, overwriting and deletion of flashback logs:

- A flashback log is created whenever necessary to satisfy the flashback retention target, as long as there is enough space in the flash recovery area.
- A flashback log can be reused, once it is old enough that it is no longer needed to satisfy the flashback retention target.
- If the database needs to create a new flashback log and the flash recovery area is full or there is no disk space, then the oldest flashback log is reused instead.

Note: Re-using the oldest flashback log shortens the flashback database window. If enough flashback logs are reused due to a lack of disk space, the flashback retention target may not be satisfied.

- If the flash recovery area is full, then an archived redo log may be automatically deleted by the flash recovery area to make space for other files. In such a case, any flashback logs that would require the use of that redo log file for the use of `FLASHBACK DATABASE` are also deleted.

Determining the Current Window for Flashback Database

When flashback logging is enabled, the earliest SCN in the flashback database window can be determined by querying `V$FLASHBACK_DATABASE_LOG.OLDEST_FLASHBACK_SCN` and `V$FLASHBACK_DATABASE_LOG.OLDEST_FLASHBACK_TIME` as shown in this example:

```
SELECT OLDEST_FLASHBACK_SCN, OLDEST_FLASHBACK_TIME
FROM V$FLASHBACK_DATABASE_LOG;
```

If the results of this query regularly indicate that the flashback database window does not satisfy the flashback retention target, you may need to increase your flash recovery area to accommodate more flashback logs.

Caution: This view indicates how much flashback log data is available for your database. However, there are operations, such as shrinking or dropping a datafile, which can prevent Flashback Database from succeeding against your entire database, even if flashback log data is available. The results of this query do not reflect such limitations.

The most recent SCN that can be reached with Flashback Database is the current SCN of the database. This query returns the current SCN:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE;
```

Performance Tuning for Flashback Database

Maintaining flashback logs imposes comparatively limited overhead on an Oracle database instance. Changed blocks are written from memory to the flashback logs at relatively infrequent, regular intervals, to limit processing and I/O overhead.

To achieve good performance for large production databases with Flashback Database enabled, Oracle recommends the following:

- Use a fast file system for your flash recovery area, preferably without operating system file caching. Files the database creates in the flash recovery area, including flashback logs, are typically large. Operating system file caching is typically not effective for these files, and may actually add CPU overhead for reading from and writing to these files. Thus, it is recommended to use a file system that avoids operating system file caching, such as ASM.
- Configure enough disk spindles for the file system that will hold the flash recovery area. For large production databases, multiple disk spindles may be needed to support the required disk throughput for the database to write the flashback logs effectively.
- If the storage system used to hold the flash recovery area does not have non-volatile RAM, try to configure the file system on top of striped storage volumes, with a relatively small stripe size such as 128K. This will allow each write to the flashback logs to be spread across multiple spindles, improving performance.
- For large, production databases, set the `init.ora` parameter `LOG_BUFFER` to be at least 8MB. This makes sure the database allocates maximum memory (typically 16MB) for writing flashback database logs.

The overhead of turning on logging for Flashback Database depends on the mixture of reads and writes in the database workload. The more write-intensive the workload, the higher the overhead caused by turning on logging for Flashback Database. (Queries do not change data and thus do not contribute to logging activity for Flashback Database.)

Monitoring Flashback Database Performance Impact

The best way to monitor system usage due to flashback logging is to take performance statistics using the Oracle Statspack. For example, if you see "flashback buf free by RVWR" as the top wait event, it indicates that Oracle cannot write flashback logs very quickly. In such a case, you may want to tune the file system and storage used by the flash recovery area, possibly using one of the methods described in "[Performance Tuning for Flashback Database](#)" on page 5-12.

The V\$FLASHBACK_DATABASE_STAT view (described in *Oracle Database Reference*) shows the bytes of flashback data logged by the database. Each row in the view shows the statistics accumulated (typically over the course of an hour). The FLASHBACK_DATA and REDO_DATA columns describe bytes of flashback data and redo data written respectively during the time interval, while the DB_DATA column describe bytes of data blocks read and written. Note that FLASHBACK_DATA and REDO_DATA correspond to sequential writes, while DB_DATA corresponds to random reads and writes.

Because of the difference between sequential I/O and random I/O, a better indication of I/O overhead is the number of I/O operations issued for flashback logs. The following statistics in V\$SYSSTAT can tell you the number of I/O operations your instance has issued for various purposes:

Column Name	Column Meaning
Physical write I/O request	The number of write operations issued for writing data blocks
physical read I/O request	The number of read operations issued for reading data blocks
redo writes	The number of write operations issued for writing to the redo log.
flashback log writes	The number of write operations issued for writing to flashback logs.

See *Oracle Database Reference* for more details on columns in the V\$SYSSTAT view.

Flashback Writer (RVWR) Behavior With I/O Errors

When flashback is enabled or when there are guaranteed restore points, the background process RVWR writes flashback data to flashback database logs in the flash recovery area. If RVWR encounters an I/O error, the following behavior is expected:

- If there are any guaranteed restore points defined, the instance will crash when RVWR encounters I/O errors.
- If no guaranteed restore points are defined, the instance will not crash when RVWR encounters I/O errors. There are 2 cases:
 - On a primary database, Oracle automatically disables flashback database while the database is open. All existing transactions and queries will proceed unaffected. This behavior is expected for both single instance and RAC.
 - On a physical or logical standby, RVWR will appear to be hung, retrying the I/O periodically. This may eventually hang the logical standby or the managed recovery of physical standby. (Oracle does not crash the standby instance because it does not want to crash the primary database in turn in max

protection mode.) To resolve the hang, a DBA can either perform a `SHUTDOWN ABORT` or issue `ALTER DATABASE FLASHBACK OFF`.

Performing Complete Restore and Recovery of Databases

This chapter introduces the tools that RMAN makes available for recovering your database from backup. It includes the following topics:

- [Database Restore and Recovery with RMAN: Overview](#)
- [Basic Database Restore and Recovery Scenarios](#)
- [Preparing and Planning Database Restore and Recovery](#)
- [RMAN RESTORE: Restoring Lost Database Files from Backup](#)

Database Restore and Recovery with RMAN: Overview

The focus of this chapter is on how you use RMAN and backups created with RMAN to return your database to normal operation after the loss of one or more database files needed for its normal operation. The database files that RMAN backs up and can recover are the control file, server parameter file, datafiles and archived redo log files.

The chapter is organized as follows:

- ["Basic Database Restore and Recovery Scenarios"](#) on page 6-3 presents some common database restore and recovery scenarios in their entirety. If one of these scenarios matches your situation, then you can follow the procedure outlined here. Even if your situation is not an exact match for any of these, a review of the scenarios can help you create a plan for your own recovery process.
- ["Preparing and Planning Database Restore and Recovery"](#) on page 6-5 provides a general outline you can use to plan a restore-and-recovery operation, including how to determine which files need restore and recovery and what steps need to be taken during the recovery. It also describes how to preview the backups that RMAN will use during the restore operation, and how to verify that the backups to be used in the restore operation are valid.
- ["RMAN RESTORE: Restoring Lost Database Files from Backup"](#) on page 6-13 presents a number of recovery procedures in isolation, such as how to restore a control file, an SPFILE, individual datafiles and redo logs, along with requirements in the event that you restore a particular type of file, such as steps you must take if you are restoring from a backup control file. If your exact restore and recovery scenario is not one of the basic scenarios outlined in the chapter, you can follow the processes outlined here to perform the individual tasks in your recovery plan.

The two most important RMAN commands used in database recovery are:

- `RESTORE`, which retrieves files from RMAN backups based on the contents of the RMAN repository
- `RECOVER`, which performs complete or point-in-time media recovery using available datafiles and redo logs.

Typically, you will set the state of the database appropriately for the data recovery operation to be performed, allocate or configure channels required to communicate with the disk and media manager, and then run a series of `RESTORE` and `RECOVER` commands. RMAN retrieves all needed files from backup and performs media recovery on all restored datafiles, to return your database to the desired state.

Scope and Limitations of this Chapter

This chapter introduces the techniques which will cover the most common restore and recovery scenarios. Anyone performing restore and recovery, even in complex scenarios not covered here, should be familiar with the techniques outlined in this chapter. Note, however, the following limitations on the scope of this discussion:

- Most of this chapter will focus on restore-and-recovery scenarios in which a media failure has damaged some or all of your database files, and your goal is to return your whole database to normal operation by restoring the damaged files from backup and recover all database changes in the redo log. While some more advanced types of recovery, such as point-in-time recovery of the whole database or individual tablespaces to recover from user errors, are mentioned in passing, [Chapter 7, "Performing Flashback and Database Point-in-Time Recovery"](#) discusses these techniques in detail.

See Also:

- [Chapter 7, "Performing Flashback and Database Point-in-Time Recovery"](#) for details on database point-in-time recovery (DBPITR) and Flashback Database
- *Oracle Database Backup and Recovery Advanced User's Guide* for details on tablespace point-in-time recovery (TSPITR)
- While there may be passing references to using RMAN with a recovery catalog, details on how use of RMAN changes with a recovery catalog are covered in *Oracle Database Backup and Recovery Advanced User's Guide*.
- The discussion in this chapter will be limited to Oracle databases running in a single-instance configuration. While RMAN can perform restore and recovery of databases in Real Application Clusters and Data Guard configurations, such scenarios are beyond the scope of this manual. After reviewing this chapter for the basics of backup and recovery, refer to *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* and *Oracle Data Guard Concepts and Administration* for more information about using RMAN in those contexts.

Restore and Recovery with Enterprise Manager

Enterprise Manager provides access to much of the database restore and recovery functionality provided by RMAN through a set of recovery wizards, that lead the DBA through a variety of recovery procedures based on an analysis of your database, your available backups and your data recovery objectives.

Using RMAN through Enterprise Manager, you can perform the simpler restore and recovery scenarios outlined in this chapter, as well as much more sophisticated restore

and recovery techniques such as point-in-time recovery and even use of the flashback features of the Oracle database, which allow for efficient repair of both media failure and user errors.

While the underlying functionality is the same, and the command-line client provides more flexibility, in many common situations, use of the Enterprise Manager interface to RMAN's restore and recovery features will be simpler than using the RMAN command line client directly.

See *Oracle Database 2 Day DBA* for more details on the restore and recovery features of Enterprise Manager.

Basic Database Restore and Recovery Scenarios

You can plan a strategy for recovering from most data losses using the process outlined in ["Preparing and Planning Database Restore and Recovery"](#) on page 6-5 and the task-specific procedures in ["RMAN RESTORE: Restoring Lost Database Files from Backup"](#) on page 6-13. However, some of the most common scenarios for database restore and recovery are presented in full here:

- [Restore and Recovery of a Whole Database: Scenario](#)
- [Restore and Complete Recovery of Individual Tablespaces or Datafiles: Scenario](#)

The procedures outlined here will restore the whole database or individual tablespaces to their original locations.

To use the procedures in this section, the following requirements must be met:

- The current control file must be intact.
- You must have the complete set of archived logs and incremental backups needed for media recovery of your available datafile backups.
- For any datafiles for which you have no backup, you must have a complete set of online and archived redo logs going back to the creation of that datafile. (With a complete set of redo logs, RMAN can re-create a datafile for which there is no backup, by creating an empty datafile and then re-applying all changes since the file was created as part of the recovery process.)

If automatic channels are configured, then RMAN allocates all channels configured for the available device types according to their parallelism settings. Otherwise, you must enclose your RESTORE and RECOVER commands in a RUN block, and begin by manually allocating the appropriate DISK or sbt channels. Otherwise, your RESTORE command will fail on attempting to retrieve backups from that device.

Restore and Recovery of a Whole Database: Scenario

In this scenario, you have a current control file and SPFILE but all datafiles are damaged or lost. You must restore and recover the whole database.

The database in this example has one read-only tablespace, `history`, which must be restored from backup but which does not need media recovery.

To restore and recover the database when the current control file is available:

1. After connecting to the target database, make sure the database is mounted.

```
RMAN> STARTUP MOUNT
```

2. Use the `SHOW ALL` command to see what channels are configured for access to backup devices. If automatic channels are *not* configured, then manually allocate one or more channels.
3. Restore the database using the `RESTORE DATABASE` command, and recover it using the `RECOVER DATABASE` command.
4. Examine the output to see if recovery was successful. If so, open the database.

This example performs restore and recovery of the database, using automatic channels.

```
RMAN> RESTORE DATABASE;  
RMAN> RECOVER DATABASE DELETE ARCHIVELOG MAXSIZE 25M;
```

The `RECOVER DATABASE` command as used here illustrates two useful options:

- `DELETE ARCHIVELOG` causes RMAN to delete restored log files after they have been applied to the datafiles, to save disk space.
- `MAXSIZE 25M` limits space occupied by restored logs at any given moment to 25MB. This gives you more control over disk space usage by the restored logs. Note that if a single archived redo log file is larger than the specified `MAXSIZE` value, you will get an error. You will have to try your command again with a larger `MAXSIZE` value.

Recovery of Databases with Read-Only Tablespaces

Read-only tablespaces may require special handling in a restore and recover operation. By default, the restore operation will skip read-only tablespaces. If a read-only tablespace is at the SCN where it became read-only after it is restored from backup, no redo will be applied to it when the rest of the database is recovered. You can force RMAN to restore any missing datafiles belonging to read-only tablespaces by using the `CHECK READONLY` option to the `RESTORE` command:

```
RMAN> RESTORE DATABASE CHECK READONLY;  
RMAN> RECOVER DATABASE DELETE ARCHIVELOG;
```

If RMAN completes the recovery without error, you can open the database:

```
RMAN> ALTER DATABASE OPEN;
```

Re-Creation of Temporary Tablespaces in Whole Database Restore and Recovery

After restore and recovery of a whole database, when the database is opened, any missing temporary tablespaces recorded in the control file version of the RMAN repository are re-created with their previous creation size, `AUTOEXTEND` and `MAXSIZE` attributes.

Note: Only temporary tablespaces that are missing are re-created. If a tempfile is still present at the location recorded in the RMAN repository but has an invalid header, then RMAN does not re-create the file.

If the tempfiles were originally created as Oracle-managed files, then they are re-created in the current `DB_CREATE_FILE_DEST`. Otherwise they are re-created at their previous locations.

If RMAN is unable to re-create the file due to an I/O error or some other cause, then the error is reported in the alert log and the database open operation continues.

Note: When using a recovery catalog, if the control file has been restored from backup, the RMAN repository stored in the restored control file is updated with information about the temporary tablespaces recorded in the recovery catalog version of the RMAN repository. This ensures that the most recent configuration of temporary tablespaces is re-created.

Restore and Complete Recovery of Individual Tablespaces or Datafiles: Scenario

In this scenario, the database is open, and some but not all of the datafiles are damaged. You want to restore and recover the damaged tablespace, while leaving the database open so that the rest of the database remains available.

Assume that, using the procedure described in "[Determining Which Database Files to Restore or Recover](#)" on page 6-6 to identify datafiles needing recovery, you discover that the damaged datafiles are from the tablespaces `users`.

This example restores and recovers the `users` tablespace, using configured channels and letting RMAN choose the backups to use in restoring the tablespace and any needed incremental backups and logs from disk or tape.

1. Connect to the target database and the recovery catalog database (if applicable), and make sure the database is mounted or open. For example:

2. Take the tablespaces affected offline using `ALTER TABLESPACE . . . OFFLINE IMMEDIATE` if they are not already offline.

```
RMAN> SQL 'ALTER TABLESPACE users OFFLINE IMMEDIATE';
```

3. Restore the tablespace or datafile with the `RESTORE` command, and recover it with the `RECOVER` command. (Use configured channels, or if desired, use a `RUN` block and allocate channels to improve performance of the `RESTORE` and `RECOVER` commands.)

```
RMAN> RESTORE TABLESPACE users;
```

```
RMAN> RECOVER TABLESPACE users;
```

4. If RMAN reported no errors during the recovery, then bring the tablespace back online:

```
RMAN> SQL 'ALTER TABLESPACE users ONLINE';
```

At this point the process is complete.

Preparing and Planning Database Restore and Recovery

While RMAN makes carrying out most database restore and recovery tasks much simpler, you still have to plan your database restore and recovery actions based on which database files have been lost and your recovery goal.

RMAN can make most of the important decisions about the restore process for you, but you may want to preview and even override its decisions in some circumstances. For example, if you know a given backup is unavailable, due to a tape being stored offsite or a device being inaccessible, you can direct RMAN to not use that backup during the restore process.

RMAN provides tools to let you preview which backups will be used in a restore, and to validate the contents of the backups to ensure that they can be used in future restore operations.

Database Restore and Recovery Procedure: Outline

The basic procedure for performing restore and recovery with RMAN is as follows:

1. Determine which database files must be restored from backup, and which backups (which specific tapes, or specific backup sets or image copies on disk) to use for the restore operation. The files to be restored may include the control file, SPFILE, archived redo log files, and datafiles.
2. Place the database in the state appropriate for the type of recovery that you are performing. For example, if you are recovering a single tablespace or datafile, then you can keep the database open and take the tablespace or datafile offline. If you are restoring all datafiles, then you must shut down the database and then mount it before you can perform the restore.
3. Restore lost database files from backup with the `RESTORE` command. You may restore files to their original locations, or you may have to restore them to other locations if, for instance, a disk has failed. You may also have to update the SPFILE if you have changed the control file locations, or the control file if you have changed the locations of datafiles or redo logs.
4. Perform media recovery on restored datafiles, if any, with the `RECOVER` command.
5. Perform any final steps required to make the database available for users again. For example, re-open the database if necessary, as happens when restoring lost control files, or bring offline tablespaces online if restoring and recovering individual tablespaces.

This outline is intended to encompass a wide range of different scenarios. Depending upon your situation, some of the steps described may not apply. (For example, you do not need to perform media recovery if the only file restored from backup is the SPFILE.) You will have to devise your final recovery plan based on your situation.

Determining Which Database Files to Restore or Recover

The methods of determining which files require restore or recovery depend upon the type of file that is lost. This section contains the following topics:

Recognizing a Lost Control File

It is generally obvious when the control file of your database must be restored. The database shuts down immediately if any of the control file copies becomes inaccessible and reports an error if you try to start it without a valid control file at each location specified in the `CONTROL_FILES` initialization parameter.

Loss of some but not all copies of your control file does not require recovery of the control file from backup. When one copy of the control file is lost, the database will automatically shut down. You can either copy an intact copy of the control file over the damaged or missing control file, or update the parameter file so that it does not refer to the damaged or missing control file. Once the `CONTROL_FILES` parameter references only present, intact copies of the control file, you can restart your database.

Note that if you restore the control file from backup, you must perform media recovery of the whole database and then perform an `OPEN RESETLOGS`, even if no datafiles have to be restored.

Identifying Datafiles Requiring Media Recovery

When and how to recover depends on the state of the database and the location of its datafiles. To determine which if any files require media recovery, use the following procedure:

1. Start SQL*Plus and connect to the target database. For example, issue the following to connect to `trgt`:

```
% sqlplus 'SYS/oracle@trgt AS SYSDBA'
```

2. Determine the status of the database by executing the following SQL query:

```
SELECT STATUS FROM V$INSTANCE;
```

If the status is `OPEN`, then the database is open. However, some datafiles may require media recovery.

3. Query the `V$DATAFILE_HEADER` view to determine the status of your datafiles. Run the following SQL statements to check the datafile headers:

```
COL FILE# FORMAT 999
COL STATUS FORMAT A7
COL ERROR FORMAT A10
COL TABLESPACE_NAME FORMAT A10
COL NAME FORMAT A30

SELECT FILE#, STATUS, ERROR, RECOVER, TABLESPACE_NAME, NAME
FROM V$DATAFILE_HEADER
WHERE RECOVER = 'YES' OR (RECOVER IS NULL AND ERROR IS NOT NULL);
```

Each row returned represents a datafile that either requires media recovery or has an error requiring a restore. Check the `RECOVER` and `ERROR` columns. `RECOVER` indicates whether a file needs media recovery, and `ERROR` indicates whether there was an error reading and validating the datafile header.

If `ERROR` is not `NULL`, then the datafile header cannot be read and validated. Check for a temporary hardware or operating system problem causing the error. If there is no such problem, you must restore the file or switch to a copy.

If the `ERROR` column is `NULL` and the `RECOVER` column is `YES`, then the file requires media recovery (and may also require a restore from backup).

Note: Because `V$DATAFILE_HEADER` only reads the header block of each datafile, it does not detect all problems that require the datafile to be restored. For example, you cannot tell whether a datafile contains corrupt data blocks using `V$DATAFILE_HEADER`.

You can also query `V$RECOVER_FILE` to list datafiles requiring recovery by datafile number with their status and error information.

```
SELECT FILE#, ERROR, ONLINE_STATUS, CHANGE#, TIME
FROM V$RECOVER_FILE;
```

Note: You cannot use `V$RECOVER_FILE` with a control file restored from backup or a control file that was re-created after the time of the media failure affecting the datafiles. A restored or re-created control file does not contain the information needed to update `V$RECOVER_FILE` accurately.

To find datafile and tablespace names, you can also perform useful joins using the datafile number and the `V$DATAFILE` and `V$TABLESPACE` views. For example:

```

COL DF# FORMAT 999
COL DF_NAME FORMAT A35
COL TBSP_NAME FORMAT A7
COL STATUS FORMAT A7
COL ERROR FORMAT A10
COL CHANGE# FORMAT 99999999
SELECT r.FILE# AS df#, d.NAME AS df_name, t.NAME AS tbspc_name,
       d.STATUS, r.ERROR, r.CHANGE#, r.TIME
FROM V$RECOVER_FILE r, V$DATAFILE d, V$TABLESPACE t
WHERE t.TS# = d.TS#
AND d.FILE# = r.FILE#
;

```

The ERROR column identifies the problem for each file requiring recovery.

See Also: *Oracle Database Reference* for information about the V\$ views

Recovery of Read-Only Tablespaces

Recovery is not needed on any **read-only tablespace** during crash or instance recovery. During startup, recovery verifies that each online read-only datafile does not need media recovery, by checking whether the file was not restored from a backup taken before it was made read-only.

Note: If you restore a read-only tablespace from a backup taken before the tablespace was made read-only, then you cannot access the tablespace until you perform media recovery on it.

Determining your DBID

In situations requiring the recovery of your SPFILE or control file from autobackup, such as disaster recovery when you have lost all database files, you will need to use your DBID. Your DBID should be recorded along with other basic information about your database, as recommended in "[Deciding Between ARCHIVELOG and NOARCHIVELOG Mode](#)" on page 2-7.

If you do not have a record of the DBID of your database, there are two places you can find it without opening your database.

- The DBID is used in forming the filename for the control file autobackup. Locate that file, and then refer to "[Configuring the Control File Autobackup Format](#)" on page 3-12 to see where the DBID appears in the filename.
- If you have any text files that preserve the output from an RMAN session, the DBID is displayed by the RMAN client when it starts up and connects to your database. Typical output follows:

```

% rman TARGET /
Recovery Manager: Release 10.2.0.1.0 - Production on Sun Jun 12 02:41:03 2005

Copyright (c) 1982, 2005, Oracle. All rights reserved.

connected to target database: RDBMS (DBID=774627068)

RMAN>

```

Previewing Backups Used in Restore Operations: RESTORE PREVIEW

The RESTORE command supports a PREVIEW option, which identifies the backups (backup sets or image copies, on disk or sequential media like tapes) required to carry out a given restore operation, based on the information in the RMAN repository. Use RESTORE... PREVIEW when planning your restore and recovery operation, to ensure that all required backups are available or to identify situations in which you may want to direct RMAN to use or avoid specific backups.

For example, RESTORE... PREVIEW output may indicate that, during a RESTORE operation, RMAN will request a backup from a tape during the restore process which you know is temporarily unavailable. You can then use the CHANGE... UNAVAILABLE command (described in "[Marking a Backup AVAILABLE or UNAVAILABLE](#)" on page 8-13) to set the backup status to UNAVAILABLE. If you then run RESTORE... PREVIEW again, RMAN will show you the backups it would use to perform a restore operation without using the unavailable backup.

In some cases, a backup may be listed as AVAILABLE but it is in fact vaulted, that is, stored remotely and requires retrieval before it can be used in a restore. If RMAN selects such a backup for use in a restore operation, the operation fails with an error. RESTORE... PREVIEW lets you identify any backups that are stored remotely, and RESTORE...PREVIEW RECALL is used to request that backups needed for a RESTORE operation but that are stored remotely be recalled from remote storage.

Using RESTORE... PREVIEW

RESTORE ... PREVIEW can be applied to any RESTORE operation to create a detailed report of every backup to be used in the requested RESTORE operation, as well as the necessary target SCN for recovery after the RESTORE operation is complete. Here are a few examples of RESTORE commands using the PREVIEW option:

```
RESTORE DATABASE PREVIEW;
RESTORE TABLESPACE users PREVIEW;
RESTORE DATAFILE 3 PREVIEW;
RESTORE ARCHIVELOG FROM LOGSEQ 200 PREVIEW;
RESTORE ARCHIVELOG FROM TIME 'SYSDATE-7' PREVIEW;
RESTORE ARCHIVELOG FROM SCN 234546 PREVIEW;
```

RESTORE... PREVIEW output is in the same format as the output of the LIST command. See *Oracle Database Backup and Recovery Reference* for details on interpreting the output of RESTORE... PREVIEW.

Using RESTORE... PREVIEW SUMMARY

If the detailed report produced by RESTORE... PREVIEW provides more information than is needed, use the RESTORE... PREVIEW SUMMARY option to suppress much of the detail about specific files used and affected by the restore process. Here are some examples of RESTORE used with the PREVIEW SUMMARY option:

```
RESTORE DATABASE PREVIEW SUMMARY;
RESTORE TABLESPACE users PREVIEW SUMMARY;
RESTORE DATAFILE 3 PREVIEW SUMMARY;
RESTORE ARCHIVELOG FROM SCN 234546 PREVIEW SUMMARY;
```

RESTORE... PREVIEW SUMMARY reports are in the same format as the output from the LIST SUMMARY command. See *Oracle Database Backup and Recovery Reference* for details on interpreting the output of RESTORE... PREVIEW SUMMARY.

Using RESTORE... PREVIEW RECALL

RESTORE... PREVIEW RECALL can be used with any RESTORE operation, in cases a restore fails due to a needed backup being stored remotely.

The following command shows the output in a case where RESTORE ... PREVIEW indicates that a needed backup is stored remotely:

```
RMAN> restore archivelog all preview;
```

```
Starting restore at 10-JUN-05
using channel ORA_DISK_1
using channel ORA_SBT_TAPE_1
```

List of Backup Sets

=====

BS Key	Size	Device Type	Elapsed Time	Completion Time
31	12.75M	SBT_TAPE	00:00:02	10-JUN-05
BP Key: 33 Status: AVAILABLE Compressed: NO Tag: TAG20050610T152755				
Handle: 15gmknbs Media: /v1,15gmknbs				

List of Archived Logs in backup set 31

Thrd	Seq	Low SCN	Low Time	Next SCN	Next Time
1	1	221154	06-JUN-05	222548	06-JUN-05
1	2	222548	06-JUN-05	222554	06-JUN-05
1	3	222554	06-JUN-05	222591	06-JUN-05
1	4	222591	06-JUN-05	246629	07-JUN-05
1	5	246629	07-JUN-05	262451	10-JUN-05

BS Key	Size	Device Type	Elapsed Time	Completion Time
32	256.00K	SBT_TAPE	00:00:01	10-JUN-05
BP Key: 34 Status: AVAILABLE Compressed: NO Tag: TAG20050610T153105				
Handle: 17gmknhp_1_1 Media: /v1,17gmknhp_1_1				

List of Archived Logs in backup set 32

Thrd	Seq	Low SCN	Low Time	Next SCN	Next Time
1	6	262451	10-JUN-05	262547	10-JUN-05
1	7	262547	10-JUN-05	262565	10-JUN-05

List of remote backup files

=====

Handle: 15gmknbs Media: /v1,15gmknbs

The "List of remote backup files" at the end of the output identifies the backups that are stored remotely. In such a case, using RESTOREARCHIVELOG ALL PREVIEW RECALL initiates recall for the remote backups and produces the following output:

```
RMAN> restore archivelog all preview recall;
```

```
Starting restore at 10-JUN-05
using channel ORA_DISK_1
using channel ORA_SBT_TAPE_1
```

List of Backup Sets

=====

```

BS Key   Size      Device Type Elapsed Time Completion Time
-----
31       12.75M    SBT_TAPE    00:00:02    10-JUN-05
          BP Key: 33   Status: AVAILABLE Compressed: NO   Tag: TAG20050610T152755
          Handle: 15gmknbs   Media: /v1,15gmknbs
    
```

List of Archived Logs in backup set 31

Thrd	Seq	Low SCN	Low Time	Next SCN	Next Time
1	1	221154	06-JUN-05	222548	06-JUN-05
1	2	222548	06-JUN-05	222554	06-JUN-05
1	3	222554	06-JUN-05	222591	06-JUN-05
1	4	222591	06-JUN-05	246629	07-JUN-05
1	5	246629	07-JUN-05	262451	10-JUN-05

```

BS Key   Size      Device Type Elapsed Time Completion Time
-----
32       256.00K    SBT_TAPE    00:00:01    10-JUN-05
          BP Key: 34   Status: AVAILABLE Compressed: NO   Tag: TAG20050610T153105
          Handle: 17gmknhp_1_1   Media: /v1,17gmknhp_1_1
    
```

List of Archived Logs in backup set 32

Thrd	Seq	Low SCN	Low Time	Next SCN	Next Time
1	6	262451	10-JUN-05	262547	10-JUN-05
1	7	262547	10-JUN-05	262565	10-JUN-05

Initiated recall for the following list of remote backup files

=====

Handle: 15gmknbs Media: /v1,15gmknbs

Finished restore at 10-JUN-05

You can repeat the RESTORE... PREVIEW command until no backups needed for the restore are reported as remote.

You can append RECALL to any RESTORE... PREVIEW command, as in these examples:

```

RESTORE TABLESPACE users PREVIEW RECALL;
RESTORE DATAFILE 3 PREVIEW RECALL;
    
```

Validating the Restore of Backups: RESTORE VALIDATE and VALIDATE BACKUPSET

The RESTORE ... VALIDATE and VALIDATE BACKUPSET commands test whether you can restore from your backups. You can test the availability of usable backups for any desired RESTORE operation, or test the contents of a specific backup for use in RESTORE operations. The contents of the backups are actually read and validated for corruption to ensure that the objects to be restored can be restored from them. You have these options:

- RESTORE . . . VALIDATE tests whether RMAN can restore a specific object from a backup. RMAN chooses which backups to use.
- VALIDATE BACKUPSET tests the validity of a backup set that you specify.

See Also:

- *Oracle Database Backup and Recovery Reference* for RESTORE syntax
- *Oracle Database Backup and Recovery Reference* for VALIDATE syntax

Validating Restore from Backup with RESTORE ... VALIDATE

You can enter any valid RESTORE command specifying the VALIDATE clause to test whether usable backups for that RESTORE operation are available. When validating backups with RESTORE . . . VALIDATE, the database can be mounted or open.

This example illustrates validating the restore of the backup control file, SYSTEM tablespace, and all archived logs:

```
RESTORE CONTROLFILE VALIDATE;
RESTORE TABLESPACE SYSTEM VALIDATE;
RESTORE ARCHIVELOG ALL VALIDATE;
RESTORE DATAFILE 4,5,6 VALIDATE;
```

Note: You do not have to take datafiles offline when validating the restore of datafiles, because validation of backups of the datafiles only reads the backups and does not affect the production datafiles.

If you see error messages in the output and the following message, then RMAN cannot restore one or more of the specified files from your available backups:

```
RMAN-06026: some targets not found - aborting restore
```

If you see an error message stack and output similar to the following, for example, then RMAN encountered a problem reading the specified backup:

```
RMAN-03009: failure of restore command on c1 channel at 12-DEC-01 23:22:30
ORA-19505: failed to identify file "oracle/dbs/1fafv9gl_1_1"
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

If you do not see an error stack, then RMAN successfully tested restore of the specified objects from the available backups and should be able to restore these objects successfully during a real restore and recovery operation.

Validating Backup Sets with VALIDATE BACKUPSET

The BACKUP VALIDATE command requires that you know the primary keys of the backup sets that you want to validate.

To specify which backup sets to validate:

Find the backup sets that you want to validate by running LIST commands, noting primary keys. For example:

```
LIST BACKUP;
```

Validate the restore of the backup sets, referencing them by the primary keys. This example validates the restore of backup sets 56 and 57:

```
VALIDATE BACKUPSET 56,57;
```

If the output contains the message " validation complete", then RMAN successfully validated the restore of the specified backup set. For example:

```
using channel ORA_DISK_1
channel ORA_DISK_1: starting validation of archive log backupset
channel ORA_DISK_1: restored backup piece 1
piece handle=/oracle/dbs/0mdg9v8l_1_1 tag=TAG20020208T155604 params=NULL
channel ORA_DISK_1: validation complete
```

RMAN RESTORE: Restoring Lost Database Files from Backup

This section discusses how to restore the different types of database file backed up by RMAN. Once you have an overall plan for restoring the lost parts of your database, look here for details on how to execute the individual tasks in your plan.

This section contains the following topics:

- [Restoring the Control File from Backup](#)
- [Restoring the Server Parameter File \(SPFILE\) from Backup](#)
- [Restoring and Recovering Datafiles and Tablespaces](#)
- [Restoring Archived Redo Logs from Backup](#)

Restoring the Control File from Backup

Loss or corruption of all copies of your control file requires restore of the control file from backup. The `RESTORE CONTROLFILE` command is used to restore the control file.

Note: After restoring the control files of your database from backup, you must perform complete media recovery of the database as described in "[Performing Media Recovery of a Restored Database, Tablespace or Datafile](#)" on page 6-18, and then open your database with the `RESETLOGS` option. The only exception is the case described in "[Restore of the Control File to a New Location](#)" on page 6-15, where you restore your control file to a location not listed in the `CONTROL_FILES` initialization parameter. In that case, you create a copy of your control file in the specified location without touching your running database.

RMAN can restore the control file to its default location (determined by rules described in the following section) or to one or more different locations of your choice, using the `RESTORE CONTROLFILE... TO destination` option.

Default Destination for Restore of the Control File

When restoring the control file, the default destination is all of the locations defined in the `CONTROL_FILES` initialization parameter. If you do not set the `CONTROL_FILES` initialization parameter, the database uses the same rules to determine the destination for the restored control file as it uses when creating a control file if the `CONTROL_FILES` parameter is not set. These rules are described in *Oracle Database SQL Reference* in the description of the `CREATE CONTROLFILE` statement.

Restore of the Control File from Control File Autobackup

If you are not using a recovery catalog, you must restore your control file from an autobackup. If you want to restore the control file from autobackup, the database must be in a NOMOUNT state. You must first set the DBID for your database, and then use the `RESTORE CONTROLFILE FROM AUTOBACKUP` command:

```
RMAN> SET DBID 320066378;
RMAN> RUN {
    SET CONTROLFILE AUTOBACKUP FORMAT
        FOR DEVICE TYPE DISK TO 'autobackup_format';
    RESTORE CONTROLFILE FROM AUTOBACKUP;
}
```

RMAN uses the autobackup format and DBID to determine where to hunt for the control file autobackup. If one is found, RMAN restores the control file from that backup to all of the control file locations listed in the `CONTROL_FILES` initialization parameter.

For information on how to determine the correct value for `autobackup_format`, see the description of `CONFIGURE CONTROLFILE AUTOBACKUP FORMAT` in the entry for `CONFIGURE` in *Oracle Database Backup and Recovery Reference*

See "[Determining your DBID](#)" on page 6-8 for details on how to determine your DBID.

Restore of the Control File When Using a Flash Recovery Area

The commands used for restoring your control file are the same, whether or not you are using a flash recovery area. However, if you are using a flash recovery area, RMAN updates a control file restored from backup, by performing an implicit crosscheck of all disk-based backups and image copies listed in the control file, and cataloging any backups in the flash recovery area that are not recorded in the restored control file. As a result the restored control file has a complete and accurate record of all backups in your flash recovery area and any other backups that were known to the control file at the time of the backup. This improves the usefulness of the restored control file in the restoration of the rest of your database.

Tape backups are not automatically crosschecked after the restore of a control file. If you are using tape backups, then after restoring the control file and mounting the database you must crosscheck the backups on tape, as shown in this example:

```
RMAN> CROSSCHECK BACKUP DEVICE TYPE SBT;
```

Restoring a Control File When Using a Recovery Catalog

Restoring a lost control file from autobackup is easier when using a recovery catalog than when using only the control file to store the RMAN repository. The recovery catalog contains a complete record of your backups, including backups of the control file. Therefore, you do not have to specify your DBID or control file autobackup format.

To restore the control file, connect RMAN to the target database and the recovery catalog, and bring the database to NOMOUNT state. Then issue the `RESTORE CONTROLFILE` command with no parameters, as in this example:

```
% rman TARGET rman/rman CATALOG catdb/catdb
RMAN> RESTORE CONTROLFILE;
```

The restored control file is written to all locations listed in the `CONTROL_FILES` initialization parameter.

For more details on restrictions on using `RESTORE CONTROLFILE` in different situations, see the discussion of `RESTORE CONTROLFILE` in *Oracle Database Backup and Recovery Reference*.

Restore of the Control File From a Known Location

You can restore the control file from a known control file copy using this form of the command:

```
RMAN> RESTORE CONTROLFILE from 'filename';
```

The control file copy found at the location specified by *filename* will be written to all locations listed in the `CONTROL_FILES` initialization parameter.

Restore of the Control File to a New Location

One way to restore the control file to one or more new locations is to change the `CONTROL_FILES` initialization parameter, and then use the `RESTORE CONTROLFILE` command with no arguments to restore the control file to the default locations. For example, if you are restoring your control file after a disk failure made some but not all `CONTROL_FILES` locations unusable, you can change `CONTROL_FILES` to replace references to the failed disk with pathnames pointing to another disk, and then run `RESTORE CONTROLFILE` with no arguments.

You can also restore the control file to any location you choose other than the `CONTROL_FILES` locations, by using the form `RESTORE CONTROLFILE TO 'filename' [FROM AUTOBACKUP]`:

```
RESTORE CONTROLFILE TO '/tmp/my_controlfile';
```

You can perform this operation with the database in `NOMOUNT`, `MOUNT` or `OPEN` states, because you are not overwriting any of the control files currently in use. Any existing file named '*filename*' is overwritten. After restoring the control file to a new location, you can then update the `CONTROL_FILES` initialization parameter to include the new location.

See Also: *Oracle Database Backup and Recovery Reference* for `RESTORE CONTROLFILE` syntax.

Limitations When Using a Backup Control File

After you restore your database using a backup control file, you **must** run `RECOVER DATABASE` and perform an `OPEN RESETLOGS` on the database.

For more details on restrictions on using `RESTORE CONTROLFILE` in different scenarios (such as when using a recovery catalog, or restoring from a specific backup), see the discussion of `RESTORE CONTROLFILE` in *Oracle Database Backup and Recovery Reference*.

Restoring the Server Parameter File (SPFILE) from Backup

If you lose your server parameter file (SPFILE), RMAN can restore it to its default location or to a location of your choice.

Unlike the loss of the control file, the loss of your SPFILE does not cause your instance to immediately stop. Your instance may continue operating, although you will have to shut it down and restart it after restoring the SPFILE.

Note the following when restoring the SPFILE:

- If the instance is already started with the server parameter file, then you cannot overwrite the existing server parameter file.
- When the instance is started with a client-side initialization parameter file, RMAN restores the SPFILE to the default SPFILE location if the TO clause is not used. The default location is platform-specific (for example, `?/dbs/spfile.ora` on Linux).
- Restoring the SPFILE is one situation in which a recovery catalog can simplify your recovery procedure, because you can avoid the step of having to record and remember your DBID. This procedure assumes that you are not using a recovery catalog.

RMAN can also create a client-side initialization parameter file based on a backup of an SPFILE.

To restore the server parameter file:

1. If the database is up at the time of the loss of the SPFILE, connect to the target database. For example, run:

```
% rman TARGET /
```

If the database is not up when the SPFILE is lost, and you are not using a recovery catalog, then you must set the DBID of the target database. See ["Determining your DBID"](#) on page 6-8 for details on determining your DBID.

2. Shut down the instance and restart it without mounting. When the SPFILE is not available, RMAN starts the instance with a dummy parameter file. For example:

```
RMAN> STARTUP FORCE NOMOUNT;
```

3. Restore the server parameter file. If restoring to the default location, then run:

```
RMAN> RESTORE SPFILE FROM AUTOBACKUP;
```

If restoring to a nondefault location, then you could run commands as in the following example:

```
RMAN> RESTORE SPFILE TO '/tmp/spfileTEMP.ora' FROM AUTOBACKUP;
```

4. Restart the instance with the restored file. If restarting with a server parameter file in a nondefault location, then create a new client-side initialization parameter file with the single line `SPFILE=new_location`, where `new_location` is the path name of the restored server parameter file. Then, restart the instance with the client-side initialization parameter file.

For example, create a file `/tmp/init.ora` which contains the single line:

```
SPFILE=/tmp/spfileTEMP.ora
```

Then use this RMAN command, to restart the instance based on the restored SPFILE:

```
RMAN> STARTUP FORCE PFILE=/tmp/init.ora; # startup with /tmp/spfileTEMP.ora
```

Restore of the SPFILE from the Control File Autobackup

If you have configured control file autobackups, the SPFILE is backed up with the control file whenever an autobackup is taken.

If you want to restore the SPFILE from the autobackup, you must first set the DBID for your database, and then use the `RESTORE SPFILE FROM AUTOBACKUP` command.

The procedure is similar to restoring the control file from autobackup. You must first set the DBID for your database, and then use the `RESTORE CONTROLFILE FROM AUTOBACKUP` command:

```
RMAN> SET DBID 320066378;
RMAN> RUN {
    SET CONTROLFILE AUTOBACKUP FORMAT
        FOR DEVICE TYPE DISK TO 'autobackup_format';
    RESTORE SPFILE FROM AUTOBACKUP;
}
```

RMAN uses the autobackup format and DBID to hunt for control file autobackups, and if a control file autobackup is found, restores the SPFILE from that backup to its default location.

For information on how to determine the correct value for `autobackup_format`, see the description of `CONFIGURE CONTROLFILE AUTOBACKUP FORMAT` in the entry for `CONFIGURE` in *Oracle Database Backup and Recovery Reference*

See "[Determining your DBID](#)" on page 6-8 for details on how to determine your DBID.

Creating a Client-Side Initialization Parameter File (PFILE) with RMAN

You can also restore the server parameter file as a client-side initialization parameter file with the `TO PFILE 'filename'` clause. The filename you specify should be on a file system accessible from the host where the RMAN client is running. This file need not be accessible directly from the host running the instance. This command creates a PFILE called `/tmp/initTEMP.ora` on the system running the RMAN client:

```
RMAN> RESTORE SPFILE TO PFILE '/tmp/initTEMP.ora';
```

To restart the instance using the client-side PFILE, use the following command, again running RMAN on the same client machine:

```
RMAN> STARTUP FORCE PFILE='/tmp/initTEMP.ora';
```

Restoring and Recovering Datafiles and Tablespaces

Restoring a tablespace to its original location and performing media recovery on it is described in "[Restore and Complete Recovery of Individual Tablespaces or Datafiles: Scenario](#)" on page 6-5. However, you may need to restore a datafile to a location other than its original location if, for example, the disk containing the original location of the datafiles has failed.

Restoring Datafiles from Backup to a New Location

The important step in restoring datafiles from backup to a new location is to update the control file to reflect the new locations of the datafiles. The following example shows the use of the `RMAN SET NEWNAME` command to specify the new names, and the `SWITCH` command to update the control file to start referring to the datafiles by their new names.

As with restoring datafiles from backup to their original locations, you should take the affected tablespaces offline at the start of restoring datafiles from backup to a new location.

Then, create a `RUN` block to encompass your `RESTORE` and `RECOVER` commands. For each file to be moved to a new location, use the `SET NEWNAME` command to specify the new location for that file.

Then, still within the RUN block, run the RESTORE TABLESPACE or RESTORE DATAFILE as normal. RMAN restores each datafile to the location specified with SET NEWNAME, rather than its original location.

After the RESTORE command but before the RECOVER command in your RUN block, use a SWITCH command to update the control file with the new filenames of the datafiles. The SWITCH command is equivalent to the SQL statement ALTER DATABASE RENAME FILE. SWITCH DATAFILE ALL updates the control file to reflect the new names for all datafiles for which a SET NEWNAME has been issued in the RUN block.

This example restores the datafiles in tablespaces users and tools to a new location, then performs recovery. Assume that the old datafiles were stored in directory /olddisk and the new ones will be stored in /newdisk.

```

RUN
{
  SQL 'ALTER TABLESPACE users OFFLINE IMMEDIATE';
  SQL 'ALTER TABLESPACE tools OFFLINE IMMEDIATE';
  # specify the new location for each datafile
  SET NEWNAME FOR DATAFILE '/olddisk/users01.dbf' TO
                                '/newdisk/users01.dbf';
  SET NEWNAME FOR DATAFILE '/olddisk/tools01.dbf' TO
                                '/newdisk/tools01.dbf';
  # to restore to an ASM disk group named dgroup, use:
  # SET NEWNAME FOR DATAFILE '/olddisk/trgt/tools01.dbf'
  #   TO '+dgroup';
  RESTORE TABLESPACE users, tools;
  SWITCH DATAFILE ALL; # update control file with new filenames
  RECOVER TABLESPACE users, tools;
}

```

If recovery is successful, then bring the tablespaces online:

```

SQL 'ALTER TABLESPACE users ONLINE';
SQL 'ALTER TABLESPACE tools ONLINE';

```

See Also: *Oracle Database Backup and Recovery Reference* for SWITCH syntax

Performing Media Recovery of a Restored Database, Tablespace or Datafile

Media recovery reapplies all changes from the archived and online redo logs and available incremental backups to datafiles restored from backup.

The simplest way to perform media recovery is to use the RECOVER DATABASE command, with no arguments:

```

RMAN> RECOVER DATABASE;

```

You can also perform media recovery of individual tablespaces or datafiles, or skip certain tablespaces while recovering the rest of the database, as shown in the following examples:

```

RMAN> RECOVER DATABASE SKIP TABLESPACE users;

```

```

RMAN> RECOVER TABLESPACE users, tools;

```

```

RMAN> RECOVER DATAFILE '/newdisk/users01.dbf', '/newdisk/tools01.dbf';

```

```

RMAN> RECOVER DATAFILE 4;

```

RMAN will restore from backup any archived redo logs required during the recovery operation. If backups are stored on a media manager, note that channels must be configured in advance or a RUN block with `ALLOCATE CHANNEL` commands must be used to enable access to backups stored there.

One very useful option in managing disk space associated with these restored files is the `DELETE ARCHIVELOG` option, which causes the deletion of restored archived redo logs from disk once they are no longer needed for the `RECOVER` operation:

```
RMAN> RECOVER TABLESPACE users, tools DELETE ARCHIVELOG;
```

Note that when RMAN restores archived redo log files to the flash recovery area in order to perform a `RECOVER` operation, the restored logs are automatically deleted after they are applied to the datafiles, even if you do not use the `DELETE ARCHIVELOG` option.

See *Oracle Database Backup and Recovery Reference* for more details on options for the `RECOVER` command.

Restore and Recover of a Single Datafile to a New Location:Example

This procedure restores a single datafile to a new location and perform media recovery on it. This lets you restore and recover if the old location is inaccessible because of a problem such as a media failure.

```
RUN {
  SET NEWNAME FOR DATAFILE 3 to 'new_location';
  RESTORE DATAFILE 3;
  SWITCH DATAFILE 3;
  RECOVER DATAFILE 3;
}
```

If you want to store a datafile to a new Oracle Managed Files location, you can use this form of the command:

```
RUN {
  SET NEWNAME FOR DATAFILE 3 to NEW;
  RESTORE DATAFILE 3;
  SWITCH DATAFILE 3;
  RECOVER DATAFILE 3;
}
```

Oracle will store the restored file in an OMF location, generating a filename for it.

Restoring Archived Redo Logs from Backup

RMAN will restore archived redo log files from backup automatically as needed to perform recovery.

However, you can also restore archived redo logs manually if you wish, in order to save the time needed to restore these files later during the `RECOVER` command, or if you want to store the restored archived redo log files in some new location.

By default, RMAN restores archived redo logs with names constructed using the `LOG_ARCHIVE_FORMAT` and the `LOG_ARCHIVE_DEST_1` parameters of the target database. These parameters are combined in a platform-specific fashion to form the name of the restored archived log.

Restoring Archived Redo Logs to a New Location

You can override the default location for restored archived redo logs with the `SET ARCHIVELOG DESTINATION` command. This command manually stages 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 initialization parameter file.

To restore archived redo logs to a new location:

1. After connecting to the target database, make sure the database is mounted or open.
2. Perform the following operations within a `RUN` block, as shown in the following example script:
 - a. Specify the new location for the restored archived redo logs using `SET ARCHIVELOG DESTINATION`.
 - b. Restore the archived redo logs.

This example restores all backup archived logs to a new location:

```
RUN
{
  SET ARCHIVELOG DESTINATION TO '/oracle/temp_restore';
  RESTORE ARCHIVELOG ALL;
  # restore and recover datafiles as needed
  .
  .
  .
}
```

Restoring Archived Redo Logs to Multiple Locations

You can specify restore destinations for archived logs multiple times in one `RUN` block, in order to distribute restored logs among several destinations. (You cannot, however specify multiple destinations simultaneously to produce multiple copies of the same log during the restore operation.) You can use this feature to manage disk space used to contain the restored logs.

This example restores 300 archived redo logs from backup, distributing them across the directories `/fs1/tmp`, `/fs2/tmp`, and `/fs3/tmp`:

```
RUN
{
  # Set a new location for logs 1 through 100.
  SET ARCHIVELOG DESTINATION TO '/fs1/tmp';
  RESTORE ARCHIVELOG FROM SEQUENCE 1 UNTIL SEQUENCE 100;
  # Set a new location for logs 101 through 200.
  SET ARCHIVELOG DESTINATION TO '/fs2/tmp';
  RESTORE ARCHIVELOG FROM SEQUENCE 101 UNTIL SEQUENCE 200;
  # Set a new location for logs 201 through 300.
  SET ARCHIVELOG DESTINATION TO '/fs3/tmp';
  RESTORE ARCHIVELOG FROM SEQUENCE 201 UNTIL SEQUENCE 300;
  # restore and recover datafiles as needed
  .
  .
  .
}
```

When you issue a `RECOVER` command, RMAN finds the needed restored archived logs automatically across the destinations to which they were restored, and applies them to the datafiles.

Performing Flashback and Database Point-in-Time Recovery

It is sometimes necessary to return some objects in your database or the entire database to a previous state, following the effects of a mistaken database update. For example, a user or DBA might erroneously delete or update the contents of one or more tables, drop database objects that are still needed for a risky operation such as an update to an application or a large batch update might fail.

In addition to point-in-time restore and recovery of the entire database, Oracle provides a group of features known as Oracle Flashback Technology, that are often faster than point-in-time recovery, and less disruptive to database availability.

This chapter presents a guide to investigating unwanted database changes, and selecting and carrying out an appropriate recovery strategy based upon Oracle Flashback Technology and database backups. It includes the following topics:

- [About Point-in-Time Recovery and Flashback Features](#)
- [Oracle Flashback Query: Recovering at the Row Level](#)
- [Oracle Flashback Table: Returning Individual Tables to Past States](#)
- [Oracle Flashback Drop: Undo a DROP TABLE Operation](#)
- [Reversing Database Changes with Flashback Database](#)
- [Performing Database Point-In-Time Recovery](#)

About Point-in-Time Recovery and Flashback Features

The most basic solution after unwanted database changes is database point-in-time recovery, in which you restore the database from backup and then apply redo logs to recreate all changes up to a point in time before the unwanted change.

Oracle Flashback Technology provides several alternatives to a set of features that support viewing past states of data, and winding data back and forth in time, without requiring the restore of the database from backup. Depending upon the changes to your database, Flashback Technology can often reverse the unwanted changes more quickly and with less impact on the availability of the rest of your database.

About Database Point-in-Time Recovery

Database point-in-time recovery (DBPITR) is a process that works at the physical level to return the datafiles to their state at a desired target time in the past. In an RMAN DBPITR operation, you specify a target point in time, and RMAN restores the database from backups prior to that time, and then applies incremental backups and performs

media recovery to recreate all changes between the time of the datafile backups and the target time. If your backup strategy is properly designed and your database is running in ARCHIVELOG mode then DBPITR is an option in nearly all circumstances.

RMAN simplifies DBPITR greatly compared to user-managed DBPITR. Given a target SCN, datafiles are restored from backup and recovered efficiently using incremental backups and archived logs available on disk or tape, with no intervention from the user. However, there are some disadvantages to point-in-time recovery:

- You cannot return only selected objects to their earlier state, only the entire database.
- Your entire database is unavailable during the database point-in-time recovery process.
- Point-in-time recovery can be time-consuming, because all datafiles must be restored, and redo logs and incremental backups must be restored from backup and used to recover the datafiles. If needed backups are on tape, this process can take even longer.

Note: If you know that unwanted database changes are extensive but confined to certain tablespaces in your database, **tablespace point-in-time recovery (TSPITR)** can return a subset of your tablespaces to an earlier SCN while the unaffected tablespaces of your database continue to be available. TSPITR is an advanced technique described in *Oracle Database Backup and Recovery Advanced User's Guide*.

Oracle Flashback Technology: Alternatives to Point-in-Time Recovery

The flashback features of Oracle are more efficient than media recovery in most circumstances in which they are available. They can also be used to investigate past states of Most Flashback Technology features operate at the logical level, viewing and manipulating database objects, as follows:

- **Oracle Flashback Query** lets you specify a target time and then run queries against your database, viewing results as they would have appeared at that time. To recover from an unwanted change like an erroneous update to a table, a user could choose a target time before the error and run a query to retrieve the contents of the lost rows.
- **Oracle Flashback Version Query** lets you view all the versions of all the rows that ever existed in one or more tables in a specified time interval. You can also retrieve metadata about the differing versions of the rows, including start time, end time, operation, and transaction ID of the transaction that created the version. This feature can be used both to recover lost data values and to audit changes to the tables queried.
- **Oracle Flashback Transaction Query** lets you view changes made by a single transaction, or by all the transactions during a period of time.
- **Oracle Flashback Table** returns a table to its state at a previous point in time. You can restore table data while the database is online, undoing changes only to the specified table.
- **Oracle Flashback Drop** reverses the effects of a `DROP TABLE` statement.

Flashback Table, Flashback Query, Flashback Transaction Query and Flashback Version Query all rely on **undo data**, records of the effects of each update to an Oracle database

and values overwritten in the update. Used primarily for such purposes as providing read consistency for SQL queries and rolling back transactions, these undo records contain the information required to reconstruct data as it stood at a past time and examine the record of changes since that past time.

Flashback Drop is built around a mechanism called the **Recycle Bin**, which Oracle uses to manage dropped database objects until the space they occupied is needed to store new data.

Oracle Flashback Database provides a more efficient direct alternative to database point-in-time recovery. It is unlike the other flashback features in that it operates at a physical level. When you use Flashback Database, your current datafiles revert to their contents at a past time. The end product is much like the result of a database point-in-time recovery, but can be much faster because it does not require you to restore datafiles from backup, and requires only limited application of redo compared to media recovery.

Flashback Database uses **flashback logs** to access past versions of data blocks, as well as some information from the archived redo log. To have the option of using Flashback Database to repair your database, you must either configure your database to generate flashback logs, or use the related capability of guaranteed restore points to protect the contents of your database at a fixed point in time, such as immediately before a risky database change.

See Also:

- *Oracle Database Concepts* and *Oracle Database Administrator's Guide* for more information on undo data and automatic undo management
- *Oracle Database Application Developer's Guide - Fundamentals* for more information on Flashback Query, Flashback Transaction Query and Flashback Version Query
- "[Data Protection with Restore Points and Flashback Database](#)" for more information on setting up your database to use Flashback Database, and on the related Restore Points feature

Oracle Flashback Query: Recovering at the Row Level

In a data recovery context, it is useful to be able to query the state of a table at a previous time. If, for instance, you discover that at 12:30 PM, an employee 'JOHN' had been deleted from your EMP table, and you know that at 9:30AM that employee's data was correctly stored in the database, you could query the contents of the table as of a time before the deletion to find out what data had been lost, and, if appropriate, re-insert the lost data in the database.

Querying the past state of the table is achieved using the **AS OF** clause of the **SELECT** statement. For example, the following query retrieves the state of the employee record for 'JOHN' at 9:30AM, April 4, 2005:

```
SELECT * FROM EMP AS OF TIMESTAMP
      TO_TIMESTAMP('2005-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
WHERE name = 'JOHN';
```

Restoring John's information to the table EMP requires the following update:

```
INSERT INTO EMP
      (SELECT * FROM EMP AS OF TIMESTAMP
      TO_TIMESTAMP('2005-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS'))
```

```
WHERE name = 'JOHN');
```

The missing row is re-created with its previous contents, with minimal impact to the running database.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for a more extensive discussion of the use of the SELECT... AS OF SQL statement and extensive examples of its use.
- *Oracle Database SQL Reference* for more details on the syntax of the SELECT... AS OF form of the SELECT statement.

Oracle Flashback Table: Returning Individual Tables to Past States

Oracle Flashback Table provides the DBA the ability to recover a table or set of tables to a specified point in time in the past very quickly, easily, and without taking any part of the database offline. In many cases, Flashback Table eliminates the need to perform more complicated point-in-time recovery operations. Flashback Table restores tables while automatically maintaining associated attributes such as current indexes, triggers and constraints, and not requiring the DBA to find and restore application-specific properties. Using Flashback Table causes the contents of one or more individual tables to revert to their state at some past SCN or time.

Flashback Table uses information in the undo tablespace to restore the table. You do not have to restore any data from backups, and the rest of your database remains available while the Flashback Table operation is being performed.

For more information on Automatic Undo Management, see *Oracle Database Administrator's Guide*.

To use the Flashback Table feature on one or more tables, use the FLASHBACK TABLE SQL statement with a target time or SCN.

Prerequisites for Using Flashback Table

The prerequisites for using the Flashback Table feature on a table are as follows:

- Row movement must be enabled on the table. You can enable row movement with the following SQL statement:

```
ALTER TABLE table ENABLE ROW MOVEMENT;
```
- You must have been granted the FLASHBACK ANY TABLE system privilege or you must have the FLASHBACK object privilege on the table.
- You must have SELECT, INSERT, DELETE, and ALTER privileges on the table.
- Undo information retained in the undo tablespace must go far enough back in time to satisfy the specified target point in time or SCN for the FLASHBACK TABLE operation.

Note: FLASHBACK TABLE... TO BEFORE DROP is a use of the Flashback Drop feature, not Flashback Table, and therefore is not subject to these prerequisites. See "[Oracle Flashback Drop: Undo a DROP TABLE Operation](#)" "[Oracle Flashback Drop: Undo a DROP TABLE Operation](#)" for more information.

Performing Flashback Table

The following SQL*Plus statement performs a `FLASHBACK TABLE` operation on the table `EMP`:

The `EMP` table is restored to its state when the database was at the time specified by the `SCN`.

```
FLASHBACK TABLE EMP TO SCN 123456;
```

You can also specify the target point in time for the `FLASHBACK TABLE` operation using `TO_TIMESTAMP`. For example:

```
FLASHBACK TABLE EMP TO TIMESTAMP
  TO_TIMESTAMP('2005-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
```

Note: The mapping of timestamps to SCNs is not always exact. When using timestamps with the `FLASHBACK TABLE` statement, the actual point in time to which the table is flashed back can vary by up to approximately three seconds of the time specified for `TO_TIMESTAMP`. If an exact point in time is required, use an `SCN` rather than a time expression.

By default, the database disables triggers on the affected table before performing a `FLASHBACK TABLE` operation, and after the operation returns them to the state they were in before the operation (enabled or disabled). If you wish for triggers on a table to apply during `FLASHBACK TABLE`, then add an `ENABLE TRIGGERS` clause to the `FLASHBACK TABLE` statement:

```
FLASHBACK TABLE table_name
  TO TIMESTAMP timestamp ENABLE TRIGGERS;
```

The following scenario is typical of the kind of logical corruption where Flashback Table could be used:

At 17:00 an HR administrator discovers that an employee `JOHN` is missing from the `EMP` table. This employee was included in the table at 14:00, the last time the report was run. Therefore, someone accidentally deleted the record for `JOHN` between 14:00 and the present time. She uses Flashback Table to return the table to its state at 14:00, respecting any triggers set on the `EMP` table, using the SQL statement shown in this example:

```
FLASHBACK TABLE EMP
  TO TIMESTAMP TO_TIMESTAMP('2005-03-03 14:00:00') ENABLE TRIGGERS;
```

See Also: *Oracle Database SQL Reference* for a simple Flashback Table scenario

Oracle Flashback Drop: Undo a DROP TABLE Operation

Oracle Flashback Drop reverses the effects of a `DROP TABLE` operation. It can be used to recover after the accidental drop of a table. Flashback Drop is substantially faster than other recovery mechanisms that can be used in this situation, such as point-in-time recovery, and does not lead to any loss of recent transactions or downtime.

When you drop a table, the database does not immediately remove the space associated with the table. Instead, the table is renamed and, along with any associated objects, it is placed in the **Recycle Bin** of the database. The Flashback Drop operation recovers the table from the recycle bin.

To understand how to use Oracle Flashback Drop, you must also understand how the recycle bin works, and how to access and manage its contents.

This section covers the following topics:

- [What is the Recycle Bin?](#)
- [How Tables and Other Objects Are Placed in the Recycle Bin](#)
- [Naming Convention for Objects in the Recycle Bin](#)
- [Viewing and Querying Objects in the Recycle Bin](#)
- [Recycle Bin Capacity and Space Pressure](#)
- [Purging Objects from the Recycle Bin](#)

What is the Recycle Bin?

The recycle bin is a logical container for all dropped tables and their dependent objects. When a table is dropped, the database will store the table, along with its dependent objects in the recycle bin so that they can be recovered later. Dependent objects which are stored in the recycle bin include indexes, constraints, triggers, nested tables, LOB segments and LOB index segments.

How Tables and Other Objects Are Placed in the Recycle Bin

Tables are placed in the recycle bin along with their dependent objects whenever a DROP TABLE statement is executed. For example, this statement places the EMPLOYEE_DEMO table, along with any indexes, constraints, or other dependent objects listed previously, in the recycle bin:

```
SQL> DROP TABLE EMPLOYEE_DEMO;  
Table Dropped
```

The table and its dependent objects will remain in the recycle bin until they are **purged** from the recycle bin. You can explicitly purge a table or other object from the recycle bin with the SQL*Plus PURGE statement, as described in "[Purging Objects from the Recycle Bin](#)" on page 7-11. If you are sure that you will not want to recover a table later, you can drop it immediately and permanently, instead of placing it in the recycle bin, by using the PURGE option of the DROP TABLE statement, as shown in this example:

```
DROP TABLE employee_demo PURGE;
```

Even if you do not purge objects from the recycle bin, the database purges objects from the recycle bin to meet tablespace space constraints. See "[Recycle Bin Capacity and Space Pressure](#)" on page 7-9 for more details.

Recycle bin objects are not counted as used space. If you query the space views to obtain the amount of free space in the database, objects in the recycle bin are counted as free space.

Dropped objects still appear in the views USER_TABLES, ALL_TABLES, DBA_TABLES, USER_INDEX, ALL_INDEX and DBA_INDEX. A new column, DROPPED, is set to YES for these objects. You can use the DROPPED column in queries against these views to view only objects that are not dropped.

To view only objects in the recycle bin, use the `USER_RECYCLEBIN` and `DBA_RECYCLEBIN` views, described later in this chapter.

Naming Convention for Objects in the Recycle Bin

When a table and its dependent objects are moved to the recycle bin, they are assigned unique names, to avoid name conflicts that may arise in the following circumstances:

- A user drops a table, creates another with the same name, then drops the second table.
- Two users have tables with the same name, and both users drop their tables.

The assigned names are globally unique and are used to identify the objects while they are in the recycle bin. Object names are formed as follows:

```
BIN$$globalUID$version
```

where:

- *globalUID* is a globally unique, 24 character long identifier generated for the object.
- *version* is a version number assigned by the database

The recycle bin name of an object is always 30 characters long.

Note that the *globalUID* used in the recycle bin name is not readily correlated with any externally visible piece of information about the object or the database.

Enabling and Disabling the Recycle Bin

The recycle bin is enabled by default. The initialization parameter `RECYCLEBIN` can be used to explicitly enable or disable the recycle bin.

To enable the recycle bin, set the value of `RECYCLEBIN` to `ON`. To disable the recycle bin, set the value of `RECYCLEBIN` to `OFF`.

If you use a parameter file (PFILE) with your database, you can specify the value of `RECYCLEBIN` in the parameter file, as in this example:

```
# turn off recycle bin
RECYCLEBIN=OFF
```

This value applies to all database sessions.

To specify recycle bin behavior for your own database session, you can use an `ALTER SESSION` statement in SQL*Plus to change the value of the `RECYCLEBIN` parameter for your sessions. For example, this command disables the recycle bin for your database session:

```
ALTER SESSION SET RECYCLEBIN=OFF;
```

Objects you drop during this session are no longer placed in the recycle bin, until you re-enable the recycle bin using `ALTER SESSION SET RECYCLEBIN=ON`. However, other users continue to be protected by the recycle bin, and in future sessions the value reverts to the current default for the entire database.

You can also use an `ALTER SYSTEM` statement in SQL*Plus to change the value of the `RECYCLEBIN` parameter for the entire database. For example:

```
ALTER SYSTEM SET RECYCLEBIN=OFF;
```

This disables the recycle bin for all sessions, unless a user specifically enables the recycle bin for their session using `ALTER SESSION SET RECYCLEBIN=ON`.

Note: Objects already in the recycle bin are not affected by enabling or disabling the recycle bin using `ALTER SYSTEM` or `ALTER SESSION`.

Viewing and Querying Objects in the Recycle Bin

To view the contents of the recycle bin, use the SQL*Plus command `SHOW RECYCLEBIN`.

```
SQL> show recyclebin;
ORIGINAL_NAME      RECYCLEBIN NAME          TYPE          DROP TIME
-----
EMPLOYEE_DEMO     BIN$gk31sj/3akk5hg3j21k15j3d==$0  TABLE       2005-04-11:17:08:54
```

The `ORIGINAL_NAME` column shows the original name of the object, while the `RECYCLEBIN_NAME` column shows the name of the object as it exists in the recycle bin. Use the `RECYCLEBIN_NAME` when issuing queries against tables in the recycle bin.

The database also provides two views for obtaining information about objects in the recycle bin:

View	Description
<code>USER_RECYCLEBIN</code>	Lets users see their own dropped objects in the recycle bin. It has a synonym <code>RECYCLEBIN</code> , for ease of use.
<code>DBA_RECYCLEBIN</code>	Lets administrators see all dropped objects in the recycle bin

This example uses the views to determine the original names of dropped objects:

```
SQL> SELECT object_name as recycle_name, original_name, type
       FROM recyclebin;

RECYCLE_NAME          ORIGINAL_NAME          TYPE
-----
BIN$gk31sj/3akk5hg3j21k15j3d==$0  EMPLOYEE_DEMO        TABLE
BIN$JKS983293M1dsab4gsz/I249==$0   I_EMP_DEMO           INDEX
BIN$NR72JUN38KM1dsam4gI348as==$0   LOB_EMP_DEMO         LOB
BIN$JKJ399SLKnas1kJSLK330SIK==$0   LOB_I_EMP_DEMO       LOB INDEX
```

You can query objects that are in the recycle bin, just as you can query other objects, if these three conditions are met:

- You must have the `FLASHBACK` privilege.
- You must have the privileges that were required to perform queries against the object before it was placed in the recycle bin.
- You must use the recycle bin name of the object in your query, rather than the object's original name.

This example shows the required syntax:

```
SQL> SELECT * FROM "BIN$KSD8DB9L345KLA==$0";
```

(Note the use of quotes due to the special characters in the recycle bin name.)

You can also use Oracle Flashback Query on tables in the recycle bin (again, assuming that you have the privileges described previously).

Recycle Bin Capacity and Space Pressure

There is no fixed amount of space preallocated for the recycle bin. Therefore, there is no guaranteed minimum amount of time during which a dropped object will remain in the recycle bin.

The rules that govern how long an object is retained in the recycle bin and how and when space is reclaimed are explained in this section.

Understanding Space Pressure

Dropped objects are kept in the recycle bin until such time as no new extents can be allocated in the tablespace to which the objects belong without growing the tablespace. This condition is referred to as **space pressure**. Space pressure can also arise due to user quotas defined for a particular tablespace. A tablespace may have free space, but the user may have exhausted his or her quota on it.

Oracle never automatically reclaims space or overwrites objects in the recycle bin unless forced to do so in response to space pressure.

How the Database Responds to Space Pressure

When space pressure arises, the database selects objects for automatic purging from the recycle bin. Objects are selected for purging on a first-in, first-out basis, that is, the first objects dropped are the first selected for purging.

Actual purging of objects is done only as needed to meet ongoing space pressure, that is, the database purges the minimum possible number of objects selected for purging to meet immediate needs for space. This policy serves two purposes:

- It minimizes the performance penalty on transactions that encounter space pressure, by not performing more purge operations than are required;
- It maximizes the length of time objects remain in the recycle bin, by leaving them there until space is needed.

Dependent objects such as indexes on a table are selected for purging before the associated table (or other required segment).

If space pressure is due to an individual user's quota on a tablespace being exhausted, the recycle bin purges objects belonging to the tablespace which count against that user's space quotas.

For AUTO EXTEND-able tablespaces, objects are purged from the recycle bin to reclaim space before datafiles are extended.

Recycle Bin Objects and Segments

The recycle bin operates at the object level, in terms of tables, indexes, and so on. An object may have multiple segments associated with it, such as partitioned tables, partitioned indexes, lob segments, nested tables, and so on. Because the database reclaims only the segments needed to immediately satisfy space pressure, it can happen that some but not all segments of an object are reclaimed. When this happens, any segments of the object not reclaimed immediately are marked as temporary segments. These temporary segments are the first candidates to be reclaimed the next time space pressure arises.

In such a case, the partially-reclaimed object can no longer be removed from the recycle bin with Flashback Drop. (For example, if one partition of a partitioned table is reclaimed, the table can no longer be the object of a Flashback Drop.)

Performing Flashback Drop on Tables in the Recycle Bin

Use the `FLASHBACK TABLE ... TO BEFORE DROP` statement to recover objects from the recycle bin. You can specify either the name of the table in the recycle bin or the original table name. This can be obtained from either the `DBA_RECYCLEBIN` or `USER_RECYCLEBIN` view as shown in ["Viewing and Querying Objects in the Recycle Bin"](#) on page 7-8. To use the `FLASHBACK TABLE ... TO BEFORE DROP` statement, you need the same privileges you need to drop the table.

The following example restores the `BIN$gk31sj/3akk5hg3j21k15j3d==$0` table, changes its name back to `hr.int_admin_emp`, and purges its entry from the recycle bin:

```
FLASHBACK TABLE "BIN$gk31sj/3akk5hg3j21k15j3d==$0" TO BEFORE DROP;
```

Note the use of quotes, due to the possibility of special characters appearing in the recycle bin object names. You can also use the table's original name in the Flashback Drop operation:

```
FLASHBACK TABLE HR.INT_ADMIN_EMP TO BEFORE DROP;
```

You can assign a new name to the restored table by specifying the `RENAME TO` clause. For example:

```
FLASHBACK TABLE "BIN$KSD8DB9L345KLA==$0" TO BEFORE DROP
      RENAME TO hr.int2_admin_emp;
```

Flashback Drop of Multiple Objects With the Same Original Name

You can create, and then drop, several objects with the same original name, and they will all be stored in the recycle bin. For example, consider these SQL statements:

```
CREATE TABLE EMP ( ...columns ); # EMP version 1
DROP TABLE EMP;
CREATE TABLE EMP ( ...columns ); # EMP version 2
DROP TABLE EMP;
CREATE TABLE EMP ( ...columns ); # EMP version 3
DROP TABLE EMP;
```

In such a case, each table `EMP` is assigned a unique name in the recycle bin when it is dropped. You can use a `FLASHBACK TABLE ... TO BEFORE DROP` statement with the original name of the table, as shown in this example:

```
FLASHBACK TABLE EMP TO BEFORE DROP;
```

The most recently dropped table with that original name is retrieved from the recycle bin, with its original name. You can retrieve it and assign it a new name using a `RENAME TO` clause. The following example shows the retrieval from the recycle bin of all three dropped `EMP` tables from the previous example, with each assigned a new name:

```
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_3;
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_2;
FLASHBACK TABLE EMP TO BEFORE DROP RENAME TO EMP_VERSION_1;
```

Because using the original name in `FLASHBACK TABLE... TO BEFORE DROP` refers to the most recently dropped table with that name, the last table dropped is the first one to be retrieved.

You can also retrieve any table you want from the recycle bin, regardless of any collisions among original names, by using the table's unique recycle bin name.

Purging Objects from the Recycle Bin

The PURGE command is used to permanently purge objects from the recycle bin. Once purged, objects can no longer be retrieved from the bin using Flashback Drop.

There are a number of forms of the PURGE statement, depending on exactly which objects you want to purge from the recycle bin.

See Also: *Oracle Database SQL Reference* for more information on the PURGE statement

PURGE TABLE: Purging a Table and Dependent Objects

The PURGE TABLE command purges an individual table and all of its dependent objects from the recycle bin. This example shows the syntax, using the table's original name:

```
PURGE TABLE EMP;
```

You can also use the recycle bin name of an object with PURGE TABLE:

```
PURGE TABLE "BIN$KSD8DB9L345KLA==$0";
```

If you have created and dropped multiple tables with the same original name, then when you use the PURGE TABLE statement the first table dropped will be the one to be purged.

Note: The behavior in this case is the opposite of the behavior of FLASHBACK TABLE... TO BEFORE DROP, where using the original name of the table retrieves the most recently dropped version from the recycle bin.

For example, consider the following series of CREATE TABLE and DROP TABLE statements:

```
CREATE TABLE EMP; # version 1 of the table
DROP TABLE EMP; # version 1 dropped
CREATE TABLE EMP; # version 2 of the table
DROP TABLE EMP; # version 2 dropped
CREATE TABLE EMP; # version 3 of the table
DROP TABLE EMP; # version 3 dropped
```

There are now three EMP tables in the recycle bin. If you execute PURGE TABLE EMP several times, the effect is as described here:

```
PURGE TABLE EMP; # version 1 of the table is purged
PURGE TABLE EMP; # version 2 of the table is purged
PURGE TABLE EMP; # version 3 of the table is purged
```

PURGE INDEX: Freeing Space in the Recycle Bin

You can use PURGE INDEX to purge just an index for a table, while keeping the base table in the recycle bin. The syntax for purging an index is as follows:

```
PURGE INDEX "BIN$GTE72KJ22H9==$0";
```

By purging indexes from the recycle bin, you can reduce the chance of space pressure, so that dropped tables can remain in the recycle bin longer. If you retrieve a table from

the recycle bin using Flashback Drop, you can rebuild the indexes after you retrieve the table.

PURGE TABLESPACE: Purging All Dropped Objects from a Tablespace

You can use the `PURGE TABLESPACE` command to purge all dropped tables and other dependent objects from a specific tablespace. The syntax is as follows:

```
PURGE TABLESPACE hr;
```

You can also purge only objects from a tablespace belonging to a specific user, using the following form of the command:

```
PURGE TABLESPACE hr USER scott;
```

PURGE RECYCLEBIN: Purging All Objects in a User's Recycle Bin

The `PURGE RECYCLEBIN` command purges the contents of the recycle bin for the currently logged-in user.

```
PURGE RECYCLEBIN;
```

It purges all tables and their dependent objects for this user, along with any other indexes owned by this user but not on tables owned by the user.

PURGE DBA_RECYCLEBIN: Purging All Recycle Bin Objects

If you have the `SYSDBA` privilege, then you can purge all objects from the recycle bin, regardless of which user owns the objects, using this command:

```
PURGE DBA_RECYCLEBIN;
```

Dropping a Tablespace, Cluster, User or Type and the Recycle Bin

When a tablespace is dropped including its contents, the objects in the tablespace are dropped immediately, and not placed in the recycle bin. Any objects in the recycle bin from the dropped tablespace are purged from the recycle bin.

If all objects from a tablespace have been placed in the recycle bin, then dropping the tablespace causes the objects to be purged, even if you do not use the `INCLUDING CONTENTS` clause with `DROP TABLESPACE`.

When a user is dropped, any objects belonging to the user that are not in the recycle bin are dropped immediately, not placed in the recycle bin. Any objects in the recycle bin that belonged to the user are purged from the recycle bin.

When you drop a cluster, all tables in the cluster are purged. When you drop a user-defined data type, all objects directly or indirectly dependent upon that type are purged.

Privileges and Security for Flashback Drop

This section summarizes the system privileges required for the operations related to Flashback Drop and the recycle bin.

- `DROP`
Any user with drop privileges over the object can drop the object, placing it in the recycle bin.
- `FLASHBACK TABLE... TO BEFORE DROP`

Privileges are tied to the privileges for DROP. That is, any user who can drop an object can perform Flashback Drop to retrieve the dropped object from the recycle bin.

- PURGE

Privileges are tied to the DROP privileges. Any user having DROP TABLE or DROP ANY TABLE privileges can purge the objects from the recycle bin.

- SELECT for objects in the Recycle Bin

Users must have SELECT and FLASHBACK privileges over an object in the recycle bin to be able to query the object in the recycle bin. Any users who had the SELECT privilege over an object before it was dropped continue to have the SELECT privilege over the object in the recycle bin. Users must have FLASHBACK privilege to query any object in the recycle bin, because these are objects from a past state of the database.

Limitations and Restrictions on Flashback Drop

- The recycle bin functionality is only available for non-system, locally managed tablespaces. If a table is in a non-system, locally managed tablespace, but one or more of its dependent segments (objects) is in a dictionary-managed tablespace, then these objects are protected by the recycle bin.
- There is no fixed amount of space allocated to the recycle bin, and no guarantee as to how long dropped objects remain in the recycle bin. Depending upon system activity, a dropped object may remain in the recycle bin for seconds, or for months.
- While Oracle permits queries against objects stored in the recycle bin, you cannot use DML or DDL statements on objects in the recycle bin.
- You can perform Flashback Query on tables in the recycle bin, but only by using the recycle bin name. You cannot use the original name of the table.
- A table and all of its dependent objects (indexes, LOB segments, nested tables, triggers, constraints and so on) go into the recycle bin together, when you drop the table. Likewise, when you perform Flashback Drop, the objects are generally all retrieved together.

It is possible, however, that some dependent objects such as indexes may have been reclaimed due to space pressure. In such cases, the reclaimed dependent objects are not retrieved from the recycle bin.

- Due to security concerns, tables which have Fine-Grained Auditing (FGA) and Virtual Private Database (VPD) policies defined over them are not protected by the recycle bin.
- Partitioned index-organized tables are not protected by the recycle bin.
- The recycle bin does not preserve referential constraints on a table (though other constraints will be preserved if possible). If a table had referential constraints before it was dropped (that is, placed in the recycle bin), then re-create any referential constraints after you retrieve the table from the recycle bin with Flashback Drop.

Reversing Database Changes with Flashback Database

If your database was previously configured for flashback logging as described in "[Setup and Maintenance for Oracle Flashback Database](#)" on page 5-8, then you can use the FLASHBACK DATABASE command to return your database contents to points in

time within the flashback window. You can also use `FLASHBACK DATABASE` to return to any guaranteed restore point you previously defined using the commands in ["Creating Normal and Guaranteed Restore Points"](#) on page 5-7.

This section explains the common scenarios for using Flashback Database to reverse unwanted changes to your database. It contains the following topics:

- [Performing Flashback Database: Scenario](#)
- [Performing Flashback Database to a Guaranteed Restore Point](#)
- [Performing Flashback Database to Undo an OPEN RESETLOGS](#)

Note: If you do not have a guaranteed restore point defined, verify that the target SCN is within the flashback window, the range of SCNs for which you can use `FLASHBACK DATABASE`. See ["Determining the Current Window for Flashback Database"](#) on page 5-11 for details on determining the flashback window, and ["Listing Restore Points"](#) on page 5-7 for details on determining whether there is a guaranteed restore point you can use in recovery.

If the flashback window does not extend far enough back into the past to reach the desired target time and you do not have a guaranteed restore point at the desired time, you can achieve similar results by using database point-in-time recovery, as described in ["Performing Database Point-In-Time Recovery"](#) on page 7-19.

Performing Flashback Database: Scenario

This section presents a basic outline of the process for performing Flashback Database in almost all cases, specifying the desired target point in time using a time expression, the name of a normal or guaranteed restore point, or an SCN.

To perform a Flashback Database operation in RMAN, use the following steps:

1. Determine the desired SCN, restore point or point in time for the `FLASHBACK DATABASE` command.

Note:

- If you plan to specify the target using an SCN, and you want to return the database to a state with changes that were abandoned using a previous point-in-time recovery or Flashback Database operation, then see ["Flashback Database and Ambiguous SCNs Across Incarnations"](#) on page 7-16 for special considerations.
 - You can use the method in ["Performing Flashback Database to a Guaranteed Restore Point"](#) on page 7-17 if your goal is to flash back to a guaranteed restore point.
 - If the goal is to return the database to the point in time immediately before the most recent `OPEN RESETLOGS` operation, use the steps in ["Performing Flashback Database to Undo an OPEN RESETLOGS"](#) on page 7-17.
-
-

2. Start RMAN and connect to the target database. For example:

```
rman TARGET /
```

3. Shut down the database cleanly, and ensure that it is not opened by any instance. Then mount it:

```

RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;

```

4. Repeat the query from "[Determining the Current Window for Flashback Database](#)" on page 5-11. Some flashback logging data is generated when the database is shut down. It is possible that your target SCN is no longer reachable, if some flashback database logs were deleted due to space pressure in the flash recovery area in response to that logging.

Note: If you attempt `FLASHBACK DATABASE` and your target SCN is now outside the flashback window, then `FLASHBACK DATABASE` will fail with an ORA-38729 error. In such a case, however, your database will not be changed.

5. During Flashback Database, RMAN may need to restore some archived redo logs from backup. If the backups are on tape and you have not configured the necessary channels to access the SBT device, then use a `RUN` block around the `FLASHBACK DATABASE` command and issue `ALLOCATE CHANNEL` commands as needed to allow RMAN to retrieve these logs from disk or tape.
6. Run the RMAN `FLASHBACK DATABASE` command, specifying the target time using one of the forms of the command shown in the following examples:

```
RMAN> FLASHBACK DATABASE TO SCN 46963;
```

```
RMAN> FLASHBACK DATABASE
      TO RESTORE POINT BEFORE_CHANGES;
```

```
RMAN> FLASHBACK DATABASE TO TIME
      "TO_DATE('09/20/00','MM/DD/YY')";
```

When the `FLASHBACK DATABASE` command completes, the database is left mounted and recovered to the specified point in time.

You can verify that you have returned the database to the desired state, by opening the database read-only and performing some queries to inspect the database contents.

```
RMAN> SQL 'ALTER DATABASE OPEN READ ONLY';
```

Options After a Successful Flashback Database Operation

If you are satisfied with the state of the database after the Flashback Database operation, you have two choices:

- Make the database available for updates by performing an `OPEN RESETLOGS` operation:

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

Note: Once you perform this `OPEN RESETLOGS` operation, all changes to the database after the target SCN for `FLASHBACK DATABASE` are abandoned. However, you can use the method in ["Flashback Database To The Right of Open Resetlogs: Example"](#) on page 7-18 to return the database to that range of SCNs while they remain in the flashback window.

- Use Oracle export utilities (Original Export or Data Pump Export) to export the objects whose state was corrupted. Then, recover the database to the present time:

```
RMAN> RECOVER DATABASE;
```

This step undoes the effect of the Flashback Database, by re-applying all changes in the redo logs to the database, returning it to the most recent SCN.

After re-opening the database read-write, you can import the exported objects using the import utility corresponding to the export utility used earlier (Original Import or Data Pump Import).

Options After Flashback Database to the Wrong Time

If, after investigating the state of your database, you find that you used the wrong restore point, time or SCN for Flashback Database, then you have several options:

- If your chosen target time was not far enough in the past, then you can use another `FLASHBACK DATABASE` command to rewind the database further in time.

```
RMAN> FLASHBACK DATABASE TO SCN 42963; #earlier than current SCN
```

- If you chose a target SCN that is too far in the past, then you can mount the database and use `RECOVER DATABASE UNTIL` to wind the database forward in time to the desired SCN:

```
RMAN> RECOVER DATABASE UNTIL SCN 56963; #later than current SCN
```

- If you want to completely undo the effect of the `FLASHBACK DATABASE` command, you can perform complete recovery of the database by using the `RECOVER DATABASE` command without an `UNTIL` clause or `SET UNTIL` command:

```
RMAN> RECOVER DATABASE;
```

This re-applies all changes to the database, returning it to the most recent SCN.

Flashback Database and Ambiguous SCNs Across Incarnations

In databases that have previously undergone point-in-time recovery or Flashback Database, an SCN can be an ambiguous method of designating a point in time.

The effect of Flashback Database or DBPITR followed by an `OPEN RESETLOGS` is to return the database to a previous SCN, and to abandon changes after that point. Therefore, some SCNs after that point can refer either to changes that were abandoned or changes in the current history of the database.

By default, an SCN used as an argument for the `FLASHBACK DATABASE` command is assumed to refer to a point in time in the current incarnation path, that is, to an incarnation that was not abandoned after some previous DBPITR or Flashback Database. If this is the goal, then you can use the procedure in this section with an SCN to specify the target for Flashback Database.

In cases when you want to use Flashback Database to reach a point in time corresponding to changes that were abandoned after a previous Flashback Database or DBPITR, you can use the method described in ["Flashback Database To The Right of Open Resetlogs: Example"](#) on page 7-18.

See Also: ["Point-in-Time Recovery and Database Incarnations: Concepts"](#) on page 7-19 for useful background information about database incarnations, abandoned changes, and the effects of OPEN RESETLOGS.

Unlike SCNs, time expressions and restore points are not ambiguous. A time expression is always associated with the incarnation that was current at that time. A restore point is always associated with the current incarnation when it was created. This is true even for times and restore points that correspond to abandoned database incarnations. The database incarnation is automatically reset to the incarnation that was current at the specified time or when the restore point was created.

Performing Flashback Database to a Guaranteed Restore Point

You can list the available guaranteed restore points using the V\$RESTORE_POINT view, as follows:

```
SQL> SELECT NAME, SCN, TIME, DATABASE_INCARNATION#,
           GUARANTEE_FLASHBACK_DATABASE
        FROM V$RESTORE_POINT
        WHERE GUARANTEE_FLASHBACK_DATABASE='YES';
```

NAME	SCN	TIME	DATABASE_INCARNATION#	GUA
BEFORE_CHANGES	5753126	04-MAR-05 12.39.45 AM	2	YES

Having identified the restore point to use, mount the database and run the FLASHBACK DATABASE command, using the restore point. For example:

```
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;
RMAN> FLASHBACK DATABASE TO RESTORE POINT 'BEFORE_CHANGES';
```

When the command completes, you may open the database read-only and inspect the effects of the operation, and if satisfied, open the database with the RESETLOGS option.

Performing Flashback Database to Undo an OPEN RESETLOGS

The basic procedure for using Flashback Database to reverse an unwanted OPEN RESETLOGS is very similar to the general case described in ["Performing Flashback Database: Scenario"](#) on page 7-14.

However, rather than specifying a particular SCN or point in time for the FLASHBACK DATABASE command, use FLASHBACK DATABASE TO BEFORE RESETLOGS, as in the following example.

Before performing the flashback, verify that the beginning of the flashback window is earlier than the time of the most recent OPEN RESETLOGS.

```
sql> select resetlogs_change# from v$database;
sql> select oldest_flashback_scn from v$flashback_database_log;
```

If `V$DATABASE.RESETLOGS_CHANGE#` is greater than `V$FLASHBACK_DATABASE_LOG.OLDEST_FLASHBACK_SCN`, then you can use Flashback Database to reverse the `OPEN RESETLOGS`.

Shut down the database, mount it, and re-check the flashback window. If the resetlogs SCN is still within the flashback window, then use this form of the `FLASHBACK DATABASE` command:

```
RMAN> FLASHBACK DATABASE TO BEFORE RESETLOGS;
```

As with other uses of `FLASHBACK DATABASE`, if the target SCN is prior to the beginning of the flashback database window, an error is returned and the database is not modified. If the command completes successfully, then the database is left mounted and recovered to the last SCN before the `OPEN RESETLOGS` in the previous incarnation.

You can open the database read-only and perform queries to make sure the data is in the desired state. To make the database available for updates again, use `ALTER DATABASE OPEN RESETLOGS` command.

Flashback Database Across OPEN RESETLOGS With Standby Databases

Support for Flashback Database across `OPEN RESETLOGS` enables several applications of Flashback Database with standby databases. These include:

- **Flashback to undo logical standby switchovers**, in which the database reverts to its role (primary or standby) at the target time for the Flashback Database operation
- **Undo of a physical standby activation**, so that you can temporarily activate a physical standby database, use it for testing or reporting purposes, then use Flashback Database to return it to its role as a physical standby
- **Ongoing use of a clone or standby database for testing**, without requiring the use of storage snapshots.

See *Oracle Data Guard Concepts and Administration* for details on these advanced applications of Flashback Database with Data Guard.

Flashback Database To The Right of Open Resetlogs: Example

In some cases, you may need to return the database to a point in time in the parent incarnation, later than the SCN of the `OPEN RESETLOGS` at which the current incarnation path branched from the old incarnation. These points, which correspond to abandoned changes in the parent incarnation, can be described as being "to the right" of the last `OPEN RESETLOGS`, with reference to an incarnation diagram such as [Figure 7-1, "Database Incarnation History With Multiple Resetlogs"](#) on page 7-20. For example, in the diagram, the database might be in incarnation 3, and you might need to return to the abandoned SCN 1500 in incarnation 1.

You can use the `RMAN RESET DATABASE TO INCARNATION` command to specify the current incarnation referred to by the SCN to use with Flashback Database.

The process is as follows:

- Verify that the flashback logs contain enough information to flash back to that SCN:

```
sql> select oldest_flashback_scn from v$flashback_database_log;
```
- Determine the target incarnation number for the flashback, that is, the incarnation key for the parent incarnation:

```
SQL> select prior_incarnation# from v$database_incarnation where status =
'CURRENT';
```

- In RMAN, shut down the database, then mount it:

```
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;
```

- Set the database incarnation to the parent incarnation:

```
RMAN> RESET DATABASE TO INCARNATION 1;
```

- Run the FLASHBACK DATABASE command:

```
RMAN> FLASHBACK DATABASE TO SCN 1500;
```

Once the flashback is complete, you can verify the results, and if successful, open the database with `RESETLOGS`.

Performing Database Point-In-Time Recovery

Database point-in-time recovery (DBPITR) restores the database from backups prior to the target time for recovery, then uses incremental backups and redo to roll the database forward to the target time.

DBPITR is sometimes called **incomplete recovery** because it does not use all of the available redo or completely recover all changes to your database.

The target SCN can be specified using a date and time, in which case the operation is sometimes called **time-based recovery**.

Requirements for Database Point-in-Time Recovery

The requirements for database point-in-time recovery are as follows:

- Your database must be running in ARCHIVELOG mode.
- You must have backups of all datafiles from before the target SCN for DBPITR and archived redo logs for the period between the SCN of the backups and the target SCN.

Point-in-Time Recovery and Database Incarnations: Concepts

Understanding DBPITR requires background information on database incarnations and how RMAN treats backups from times not in the current incarnation path. In particular, there are special considerations if you are returning your database to a point in time prior to the most recent `OPEN RESETLOGS`.

This section contains the following topics:

- [Understanding Parent, Ancestor and Sibling Database Incarnations](#)
- [Incarnation History of a Database: Example](#)
- [Database Incarnations and Orphaned Backups](#)

Understanding Parent, Ancestor and Sibling Database Incarnations

A new incarnation of a database is created whenever each time the database is opened with the `RESETLOGS` option. Performing an `OPEN RESETLOGS` archives the current

online redo logs, Incarnation resets the log sequence number to 1, and then gives the online redo logs a new time stamp and SCN. It also increments the incarnation number, which is used to uniquely tag and identify a stream of redo.

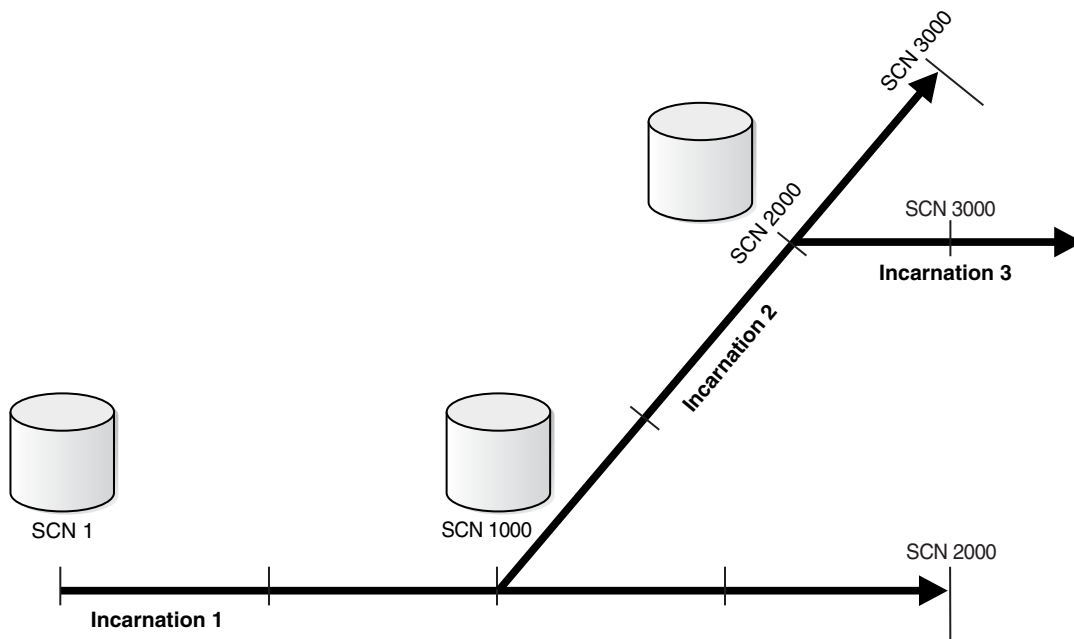
Incarnations can stand in several relations to each other:

- The incarnation from which the current incarnation branched following an OPEN RESETLOGS operation is called the **parent incarnation** of the current incarnation.
- The parent incarnation and all of its parent incarnations are the **ancestor incarnations** of the current incarnation.
- Two incarnations that share a common ancestor are **sibling incarnations** if neither one is an ancestor of the other.

Incarnation History of a Database: Example

Figure 7–1, "Database Incarnation History With Multiple Resetlogs" shows a database that goes through several incarnations.

Figure 7–1 Database Incarnation History With Multiple Resetlogs



Incarnation 1 of the database starts at SCN 1, and continues through SCN 1000 to SCN 2000. At SCN 2000 in incarnation 1, you perform a point-in-time recovery back to SCN 1000, and open the database with a RESETLOGS operation. This creates incarnation 2, which begins at SCN 1000 and continues to SCN 3000. At SCN 3000 in incarnation 2, you perform another point-in-time recovery and RESETLOGS operation. This creates incarnation 3, starting at SCN 2000.

You can view the incarnation history of a database using the LIST INCARNATION command. Output describing the incarnation history in the figure is:

```
LIST INCARNATION;
```

```
List of Database Incarnations
DB Key  Inc Key  DB Name  DB ID          STATUS  Reset SCN  Reset Time
-----
```

1	1	TRGT	930791268	PARENT	1	23-APR-05
2	2	TRGT	930791268	PARENT	1000	23-APR-05
3	3	TRGT	930791268	CURRENT	2000	23-APR-05

The value in the `Reset SCN` column is the SCN at which the `RESETLOGS` was performed. The `Inc Key` column is the incarnation key. RMAN uses the incarnation key to identify the database incarnation in some commands, such as using `RESET DATABASE TO INCARNATION` to change the current incarnation in some complex recovery scenarios.

See also: *Oracle Database Backup and Recovery Reference* for details about the `RESET DATABASE` command

Sibling Incarnations, Ambiguous SCNs and RESET DATABASE INCARNATION When working with a database where flashback or point-in-time recovery operations have produced sibling incarnations, note that a given SCN value can refer to more than one point in time, depending upon which incarnation has been set as the current incarnation. For example, in the figure, SCN 1500 could refer to a point in incarnation 1 or 2.

By default, when used with an RMAN command like `FLASHBACK DATABASE` or `RECOVER... UNTIL`, an SCN is assumed to refer to the current incarnation path, rather than sibling incarnations. However, you can use the `RESET DATABASE TO INCARNATION` command to specify that SCNs are to be interpreted in the frame of reference of another incarnation. For example, consider the following command used in point-in-time recovery:

```
RMAN> RECOVER DATABASE TO SCN 1500;
```

If used in the database described in [Figure 7-1](#), SCN 1500 refers to incarnation 2 by default. If, however, you run the following sequence of commands:

```
RMAN> RESET DATABASE INCARNATION TO 1;
RMAN> RECOVER DATABASE TO SCN 1500;
```

SCN 1500 refers to the point in time during incarnation 1 when the SCN was 1500.

Database Incarnations and Orphaned Backups

When a database goes through multiple incarnations, some backups can become **orphaned**. Orphaned backups are backups that are created during incarnations of the database that are not ancestors of the current incarnation.

Given the database from the example in "[Point-in-Time Recovery and Database Incarnations: Concepts](#)" on page 7-19, the following table explains which backups are orphans, based upon which incarnation is current.

Current Incarnation	Usable Backups (Nonorphaned)	Orphaned Backups
Incarnation 1	All backups from incarnation 1	All backups from incarnations 2 and 3
Incarnation 2	<ul style="list-style-type: none"> ■ All backups from incarnation 1 prior to SCN 1000 ■ All backups from incarnation 2 	<ul style="list-style-type: none"> ■ Backups from incarnation 1 after SCN 1000. ■ All backups from incarnation 3

Current Incarnation	Usable Backups (Nonorphaned)	Orphaned Backups
Incarnation 3	<ul style="list-style-type: none"> ■ All backups from incarnation 1 prior to SCN 1000 ■ All backups from incarnation 2 prior to SCN 2000 ■ All backups from incarnation 3 	<ul style="list-style-type: none"> ■ All backups from incarnation 1 after SCN 1000 ■ All backups from incarnation 2 after SCN 2000

Uses of Orphaned Backups Orphaned backups are usable by RMAN in cases where you wish to restore the database to a point in time not in the current incarnation path. RMAN is able to restore backups from direct ancestor incarnations and recover to the current time, even across OPEN RESETLOGS operations, as long as a continuous path of archived logs exists from the earliest backups to the point to which you want to recover. RMAN can also perform restore and recovery with orphaned backups, if you restore a control file from an incarnation in which the changes represented in the backups had not been abandoned.

Preparing for Database Point-in-Time Recovery

Take the following steps to prepare for DBPITR:

- Determine the target time, SCN, restore point, or log sequence number that should end recovery. The Flashback Query, Flashback Version Query and Flashback Transaction Query features can help you identify when the logical corruption occurred.

You can also examine the alert.log for information that may help you determine the time of the event from which you need to recover.

Alternatively, you can determine the log sequence number that contains the target SCN, and then recover through that log. For example, query V\$LOG_HISTORY to view the logs that have been archived.

RECID	STAMP	THREAD#	SEQUENCE#	FIRST_CHAN	FIRST_TIM	NEXT_CHANG
1	344890611	1	1	20037	24-SEP-04	20043
2	344890615	1	2	20043	24-SEP-04	20045
3	344890618	1	3	20045	24-SEP-04	20046

If, for example, you discover 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 after that time.

- If you are using a target time expression instead of a target SCN, then make sure that the time format environment variables are set appropriately before invoking RMAN. The following are sample Globalization Support settings:

```
NLS_LANG = american_america.us7ascii
NLS_DATE_FORMAT="Mon DD YYYY HH24:MI:SS"
```

Database Point-in-Time Recovery Within the Current Incarnation

DBPITR within the current incarnation is performed using the current control file. When performing DBPITR, you can avoid errors by using the SET UNTIL command to set the target time at the beginning of the process, rather than specifying the UNTIL clause on the RESTORE and RECOVER commands individually. This ensures that the

datafiles restored from backup will have timestamps early enough to be used in the subsequent RECOVER operation.

The steps required for DBPITR are as follows:

1. Connect RMAN to the target database and, if applicable, the recovery catalog database. Bring the database to a MOUNT state:

```
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
```

2. Perform the following operations within a RUN block:

- a. Use SET UNTIL to specify the target time, restore point, SCN, or log sequence number for DBPITR. If specifying a time, then use the date format specified in the NLS_LANG and NLS_DATE_FORMAT environment variables.
- b. If automatic channels are not configured, then manually allocate disk and tape channels as needed.
- c. Restore and recover the database.

The following example performs DBPITR on the target database until SCN 1000:

```
RUN
{
    SET UNTIL SCN 1000;
    # Alternatives:
    # SET UNTIL TIME 'Nov 15 2004 09:00:00';
    # SET UNTIL SEQUENCE 9923;
    RESTORE DATABASE;
    RECOVER DATABASE;
}
```

Note: You can also use time expressions, restore points, or log sequence numbers to specify the SET UNTIL time:

```
SET UNTIL TIME 'Nov 15 2004 09:00:00';
SET UNTIL SEQUENCE 9923;
SET UNTIL RESTORE POINT before_update;
```

If the operation completes without errors, then your DBPITR has succeeded. You can open the database read-only and perform queries as needed to ensure that the effects of the logical corruption have been reversed. If not, you may have chosen the wrong target SCN. In such a case, investigate the unwanted change further and determine a new target SCN, then repeat the DBPITR process.

Using a Time Expression for Database Point-in-Time Recovery

You can use a time expression instead of the SCN in the SET UNTIL statement, as shown in the preceding example. However, note that if you use SET UNTIL TIME to specify the target time for point-in-time recovery, some times that you can specify may not be in the current incarnation. The database may have been in an ancestor incarnation, or even in a sibling incarnation, at the target time. If your target time is not in the current incarnation, then see ["Point-in-Time Recovery to an Ancestor Incarnation"](#) on page 7-24 for more information on DBPITR to ancestor incarnations, and *Oracle Database Backup and Recovery Advanced User's Guide* for more information on DBPITR to incarnations that are not ancestors of the current incarnation.

Options After Database Point-in-Time Recovery

After a successful DBPITR, your choices are:

- Export one or more objects from your database using an Oracle export utility such as Data Pump Export. You can then recover the database to the current point in time and re-import the exported objects, as a way to return these objects to their state prior to the unwanted change without abandoning all other changes.
- Open your database for read-write, abandoning all changes after the target SCN. In such a case, you must open the database with the `RESETLOGS` option, as shown here:

```
RMAN> ALTER DATABASE OPEN RESETLOGS;
```

The current online redo logs are archived, the log sequence number is reset to 1, and the online redo logs are given a new time stamp and SCN. Identifying redo log files with a new log sequence number and incarnation eliminates the possibility of corrupting datafiles by the application of obsolete archived redo logs.

The `OPEN RESETLOGS` operation will fail if a datafile is off-line, unless the datafile went offline normally or is read-only. You can bring files in read-only or offline normal tablespaces online after the `RESETLOGS` because they do not need any redo.

Point-in-Time Recovery to an Ancestor Incarnation

The main differences between DBPITR within the current incarnation and to an SCN in an ancestor incarnation are that you must reset the incarnation of the database to the incarnation that was current at the target SCN, and you must restore a control file from the incarnation containing the target SCN.

Assume the following situation:

- You run RMAN with a recovery catalog.
- You have a backup of target database `trgt` from October 2, 2004.
- DBPITR was performed on this database on October 10, 2004 to correct an earlier error. The `OPEN RESETLOGS` operation at the end of that DBPITR started a new incarnation.

On October 25, you discover that you need crucial data that was dropped from the database at 8:00 a.m. on October 8, 2004. This time is prior to the beginning of the current incarnation.

To perform point-in-time recovery to the older incarnation, use the following steps:

1. Determine which incarnation was current at the time of the backup of 2 October. Use `LISTINCARNATION` to find the primary key of the incarnation that was current at the target time:

```
LIST INCARNATION OF DATABASE trgt;
```

```
List of Database Incarnations
```

DB Key	Inc Key	DB Name	DB ID	STATUS	Reset SCN	Reset Time
-----	-----	-----	-----	-----	-----	-----
1	2	TRGT	1224038686	PARENT	1	02-OCT-04
1	582	TRGT	1224038686	CURRENT	59727	10-OCT-04

Look at the `Reset SCN` and `Reset Time` columns to identify the correct incarnation, and note the incarnation key in the `Inc Key` column. In this case, the incarnation key value is 2.

2. Make sure the database is started but not mounted.

```
STARTUP FORCE NOMOUNT
```

3. Reset `trgt` to the incarnation that was current at the time of the backup of 2 October. Use the value from the `Inc Key` column to identify the incarnation.

```
# reset database to old incarnation
RESET DATABASE TO INCARNATION 2;
```

4. Restore and recover the database, performing the following actions in the `RUN` command:

- Set the end time for recovery to the time just before the loss of the data.
- Allocate any channels required that are not already configured.
- Restore the control file from the October 2 backup and mount it.
- Restore the datafiles and recover the database. Use the `RECOVER DATABASE ... UNTIL` command to perform point-in-time recovery, bringing the database to the target time of 7:55 a.m. on October 8, just before the data was lost.

The following example shows all of the steps required in this case:

```
RUN
{
  # set target time for all operations in the RUN block
  SET UNTIL TIME 'Oct 8 2004 07:55:00';
  RESTORE CONTROLFILE ;
  # without recovery catalog, use RESTORE CONTROLFILE FROM AUTOBACKUP
  ALTER DATABASE MOUNT;
  RESTORE DATABASE;
  RECOVER DATABASE;
}
```

```
ALTER DATABASE OPEN RESETLOGS;
```

Recovery Manager Maintenance Tasks

This chapter describes how to manage the RMAN repository and some maintenance tasks related to the flash recovery area.

This chapter contains these topics:

- [Managing the RMAN Repository Using Only the Control File](#)
- [Using CROSSCHECK to Update the RMAN Repository](#)
- [Using Multiple RMAN Channels for Maintenance Operations](#)
- [Changing the Status of a Backup Record](#)
- [Cataloging Archived Logs and User-Managed Copies](#)
- [Uncataloging RMAN Records](#)
- [Flash Recovery Area Maintenance](#)

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for details on managing the RMAN repository when using a recovery catalog

Managing the RMAN Repository Using Only the Control File

The authoritative RMAN repository is always stored in the database control file. RMAN repository data can also be stored in one or more recovery catalog databases, as an adjunct to the information stored in the control file, and to provide a longer history of backups.

While RMAN is designed to work without a recovery catalog, if you choose not to use a recovery catalog, you must perform some additional administrative tasks.

See Also: *Oracle Database Administrator's Guide* for a conceptual overview of the control file and more details about managing control files.

Backing Up and Restoring the Control File

If you are not using a recovery catalog, the control file is the sole storage for the RMAN repository, so it is doubly important that you protect it. Maintain alternate control files through multiplexing or operating system mirroring, and back up the control file frequently.

Configure `CONTROLFILE AUTOBACKUP` to `ON` to ensure extra protection for your control file.

So long as a control file autobackup is available, RMAN can restore the SPFILE and backup control file, and mount the database. After the control file is mounted, you can restore the remainder of the database.

Note that, if you restore a control file from autobackup, any persistent settings you set with the `CONFIGURE` command will revert to the values they had at the time of the control file autobackup. You should review the settings with the `SHOW ALL` command after restoring the control file.

See Also:

- ["Backing Up Control Files with RMAN"](#) to learn about manual and automatic control file backups
- *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to restore a database when the current control file and recovery catalog are unavailable

Monitoring the Overwriting of Control File Records

When you do not use a recovery catalog, the control file is the sole source of information about RMAN backups. As you make backups, Oracle records these backups in the control file. To prevent the control file from growing without bound to hold RMAN repository data, records can be reused if they are older than a threshold you specify.

The `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter determines the minimum age in days of a record before it can be overwritten:

```
CONTROL_FILE_RECORD_KEEP_TIME = integer
```

For example, if the parameter value is 14, then any record aged 14 days and older is a candidate for reuse. Information in an overwritten record is lost. The oldest record available for reuse will be used first.

When Oracle needs to add new RMAN repository records to the control file, but no record is older than the threshold, Oracle attempts to expand the size of the control file. If the underlying operating system prevents the expansion of the control file (due to a disk full condition, for instance), Oracle overwrites the oldest record in the control file and logs this action in the alert log.

The default value of `CONTROL_FILE_RECORD_KEEP_TIME` is 7 days. If you are not using a recovery catalog, then set the `CONTROL_FILE_RECORD_KEEP_TIME` value to slightly longer than the oldest file that you need to keep. For example, if you back up the whole database once a week, then you need to keep every backup for at least seven days. Set `CONTROL_FILE_RECORD_KEEP_TIME` to a value such as 10 or 14.

Caution: Regardless of whether you use a recovery catalog, never use RMAN when `CONTROL_FILE_RECORD_KEEP_TIME` is set to 0. If you do, then you may lose backup records.

Managing the Overwriting of Control File Records

Assume the following scenario:

- You do not use a recovery catalog.
- `CONTROL_FILE_RECORD_KEEP_TIME` is set to 14.
- All records currently in the control file are between 1 and 13 days old.

- The control file is at the maximum size permitted by the operating system.

You make a backup of the database. Because Oracle cannot expand the control file beyond the operating system file size limit, it begins overwriting records in the control file, starting with the oldest ones. For each record that it overwrites, it records an entry in the `alert.log` similar to the one shown here:

```
kccwmc: following control file record written over:
RECID #72 Recno 72 Record timestamp
07/28/00 22:15:21
Thread=1 Seq#=3460
Backup set key: stamp=372031415, count=17
Low scn: 0x0000.3af33f36
07/27/00 21:00:08
Next scn: 0x0000.3af3871b
07/27/00 23:23:54
Resetlogs scn and time
scn: 0x0000.00000001
08/05/99 10:46:44
Block count=102400 Blocksize=512
```

The most certain way to avoid this situation is to use a recovery catalog. If you cannot use a recovery catalog, then you can take the following measures to avoid problems associated with overwriting control file records:

- Store the control file in a file system rather than raw disk, so that it can expand as needed.
- Monitor the `alert.log` to make sure that Oracle is not overwriting control file records.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for a conceptual overview of control file records and how they are reused

Interaction of Flash Recovery Area and CONTROL_FILE_RECORD_KEEP_TIME

When a control file record containing information about a file created in the flash recovery area is about to be reused (because the record is older than `CONTROL_FILE_RECORD_KEEP_TIME`), if the file is eligible for deletion then the database attempts to delete the file from the flash recovery area. Otherwise, the database expands the size of the control file section containing the record for this file, logging the expansion in the alert log with a message like this example:

```
kccwmc: trying to expand control file section nnnn for Oracle Managed Files
```

where *nnnn* is the number of the control file record type.

If the control file is at the maximum size supported under the host operating system, then the database cannot expand the control file. In such a situation, this warning appears in the alert log:

```
WARNING: Oracle Managed File filename is unknown to control file. This is the
result of limitation in control file size that could not keep all recovery area
files.
```

This means that the control file cannot hold a record of all flash recovery area files needed to satisfy the configured retention policy.

There are several ways to avoid or alleviate this problem:

- Use a control file of larger block size, preferably one with 32K block size. To achieve this, you must set the SYSTEM tablespace block size to be greater than or equal to the control file block size, and re-create the control file after changing DB_BLOCK_SIZE.
- Make the files in the flash recovery area eligible for deletion, by backing them up to tertiary storage such as tape. For example, you can use the RMAN command `BACKUP RECOVERY AREA` to specifically back up files in the flash recovery area to your media manager.
- Make more files in the flash recovery area eligible for deletion, by changing the retention policy to a shorter recovery window or lower degree of redundancy.

Using CROSSCHECK to Update the RMAN Repository

To ensure that data about backups in the recovery catalog or control file is synchronized with corresponding data on disk or in the media management catalog, perform a **crosscheck**. The CROSSCHECK command operates only on files that are recorded in the recovery catalog or the control file.

This section contains these topics:

- [About RMAN Crosschecks](#)
- [Basic Use of CROSSCHECK with Backup Sets and Image Copies](#)
- [Crosschecking Specific Backup Sets and Copies](#)
- [Crosschecking Backups of Specific Database Files](#)

About RMAN Crosschecks

Crosschecks update outdated RMAN repository information about backups whose repository records do not match their physical status. For example, if a user removes archived logs from disk with an operating system command, the repository still indicates that the logs are on disk, when in fact they are not.

If the backup is on disk, then the CROSSCHECK command determines whether the header of the file is valid. If the backup is on tape, then the command simply checks that the backup exists. The possible status values for backups are AVAILABLE, UNAVAILABLE, and EXPIRED.

You can view the status of backups using the RMAN LIST command, or by querying V\$BACKUP_FILES or many of the recovery catalog views such as RC_DATAFILE_COPY or RC_ARCHIVED_LOG. Running a crosscheck updates the RMAN repository so that all of these methods provide accurate information. For each backup in the RMAN repository, if the backup is no longer available, then RMAN marks it as EXPIRED. If it was EXPIRED and is now available, then RMAN marks it AVAILABLE.

Note: The CROSSCHECK command does *not* delete operating system files, and it does not remove RMAN repository records of backups that are not available at the time of the crosscheck. It only updates the repository records with the status of the backups. Use the DELETE command to remove records of expired backups from the RMAN repository.

- : ■ ["Deleting Backups"](#) on page 8-6 to learn how to delete unwanted backups and update repository records
- *Oracle Database Backup and Recovery Reference* for CROSSCHECK command syntax and a description of the repository status values

Basic Use of CROSSCHECK with Backup Sets and Image Copies

After connecting to the target database and recovery catalog (if you use one), run CROSSCHECK commands as needed to verify the status and availability of backups known to RMAN.

If you have channels configured for your tape or other media manager, you can crosscheck all backups on all media with CROSSCHECK BACKUP as shown in this example:

```
CROSSCHECK BACKUP;
```

If you do not have configured channels for your tape backups, you can allocate a maintenance channel in a RUN block before running CROSSCHECK as in this example:

```
RUN {
    ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
    CROSSCHECK BACKUP;
}
```

This example shows how to crosscheck disk image copies only:

```
CROSSCHECK COPY;    # crosschecks only disk copies of
                   # database files
```

This example shows how to crosscheck backup sets only:

```
CROSSCHECK BACKUPSET;    # crosschecks backupsets on disk and SBT
```

Crosschecking Specific Backup Sets and Copies

You can use the LIST command to report your backups and then use the CROSSCHECK command to check that specific backups described in the LIST output still exist. The DELETE EXPIRED command deletes repository records for backups that fail the crosscheck.

To crosscheck specified backups:

1. Identify the desired backups that you want to check by issuing a LIST command. For example, issue:

```
LIST BACKUP;    # lists all backup sets, proxy copies, and image copies
```

2. Check whether the specified backups still exist. For example, enter:

```
CROSSCHECK BACKUP;    # checks backup sets, proxy copies, and image copies
CROSSCHECK COPY OF DATABASE;
CROSSCHECK BACKUPSET 1338, 1339, 1340;
CROSSCHECK BACKUPPIECE TAG = 'nightly_backup';
CROSSCHECK CONTROLFILECOPY '/tmp/control01.ctl';
CROSSCHECK DATAFILECOPY 113, 114, 115;
CROSSCHECK PROXY 789;
```

Crosschecking Backups of Specific Database Files

You can use options of the `CROSSCHECK` command to check only backups of a specific database, tablespace, datafile, control file, or archived redo log. For example:

```
CROSSCHECK BACKUP OF DATAFILE "?/oradata/trgt/system01.dbf"
  COMPLETED AFTER 'SYSDATE-14';
CROSSCHECK BACKUP OF ARCHIVELOG ALL SPFILE;
```

You can check for backups of archived redo logs and SPFILE using this command:

```
CROSSCHECK BACKUP OF ARCHIVELOG ALL SPFILE;
```

See Also: *Oracle Database Backup and Recovery Reference* for more details on using `CROSSCHECK` to check backups of specific database files

Limiting RMAN CROSSCHECK to a Backups Since a Specific Time

You can add a `COMPLETED AFTER` clause to a `CROSSCHECK` command to restrict the checking to backups created after a specified point in time. For example, this command checks for backups of a datafile created in the last week:

```
CROSSCHECK BACKUP OF DATAFILE 4
  COMPLETED AFTER 'SYSDATE-14';
```

Deleting Backups

You can use `RMAN` to delete backups created with `RMAN`. Deleting backups using `RMAN` both deletes the specified backups and updates the `RMAN` repository to reflect the deletion.

This section contains these topics:

- [Deleting Specified Backups](#)
- [Deleting Expired RMAN Backups after CROSSCHECK](#)
- [Using DELETE FORCE With RMAN Backups](#)
- [Deleting Obsolete RMAN Backups Based on Retention Policies](#)

Note: Backups with `DELETED` status do not appear in the `LIST` command output. You can, however, query `V$` control file views to get information about deleted backups.

See Also:

- *Oracle Database Backup and Recovery Reference* for `DELETE` syntax
- *Oracle Database Backup and Recovery Reference* for descriptions of the recovery catalog views

Deleting Specified Backups

In general, use the `DELETE` command to remove backups that you do not want to retain. `DELETE` removes the physical files from the backup media, deletes the record of the backup from the recovery catalog (if `RMAN` is connected to a recovery catalog), and updates the records of these backups in the control file to status `DELETED`.

The `DELETE` command supports deleting several types of backups, including:

DELETE BACKUP (which deletes backup sets, proxy copies, and image copies), DELETE COPY (which deletes only image copies), or DELETE ARCHIVELOG as in these examples:

The DELETE command supports a wide range of options to identify backups to delete. For complete information about these options, see *Oracle Database Backup and Recovery Reference*. The following examples show many of the common ways to specify backups and archived logs to delete using the DELETE command:

- Deleting backups using primary keys from LIST output:

```
DELETE BACKUPPIECE 101;
```

- Deleting backups by filename on disk:

```
DELETE CONTROLFILECOPY '/tmp/control01.ctl';
```

- Deleting archived redo logs from disk:

```
DELETE NOPROMPT ARCHIVELOG UNTIL SEQUENCE = 300;
```

- Deleting backups based on tags:

```
DELETE BACKUP TAG='before_upgrade';
```

- Delete backups based on the objects backed up and the media or disk location where the backup is stored:

```
DELETE BACKUP OF TABLESPACE users DEVICE TYPE sbt; # delete only from tape
DELETE COPY OF CONTROLFILE LIKE '/tmp/%'; #
```

- Delete all backups for this database recorded in the RMAN repository:

```
DELETE BACKUP;
```

- Delete backups and archived redo logs from disk based on whether they are backed up on tape:

```
DELETE ARCHIVELOG ALL
BACKED UP 3 TIMES TO sbt;
```

If you run RMAN interactively, then RMAN asks for confirmation before deleting any files. You can suppress these confirmations by using the NOPROMPT keyword with any form of the BACKUP command:

```
DELETE NOPROMPT ARCHIVELOG ALL;
```

Deleting Expired RMAN Backups after CROSSCHECK

When the CROSSCHECK command is used to determine whether backups recorded in the repository still exist on disk or tape, if RMAN cannot locate the backups, then it updates their records in the RMAN repository to EXPIRED status. You can then use the DELETE EXPIRED command to remove records of expired backups from the RMAN repository. If the expired files still exist, then the DELETE EXPIRED command terminates with an error.

To delete expired repository records:

1. If you have not performed a crosscheck recently, then issue a CROSSCHECK command. For example, issue:

```
CROSSCHECK BACKUP; # checks backup sets and copies on configured channels
```

2. Delete the expired backups. For example, issue:

```
DELETE EXPIRED BACKUP;
```

Note: The `DELETE EXPIRED` command issues warnings if any objects marked as `EXPIRED` actually exist. In rare cases, the repository can mark an object as `EXPIRED` even though the object exists. For example, a directory containing an object is corrupted at the time of the crosscheck, but is later repaired, or the media manager was not configured properly and reported some backups as not existing when they really existed.

Using `DELETE FORCE` With RMAN Backups

It is possible for the RMAN repository to indicate that an object has one status while the actual status of the object on the media is different. For example, the RMAN repository says that a backup set is `AVAILABLE` when it is in fact no longer present on disk or tape (or missing from the media manager's catalog of the contents of tapes or other backup media). If you attempt to delete the object, then you receive a warning such as the following:

```
RMAN-06207: WARNING: 1 objects could not be deleted for DISK channel(s) due
RMAN-06208:           to mismatched status. Use CROSSCHECK command to fix status
List of Mismatched objects
=====
   Object Type   Filename/Handle
-----
Backup Piece    0id270ud_1_1
```

If you use the `DELETE` command with the optional `FORCE` keyword, RMAN deletes the specified backups, but ignores any I/O errors, including those that occur when a backup is missing from disk or tape. It then updates the RMAN repository to reflect the fact that the backup is deleted, regardless of whether RMAN was able to delete the file or whether the file was already missing. For example:

```
DELETE FORCE NOPROMPT BACKUPSET TAG 'weekly_bkup';
```

See Also:

- *Oracle Database Backup and Recovery Advanced User's Guide* to learn about `DELETE` behavior when the backup media and repository are not synchronized
- *Oracle Database Backup and Recovery Reference* for `DELETE` command syntax

Deleting Obsolete RMAN Backups Based on Retention Policies

The RMAN `DELETE` command supports an `OBSOLETE` option, which deletes backups that are obsolete, that is, no longer needed to satisfy specified recoverability requirements. You can delete files obsolete according to the configured default retention policy, or another retention policy that you specify as an option to the `DELETE OBSOLETE` command. As with other forms of the `DELETE` command, the files deleted are removed from backup media, deleted from the recovery catalog, and marked as `DELETED` in the control file.

If you specify the `DELETE OBSOLETE` command with no arguments, then RMAN deletes all obsolete backups defined by the currently configured retention policy. For example:

```
DELETE OBSOLETE;
```

You can also use the `REDUNDANCY` or `RECOVERY WINDOW` clauses with `DELETE` to delete backups obsolete under a specific retention policy instead of the configured default:

```
DELETE OBSOLETE REDUNDANCY = 3;
DELETE OBSOLETE RECOVERY WINDOW OF 7 DAYS;
```

DELETE OBSOLETE Behavior When KEEP UNTIL Time Expires

`DELETE OBSOLETE` does not delete backups required to satisfy the specified retention policy, even if some backups have `KEEP UNTIL` times set which have passed to override the retention policy.

The `KEEP UNTIL` clause never causes RMAN to consider a backup obsolete, if the backup is still required to satisfy the retention policy. `KEEP UNTIL` can cause backups to be kept longer than the retention policy requires, but never causes a backup to become obsolete sooner than the retention policy requires.

Backups are never obsolete if they are still needed to meet the retention policy, regardless of any `KEEP UNTIL` time. With a recovery window-based retention policy, even if the specified `KEEP UNTIL` time has expired, the backup is retained if the backup is needed to satisfy the recovery window. With a redundancy-based retention policy, even if the specified `KEEP UNTIL` time has expired, the backup is retained as long as it is required to satisfy the redundancy requirement.

Using Multiple RMAN Channels for Maintenance Operations

This section contains these topics:

- [About Allocating Multiple RMAN Channels for Maintenance Commands](#)
- [How RMAN Crosschecks and Deletes on Multiple Channels](#)
- [Crosschecking Disk and Tape Channels with One Command: Example](#)
- [Crosschecking on Multiple Oracle Real Application Cluster Nodes: Example](#)
- [Deleting on Disk and Tape Channels with One DELETE Command: Example](#)
- [Releasing Multiple Channels: Example](#)

About Allocating Multiple RMAN Channels for Maintenance Commands

You can configure or manually allocate multiple channels before issuing `CROSSCHECK` or `DELETE` commands. RMAN searches for each backup on all channels that have the same device type as the channel used to create the backup. The multichannel feature is primarily intended for use when crosschecking or deleting backups on both disk and tape within a single command.

How RMAN Crosschecks and Deletes on Multiple Channels

When you configure or manually allocate multiple channels and run a `CROSSCHECK` or `DELETE` command, RMAN performs the crosscheck or delete on all channels that have the appropriate device type.

For example, assume that you prefer to manually allocate channels rather than use configured channels. You have a media manager configured, but have never made backups to tape. You have created only one backup of a database to disk, as follows:

```
RUN
{
  ALLOCATE CHANNEL ch1 DEVICE TYPE DISK CONNECT 'SYS/sys_pwd@node2';
  BACKUP DATABASE;
}
```

Assume that you issue the following series of commands at the RMAN prompt:

```
ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE DISK CONNECT 'SYS/oracle@node1';
ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE DISK CONNECT 'SYS/oracle@node2';
ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
CROSSCHECK BACKUP OF DATABASE;
```

Because the RMAN repository has a record of the database backup to disk, it crosschecks backups accessible on the two disk channels (and finds the backup on the second channel). Because the RMAN repository has no record of any *sbt* backups, the *CROSSCHECK* command does not access the allocated SBT channel.

Crosschecking Disk and Tape Channels with One Command: Example

If you have multiple configured channels for disk and tape, RMAN uses all configured channels to perform crosschecks on both backup destinations simultaneously.

For example, assume that you have an *sbt* channel configured as follows:

```
CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
CONFIGURE DEFAULT DEVICE TYPE sbt;
```

In this case you can run the following command to perform a crosscheck on both *DISK* and *sbt*:

```
CROSSCHECK BACKUP OF DATABASE;
```

RMAN uses both the *sbt* channel and the preconfigured *DISK* channel to perform the crosscheck. Sample output follows:

```
allocated channel: ORA_SBT_TAPE_1
channel ORA_SBT_TAPE_1: sid=12 devtype=SBT_TAPE
channel ORA_SBT_TAPE_1: WARNING: Oracle Test Disk API
using channel ORA_DISK_1
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=/oracle/dbs/16c5esv4_1_1 recid=36 stamp=408384484
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=/oracle/dbs/c-674966176-20000915-01 recid=37 stamp=408384496
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=12c5erb2_1_1 recid=32 stamp=408382820
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=13c5erba_1_1 recid=33 stamp=408382829
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=14c5erce_1_1 recid=34 stamp=408382863
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=c-674966176-20000915-00 recid=35 stamp=408382869
```

If you do not have an automatic *sbt* channel configured, then can also manually allocate maintenance channels on disk and tape as in the following example:

```
RUN {
  ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
```

```
CROSSCHECK BACKUP OF DATABASE;
}
```

Note that you do not have to manually allocate a disk channel because RMAN uses the preconfigured disk channel.

Crosschecking on Multiple Oracle Real Application Cluster Nodes: Example

When crosschecking on multiple nodes (and when operating RMAN in general), configure the cluster so that all backups can be accessed by every node, regardless of which node created the backup. When the cluster is configured this way, you can allocate channels at any node in the cluster during restore or crosscheck operations.

If you cannot configure the cluster so that each node can access all backups, then during restore and crosscheck operations, you must allocate channels on multiple nodes by providing the `CONNECT` option to the `CONFIGURE CHANNEL` command, so that every backup can be accessed by at least one node. If some backups are not accessible during crosscheck because no channel was configured on the node that can access those backups, then those backups are marked `EXPIRED` in the RMAN repository after the crosscheck.

For example, you can use `CONFIGURE CHANNEL... CONNECT...` in an Oracle Real Application Cluster (RAC) configuration in which tape backups are created on various nodes in the cluster and each backup is only accessible on the node on which it is created.

In this example, assume a RAC cluster with two nodes, each one of which requires a channel to access some backups for the crosscheck. You configure the channels as shown here:

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 2;
CONFIGURE CHANNEL 1 DEVICE TYPE DISK CONNECT 'SYS/oracle@node_1';
CONFIGURE CHANNEL 2 DEVICE TYPE DISK CONNECT 'SYS/oracle@node_2';
```

Then, crosscheck the cluster nodes with the following command:

```
CROSSCHECK BACKUP;
```

Deleting on Disk and Tape Channels with One DELETE Command: Example

You can also perform deletions on all allocated channels. In the following example, you configure an `sbt` channel:

```
CONFIGURE DEVICE TYPE sbt PARALLELISM 1;
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

Then, you run a command to delete all backup sets from disk and tape:

```
DELETE BACKUPSET;
```

RMAN uses both the configured `sbt` channel and the preconfigured `DISK` channel when deleting. In the following sample output, note that RMAN prompts you for confirmation before deleting any files:

```
using channel ORA_SBT_TAPE_1
using channel ORA_DISK_1
```

List of Backup Pieces

BP Key	BS Key	Pc#	Cp#	Status	Device Type	Piece Name
388	387	1	1	AVAILABLE	SBT_TAPE	12c5erb2_1_1

```

397      396      1      1      UNAVAILABLE SBT_TAPE      13c5erba_1_1
424      423      1      1      AVAILABLE   SBT_TAPE      14c5erce_1_1
428      427      1      1      AVAILABLE   SBT_TAPE      c-674966176-20000915-00
433      432      1      1      AVAILABLE   DISK          /oracle/dbs/16c5esv4_1_1
437      436      1      1      AVAILABLE   DISK          /oracle/dbs/c-674966176-20000915-01

```

```

Do you really want to delete the above objects (enter YES or NO)? y
deleted backup piece
backup piece handle=/oracle/dbs/16c5esv4_1_1 recid=36 stamp=408384484
deleted backup piece
backup piece handle=/oracle/dbs/c-674966176-20000915-01 recid=37 stamp=408384496
deleted backup piece
backup piece handle=12c5erb2_1_1 recid=32 stamp=408382820
deleted backup piece
backup piece handle=13c5erba_1_1 recid=33 stamp=408382829
deleted backup piece
backup piece handle=14c5erce_1_1 recid=34 stamp=408382863
deleted backup piece
backup piece handle=c-674966176-20000915-00 recid=35 stamp=408382869

```

The following example manually allocates DISK and sbt maintenance channels and then deletes specific backup sets from both disk and tape:

```

RUN {
    ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE DISK;
    ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
    DELETE BACKUPSET 1,2,3,4,5;
}

```

RMAN looks for the specified backup sets on the channels and deletes any it finds. If RMAN does not find a backup on any channel, then RMAN marks the object as deleted in the control file and deletes the recovery catalog record (if you use a recovery catalog).

Releasing Multiple Channels: Example

You can release all allocated maintenance channels by running this command:

```
RELEASE CHANNEL;
```

Deleting a Database with RMAN

You may need to remove a database from the operating system. For example, you create a test database and then no longer have a use for it. In such a situation, use the DROP DATABASE command from within RMAN, or the DROP DATABASE statement in SQL*Plus.

DROP DATABASE requires that RMAN be connected to the target database, and that the target database be mounted. The command does not require connection to the recovery catalog. If RMAN is connected to the recovery catalog, and if you specify the option INCLUDE COPIES AND BACKUPS, then RMAN also unregisters the database.

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to use the SQL*Plus DROP DATABASE command

To drop a database:

1. Connect RMAN to the target database and (optionally) recovery catalog. For example:

```
rman TARGET / CATALOG rman/rman@catdb
```

2. Catalog all backups that are associated with the database. For example, the following commands catalogs files in the flash recovery area, and then in a secondary archiving destination:

```
RMAN> CATALOG START WITH '+disk1';      # all files from flash recovery area
                                           # (stored on ASM disk)
RMAN> CATALOG START WITH '/arch_dest2'; # all files from second arch dest
```

3. Delete all backups and copies associated with the database. For example:

```
RMAN> DELETE BACKUPSET; # deletes all backups
RMAN> DELETE COPY; # delete all image copies (including archived logs)
```

4. Remove the database from the operating system (and automatically unregister it from the recovery catalog if you are connected to the catalog). For example:

```
DROP DATABASE; # delete all database files and unregister the database
```

Changing the Status of a Backup Record

This section contains the following sections:

- [Marking a Backup AVAILABLE or UNAVAILABLE](#)
- [Exempting a Long-Term Backup from the Retention Policy](#)

Marking a Backup AVAILABLE or UNAVAILABLE

Run the `CHANGE . . . UNAVAILABLE` command when a backup cannot be found or has migrated offsite. RMAN does not use files marked UNAVAILABLE in `RESTORE` or `RECOVER` commands. If the file is later found or returns to the main site, then you can mark it available again by issuing `CHANGE . . . AVAILABLE`.

Note that files in the flash recovery area cannot be marked as UNAVAILABLE.

To mark a file's status in the repository as UNAVAILABLE or AVAILABLE:

1. Issue a `LIST` command to determine the availability status of RMAN backups. For example, issue:

```
LIST BACKUP;
```

2. Run `CHANGE` with the `UNAVAILABLE` or `AVAILABLE` keyword to update its status in the RMAN repository. For example, enter:

```
CHANGE DATAFILECOPY '/tmp/control01.ctl' UNAVAILABLE;
CHANGE COPY OF ARCHIVELOG SEQUENCE BETWEEN 1000 AND 1012 UNAVAILABLE;
CHANGE BACKUPSET 12 UNAVAILABLE;
CHANGE BACKUP OF SPFILE TAG "TAG20020208T154556" UNAVAILABLE;
CHANGE DATAFILECOPY '/tmp/system01.dbf' AVAILABLE;
CHANGE BACKUPSET 12 AVAILABLE;
CHANGE BACKUP OF SPFILE TAG "TAG20020208T154556" AVAILABLE;
```

See Also: *Oracle Database Backup and Recovery Reference* for `CHANGE` command syntax

Exempting a Long-Term Backup from the Retention Policy

The `BACKUP . . . KEEP` command can create a backup that is retained for a different period of time from that specified by the configured retention policy. The backup is

still a fully valid backup, however, and can be restored just as any other RMAN backup. This type of backup is called a **long-term backup**.

Note: The `KEEP FOREVER` clause requires the use of a recovery catalog, because the control file cannot contain an infinitely large set of RMAN repository data.

Specify `KEEP . . . LOGS` to save archived logs for a possible incomplete recovery and `KEEP . . . NOLOGS` not to save archived logs for a possible incomplete recovery. Note that `NOLOGS` is not valid with an inconsistent backup.

Use the `CHANGE` command to alter the `KEEP` status of a backup that already exists. For example, you may decide that you no longer want to keep a long-term backup. The same options available for `BACKUP . . . KEEP` are available with `CHANGE . . . KEEP`.

Note that you cannot set `KEEP` attributes on files stored in the flash recovery area.

To alter the KEEP status of a backup:

Issue `CHANGE . . . KEEP` to define a different retention period for this backup, or `CHANGE . . . NOKEEP` to let the retention policy apply to this file.

This example allows a backup set to be marked obsolete by the retention policy:

```
CHANGE BACKUPSET 231 NOKEEP;
```

This example makes a datafile copy exempt from the retention policy for 180 days (6 months):

```
CHANGE DATAFILECOPY '/tmp/system01.dbf' KEEP UNTIL 'SYSDATE+180';
```

Note: If you use `KEEP UNTIL` to specify a minimum time to keep a backup, the backup can be kept longer than the time specified if it is needed to satisfy the retention policy. RMAN's `DELETE OBSOLETE` command does not delete backups needed to satisfy the retention policy, even if `KEEP UNTIL` has been used to set a minimum length of time for the backup to be kept.

Cataloging Archived Logs and User-Managed Copies

You can make RMAN aware of the existence of archived logs that are not recorded in the repository as well as file and backup piece copies that are created through means other than RMAN. This section contains the following topics:

- [About Cataloging Archived Logs and User-Managed Copies](#)
- [Cataloging User-Managed Datafile Copies](#)
- [Cataloging Backup Pieces](#)
- [Cataloging All Files in a Disk Location](#)

About Cataloging Archived Logs and User-Managed Copies

The target database control file keeps records of all archived logs generated by the target database as well as all RMAN backups. The purpose of the `CATALOG` command is to add metadata to the repository when it does not have a record of files that you want RMAN to know about.

Run the RMAN CATALOG command when:

- You use an operating system utility to make copies of datafiles, archived logs, or backup pieces. In the case, the repository has no record of them.
- You perform recovery with a backup control file and you change the archiving destination or format during recovery. In this situation, the repository will not have information about archived logs needed for recovery. Hence, you must catalog these logs.
- You want to catalog datafile copy as a level 0 backup, thus enabling you to perform an incremental backup later by using the datafile copy as the base of an incremental backup strategy
- You want to catalog user-managed copies of Oracle7 database files created before you migrated to a higher release, or of Oracle8 and higher database files created before you started to use RMAN. These datafile copies enable you to recover the database if it crashes after migration but before you have a chance to take a backup of the migrated database.

Whenever you make a user-managed copy, for example, by using the UNIX `cp` command to copy a datafile, make sure to catalog it. When making user-managed copies, you can use the `ALTER TABLESPACE . . . BEGIN/END BACKUP` statement to make datafile copies off an online tablespace. Although RMAN does not create such datafile copies, you can use the CATALOG command to add them to the recovery catalog so that RMAN is aware of them.

For a user-managed copy to be cataloged, it must be:

- Accessible on disk
- A complete image copy of a single file
- Either a datafile copy, control file copy, archived redo log copy, or backup piece copy

For example, if you store datafiles on mirrored disk drives, then you can create a user-managed copy by breaking the mirror. In this scenario, use the CATALOG command to notify RMAN of the existence of the user-managed copy after breaking the mirror. Before reforming the mirror, run a `CHANGE . . . UNCATALOG` command to notify RMAN that the file copy no longer exists.

Cataloging User-Managed Datafile Copies

Use the CATALOG command to propagate information about user-managed copies to the RMAN repository. After the files are cataloged, you can run LIST or query `V$BACKUP_FILES` to confirm.

To create and catalog a user-managed copy of a datafile:

1. Make a datafile copy with an operating system utility. `ALTER TABLESPACE BEGIN/END BACKUP` is necessary if the database is open and the datafiles are online while the backup is in progress. This example backs up an online datafile.

```
SQL> ALTER TABLESPACE users BEGIN BACKUP;
% cp $ORACLE_HOME/oradata/trgt/users01.dbf /tmp/users01.dbf;
SQL> ALTER TABLESPACE users END BACKUP;
```

2. After connecting to the target database and, if desired, the recovery catalog, run the CATALOG command. For example, enter:

```
CATALOG DATAFILECOPY '/tmp/users01.dbf';
```

If you try to catalog a datafile copy from a database other than the connected target database, then RMAN issues an error such as the following:

```
RMAN-00571: =====
RMAN-00569: ===== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03009: failure of catalog command on default channel at 08/29/2001
14:44:34
ORA-19563: datafile copy header validation failed for file /tmp/tools01.dbf
```

See Also: *Oracle Database Backup and Recovery Reference* for CATALOG command syntax

Cataloging Backup Pieces

You can catalog backup pieces on disk. This technique is useful if you use an operating system utility to copy backup pieces from location to another on the same host, or from one host to another. You can even catalog a backup piece from a prior incarnation of the database. RMAN can determine whether that backup piece can be used during a subsequent restore and recovery operation.

To catalog a backup piece:

1. After connecting RMAN to the target database, catalog the filenames of the backup pieces. For example:

```
CATALOG BACKUPPIECE '/disk2/09dtq55d_1_2', '/disk2/0bdtqdou_1_1';
```

After a backup piece is cataloged, you can display its metadata by querying V\$BACKUP_PIECE, V\$BACKUP_SET, V\$BACKUP_DATAFILE, V\$BACKUP_REDOLOG, and V\$BACKUP_SPFIL.

Note: If you are cataloging backup pieces from a release earlier than Oracle9i, you can achieve significant performance gains by cataloging the highest copy numbers first. Otherwise, RMAN must examine all pieces to determine the correct order. For example, catalog copy 3 of a piece before copy 2:

```
CATALOG BACKUPPIECE '/disk2/09dtq55d_1_3',
'/disk2/09dtq55d_1_2';
```

Backup pieces from Oracle Database Release 10g are not affected by this issue and can be cataloged in any order.

2. You can query V\$ views to verify your changes. For example:

```
SELECT HANDLE FROM V$BACKUP_PIECE;
```

See Also: *Oracle Database Backup and Recovery Reference* for CATALOG BACKUPPIECE restrictions

Cataloging All Files in a Disk Location

If you use Automatic Storage Management (ASM), an Oracle Managed Files framework, or the flash recovery area, then you may need a way to recatalog files that are known to the disk management system but are no longer listed in the RMAN repository. This situation can occur when the intended mechanisms for tracking filenames fails due to media failure, software bug, or user error.

The `CATALOG START WITH` command enables you to search through all files in an ASM disk group, Oracle Managed Files location, or traditional file system directory and investigate those that are not recorded in the RMAN repository. If the command can catalog a file, then it will; if it cannot catalog it, then it makes its best guess about the contents of the skipped file.

To catalog all files in a disk location:

After connecting RMAN to the target database, specify the disk location whose files you want to catalog. For example:

```
RMAN> CATALOG START WITH '+disk'; # catalog all files from an ASM disk group
RMAN> CATALOG START WITH '/fs1/datafiles/'; # catalog all files in directory
```

Note: Wildcard characters are not legal in the `START WITH` clause.

Cataloging Flash Recovery Area Contents

The `CATALOG RECOVERY AREA` command will catalog all files in the flash recovery area. During this operation, any files in the recovery area that are not listed in the RMAN repository are added to the RMAN repository. For example:

```
CATALOG RECOVERY AREA;
```

Uncataloging RMAN Records

This section contains the following topics:

- [About Uncataloging RMAN Records](#)
- [Removing Records for Files Deleted with Operating System Utilities](#)

About Uncataloging RMAN Records

Run the `CHANGE . . . UNCATALOG` command to perform the following actions on RMAN repository records:

- Update a backup record in the control file repository to status `DELETED`
- Delete a specific backup record from the recovery catalog (if you use one)

RMAN does not touch the specified physical files: it only alters the repository records for these files.

You can use this command when you have deleted a backup through a means other than RMAN. For example, if you delete archived redo logs with an operating system utility, then remove the record for this log from the repository by issuing `CHANGE ARCHIVELOG . . . UNCATALOG`.

Removing Records for Files Deleted with Operating System Utilities

To remove catalog records for files deleted with operating system utilities, run the `CHANGE . . . UNCATALOG` command.

To remove the catalog record for a backup deleted with an operating system utility:

1. Run a `CHANGE . . . UNCATALOG` command for the backups that you deleted from the operating system with operating system commands. This example deletes repository references to disk copies of the control file and datafile 1:

```
CHANGE CONTROLFILECOPY '/tmp/control01.ctl' UNCATALOG;  
CHANGE DATAFILECOPY '/tmp/system01.dbf' UNCATALOG;
```

2. Optionally, view the relevant recovery catalog view, for example, `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:

```
SELECT CDF_KEY, STATUS  
FROM RC_DATAFILE_COPY  
WHERE CDF_KEY = 4833;
```

```
CDF_KEY    STATUS  
-----  
0 rows selected.
```

Flash Recovery Area Maintenance

While the flash recovery area is largely self-managing, there are some situations in which DBA intervention may be required.

Resolving a Full Flash Recovery Area

You have a number of choices on how to resolve a full flash recovery area when there are no files eligible for deletion:

- Make more disk space available, and increase `DB_RECOVERY_FILE_DEST_SIZE` to reflect the new space.
- Move backups from the flash recovery area to a tertiary device such as tape. One convenient way to back up all of your flash recovery area files to tape at once is the `BACKUP RECOVERY AREA` command.

After you transfer backups from the flash recovery area to tape, you can resolve the full recovery area condition by deleting files from the flash recovery area, using forms of the `RMAN DELETE` command.

Note: Flashback logs, by design, cannot be backed up outside the flash recovery area. Therefore, in a `BACKUP RECOVERY AREA` operation the flashback logs are not backed up to tape.

Flashback logs are deleted automatically to satisfy the need for space for other files in the flash recovery area. However, a guaranteed restore point can force the retention of flashback logs required to perform Flashback Database to the restore point SCN. See

- Delete unnecessary files from the flash recovery area using the `RMAN DELETE` command. (Note that if you use host operating system commands to delete files, then the database will not be aware of the resulting free space. You can run the `RMAN CROSSCHECK` command to have RMAN re-check the contents of the flash recovery area and identify expired files, and then use the `DELETE EXPIRED` command to remove missing files from the RMAN repository.)

You may also need to consider changing your backup retention policy and, if using Data Guard, consider changing your archivelog deletion policy.

See Also: [Chapter 4, "Backing Up Databases Using RMAN"](#) for more on how to decide on a retention policy, and *Oracle Data Guard Concepts and Administration* for more on archivelog deletion policy with Data Guard

Changing the Flash Recovery Area to a New Location

If you need to move the flash recovery area of your database to a new location, you can follow this procedure:

1. Invoke SQL*Plus to change the `DB_RECOVERY_FILE_DEST` initialization parameter. For example:

```
ALTER SYSTEM SET DB_RECOVERY_FILE_DEST='+disk1' SCOPE=BOTH SID='*';
```

After you change this parameter, all new flash recovery area files will be created in the new location.

2. The permanent files (control files and online redo log files), flashback logs and transient files can be left in the old flash recovery area location. The database will delete the transient files from the old flash recovery area location as they become eligible for deletion.

If you need to actually move your current permanent files, transient files, or flashback logs to the new flash recovery area, see *Oracle Database Backup and Recovery Advanced User's Guide*. The process outlined there for moving database files into and out of an ASM disk group with RMAN will also work when moving files into and out of a flash recovery area location.

See Also: ["Backing Up to the Flash Recovery Area and to Tape: Basic Scenarios"](#) on page A-7

Oracle will clean up transient files remaining in the old flash recovery area location as they become eligible for deletion.

Flash Recovery Area Behavior When Instance Crashes During File Creation

As a rule, the flash recovery area is self-maintaining, but when an instance crashes during the creation of a file in the flash recovery area, Oracle may leave the file in the flash recovery area. When this occurs, you will see the following error in the alert log:

```
ORA-19816: WARNING: Files may exist in location that are not known to database.
```

where *location* is the location of the flash recovery area.

In such a situation, you should use the RMAN command `CATALOG RECOVERY AREA` to re-catalog any such files so that they appear in the RMAN repository. If the file header of the file in question is corrupted, then you will have to delete the file manually using an operating system-level utility.

RMAN-Based Disk and Tape Backup Strategies: Scenarios

This chapter presents in detail several disk-only and disk-and-tape-based backup strategies that make effective use of RMAN, incrementally updated backups, and the flash recovery area.

This chapter contains the following sections:

- [Backing Up to the Flash Recovery Area: Basic Scenarios](#)
- [Backing Up to the Flash Recovery Area and to Tape: Basic Scenarios](#)

Backing Up to the Flash Recovery Area: Basic Scenarios

For all of the following scenarios, assume that the RMAN environment and flash recovery area are configured as shown in "[Configure Flash Recovery Area for Disk-Based Backups: Example](#)" on page 3-21.

Scripting Disk-Only Backups

The details of your backup script depend on the database load and the amount of disk space allocated. The following scripts are categorized based on different patterns of database updates and the minimum amount of disk space allocated for the flash recovery area required to achieve the recovery window on disk.

If few database blocks change, then the incremental backup size will be significantly less than the size of the database, and the best practice is to take incremental backups, to make efficient use of disk space and other resources.

If most or all database blocks change frequently, then the size of incremental backup will be roughly as large as the size of the database, and the best practice is to periodically make a complete image copy of the database.

In the following scenarios, you do not need to back up archived logs because Oracle archives the logs to the flash recovery area. They will remain in the flash recovery area for as long as they are required by the retention policy. All database backups are also all directed to the flash recovery area.

This section contains the following topics:

- [Backup Scripts When Few Data Blocks Change](#)
- [Backup Scripts When Blocks Change Frequently](#)
- [Backup Scripts When a Moderate Number of Blocks Change Weekly](#)

Backup Scripts When Few Data Blocks Change

If most data blocks do not change each day, incremental backups will be small and you can plan your strategy around daily incremental backups.

Initial Setup The minimum recommended flash recovery area size on disk is dependent on the how frequent the backups are taken and retention policy used.

If your configured retention policy is REDUNDANCY X and you plan to take a backup every Y days, then the formula for the recommended flash recovery area disk quota is:

```
Disk Quota =  
Size of X copies of database +  
Size of one copy of incremental backups +  
Size of (Y+1) days of archived logs
```

If you configure your retention policy to a recovery window of X days and execute your backup script once in Y days then you can calculate your required disk quota by one of two formulas. If $X \geq Y$ then the formula for the recommended flash recovery area disk quota is:

```
Disk Quota =  
Size of one copy of database +  
Size of (floor(X/Y) + 1) copies of incremental backups +  
Size of (X * ceil(X/Y) + 1) days of archived logs
```

where $\text{ceil}(X/Y)$ is the smallest integer greater than or equal to X/Y and $\text{floor}(X/Y)$ is the largest integer less than or equal to X/Y .

If $X < Y$ then the recommended formula is:

```
Disk Quota =  
Size of one copy of database +  
Size of 2 copies of incremental backups +  
Size of (Y + 1) days of archived logs
```

Size your flash recovery area according to the applicable formula.

For this example, assume that the retention policy is REDUNDANCY 1:

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 1;
```

Also assume that backups will be performed daily.

Daily Script The daily backup script would look like this:

```
RMAN> RECOVER COPY OF DATABASE WITH TAG "whole_db_copy";  
# Make an incremental backup of the database to the flash recovery area.  
RMAN> BACKUP INCREMENTAL LEVEL 1  
      FOR RECOVER OF COPY WITH TAG "whole_db_copy"  
      DATABASE;
```

Assume that there are no backups tagged "whole_db_copy" as of the first day of this backup scheme. The results of running this script daily are as follows:

- On the first day, the RECOVER statement would have no effect, and the BACKUP... FOR RECOVER OF COPY would create the initial level 0 database copy.
- On the second day, the RECOVER statement still has no effect, as there is no level 1 incremental backup to be applied to the level 0 incremental database copy. The BACKUP command creates the first incremental level 1 backup.

- Each day after that, the incremental level 1 from the previous day is applied over the level 0 database copy, rolling it forward to the time of the incremental level 1 of the previous day, and a new incremental level 1 is created, containing all changes since the level 1 incremental backup of the previous day.

Assuming that this backup strategy is put into effect on February 1, the following table shows how the flash recovery area contents change over time as a result of the strategy. (Note that the flash recovery area may contain other files based on other backup activities; this table only represents files related to this strategy.)

Table A-1 Backup Timeline for Scenario When Few Blocks Change: Version 1

Day	Script Effects	Flash Recovery Area Contents After Script
Sun Feb 1	<ol style="list-style-type: none"> 1. Attempt incremental update, which has no effect because there is no level 0 backup. 2. Create level 0 backup. 	Level 0 incremental image copy backup, with SCN as of Sun Feb 1
Mon Feb 2	<ol style="list-style-type: none"> 1. Attempt incremental update, which has no effect because there is no level 1 incremental. 2. Back up level 1 incremental. 	Level 0 incremental image copy backup with SCN as of Sun Feb 1, level 1 incremental backup containing changes from Feb 1 through Feb. 2, archived logs from Feb 1 through today
Feb 3 and after	<ol style="list-style-type: none"> 1. Perform incremental update of level 0, rolling it forward to the previous day. 2. Back up level 1 incremental. 	Level 0 incremental image copy backup rolled forward to previous day's level 1 SCN, level 1 incremental backup with changes for previous 24 hours, archived logs from February 1 through today. Old incremental backups and archived logs not useful for recovery of the rolled-forward level 0 backup are automatically deleted when more free space is needed in the flash recovery area.

To alter the example slightly: if you can size the flash recovery area to hold n days worth of archived logs and incremental backups (where $n > 1$), then you can alter the RECOVER line to RECOVER COPY UNTIL TIME 'SYSDATE- n '. For example, if the flash recovery area is large enough to hold three days of incremental backups, then you can change the script as follows:

```
RECOVER COPY OF DATABASE TAG "whole_db_copy" UNTIL TIME 'SYSDATE-3';
# Make an incremental backup of the database to the flash recovery area.
BACKUP INCREMENTAL LEVEL 1
    FOR RECOVER OF COPY WITH TAG "whole_db_copy"
    DATABASE;
```

Every day that you run the script, RMAN rolls forward the whole database copy to an SCN three days before the current time. Hence, the disk quota rules will preserve archived logs and incremental backups created *after* SYSDATE-3 (because they are needed to recover the database to an SCN within the last three days).

The following table lists the set of files expected to be in the flash recovery area each day after the daily script for this strategy is run.

Table A-2 Backup Timeline, Scenario When Few Blocks Change: Three Days of Backups

Day	Script Effects	Contents of Flash Recovery Area
Sun Feb 1	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which has no effect because there is no level 0 backup old enough to roll forward to "SYSDATE-3" 2. Create level 0 backup 	Level 0 backup from Feb 1
Mon Feb 2	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which has no effect because there is no level 0 backup old enough to roll forward to "SYSDATE-3" 2. Create level 1 incremental backup for Feb 2 changes 	Level 0 backup from Feb 1, level 1 backup with changes from Feb 2, archived logs from Feb 1 through today
Tue Feb 3	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which has no effect because there is no level 0 backup old enough to roll forward to "SYSDATE-3" 2. Create level 1 incremental backup for Feb 3 changes 	Level 0 backup, level 1 incremental backups from Feb 2 through Feb 3, archived logs from Feb 1 through Feb 3
Wed Feb 4	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which has no effect because there is no level 0 backup old enough to roll forward to "SYSDATE-3" 2. Create level 1 incremental backup for Feb 4 changes 	Level 0 backup, level 1 backups from Feb 2 through today, archived logs from Feb 1 through today
Thur Feb 5	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which rolls level 0 backup forward to Feb 2 2. Create level 1 incremental backup for Feb 5 changes. 	Level 0 backup rolled forward to Feb 2, level 1 backups from Feb 3 through today, archived logs from Feb 2 through today
After Feb 5	<ol style="list-style-type: none"> 1. Attempt incremental update of level 0 backup, which rolls level 0 backup to Feb 3 2. Create level 1 incremental backup for the changes for that day 	<p>Level 0 backup rolled forward to level 1 backups for the last three days, archived logs from Feb 1 through today.</p> <p>Old incremental backups and archived logs not useful for recovery of the rolled-forward level 0 backup are automatically deleted when more space is needed in the flash recovery area.</p>

Backup Scripts When Blocks Change Frequently

In this scenario, typical of a CRM environment, many or most of the data blocks are updated over the course of a week.

If you use a retention policy of redundancy X , and a backup script executed once each Y days, the minimum disk quota required is determined by the following formula:

$$\text{Disk Quota} = \text{Size of } X \text{ copies of the database} + \text{size of } Y \text{ days of archived redo logs}$$

If you use a retention policy of a recovery window of X days, and the backup script is executed once each Y days, then if $X \geq Y$, then disk quota is determined by the following formula:

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \text{size of } ((X * \text{ceil}(X/Y)) + 1) \text{ days of archived redo logs}$$

where $\text{ceil}(X/Y)$ is the smallest integer greater than or equal to X/Y .

If $X < Y$, the disk quota is the same as when $X=Y$, that is,

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \text{size of } (Y + 1) \text{ days of archived redo logs}$$

For this example, assume that the retention policy is `REDUNDANCY 1`, and you are executing the backup script once each week. The minimum disk quota required for this scenario is as follows:

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \text{size of 8 days of archived logs}$$

This scenario requires no initial setup script. The weekly backup script is shown here:

```
# Execute once a week
# Make a full backup of the database to the flash recovery area.
BACKUP DATABASE TAG "weekly_full_bkup";
```

As shown in the following table, the disk quota needs only to keep one level 0 copy of the database and one week's worth of archived logs.

Table A-3 Backup Timeline for Scenario When Most Blocks Change

Day	Action	Contents of Flash Recovery Area
Sun Feb 1	Back up level 0.	Full backup
Sun Feb 8	Back up level 0.	Full backup from today, archived logs from Feb 1 through Feb 8
Each subsequent Sunday	Back up level 0.	Full backup from this Sunday, archived logs from the previous Sunday through this Sunday

Backup Scripts When a Moderate Number of Blocks Change Weekly

Use this strategy when roughly half the data blocks change weekly, or when the number of data blocks that change varies widely from week to week. The scripts in this strategy are "all purpose" scripts that use incrementally updated backups, starting with an image copy of the database, taking incremental level 1 backups at regular intervals, and rolling forward the existing level 0 copy.

The formula for determining space required for the flash recovery area depends upon the retention policy. For a retention policy of REDUNDANCY X and a backup script that runs once every Y days, the disk space required is:

$$\text{Disk Quota} = \text{Size of } X \text{ copies of the database} + \\ \text{Size of 1 incremental backup} + \\ \text{Size of } Y+1 \text{ days of archived redo logs}$$

For a recovery window of X days and a backup script executed once each Y days, if $X \geq Y$ then the space required is:

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \\ \text{Size of } \text{floor}(X/Y)+1 \text{ incremental backups} + \\ \text{Size of } (X * \text{ceil}(X/Y)) + 1 \text{ days of archived logs}$$

where $\text{ceil}(X/Y)$ is the smallest integer greater than or equal to X/Y and $\text{floor}(X/Y)$ is the largest integer less than or equal to X/Y .

If $X < Y$ then the space required is the same as when $X=Y$:

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \\ \text{Size of 2 incremental backups} + \\ \text{Size of } Y+1 \text{ days of archived logs}$$

For the example that follows, assume that the retention policy is REDUNDANCY 1 and the backup script is run once each week.

Initial Setup The minimum disk quota for such a scenario is

$$\text{Disk Quota} = \text{Size of 1 copy of the database} + \\ \text{Size of 1 incremental backup} + \\ \text{Size of 8 days of archived logs}$$

The first week, the `BACKUP . . . FOR RECOVER OF COPY` command makes a level 0 incremental backup of the database to the flash recovery area. This backup will be the basis for the weekly strategy. Each subsequent week, this copy is rolled forward to the last incremental backup checkpoint time. In this way, you create an on-disk recovery window of one week.

Weekly Script After the initial setup, the weekly backup script (which, for example, you could run every Sunday night) should look like the following:

```
# Execute once a week
# Roll forward the whole copy of the database to last incremental backup SCN
RECOVER COPY OF DATABASE TAG "whole_db_copy";
# Make an incremental backup of the database to the flash recovery area.
BACKUP INCREMENTAL LEVEL 1
    FOR RECOVER OF COPY WITH TAG "whole_db_copy"
    DATABASE;
```

The following table illustrates how the contents of the flash recovery area change with each run of the backup script, always keeping enough archived logs and backups on hand to maintain the seven day recovery window.

Table A–4 Backup Timeline for Scenario When Moderate Number of Blocks Change

Day	Action	Contents of Flash Recovery Area After Run
Sun Feb 1	<ol style="list-style-type: none"> 1. Attempt roll-forward of incremental level 0 copy of database with tag "whole_db_copy". Fail, because there is no level 0 copy. 2. Create level 0 incremental backup. 	Level 0 backup
Sun Feb 8	<ol style="list-style-type: none"> 1. Attempt roll-forward of incremental level 0 copy of database with tag "whole_db_copy" using previous week's incremental level 1 backup. Fail, because there is no level 1 backup from the previous week. 2. Create level 1 incremental backup. 	Level 0 backup from Sunday Feb 1 (not rolled forward because no level 1 exists), archived logs from Feb 1 through Feb 8
All future Sundays	<ol style="list-style-type: none"> 1. Roll forward level 0 copy of database to SCN of most recent incremental level 1 backup. 2. Create incremental level 1 backup with changes since last incremental level 1 backup SCN. 	<p>Level 0 backup rolled forward to SCN of previous Sunday's incremental level 1, archived logs from previous Sunday through today.</p> <p>Archived logs from prior to previous Sunday and the previous week's incremental level 1 backup are obsolete after the roll-forward of the level 0 backup, because there is no other backup to which they could be applied.</p> <p>Obsolete files may remain in the flash recovery area until the need for disk space causes the database to delete them or until the <code>DELETE OBSOLETE</code> command is used.</p>

Backing Up to the Flash Recovery Area and to Tape: Basic Scenarios

This scenario describes how to use RMAN to make backups to the flash recovery area and then later move them to tape.

The scripts in this section use the flash recovery area as the log archiving destination and the destination for all disk backups. Therefore, whenever there is no space to create archived logs or backups in the flash recovery area, Oracle automatically deletes archived logs and backups that are obsolete or that have been moved to tape.

By moving backup files and archived redo logs to tape, more space becomes available in the flash recovery area for new files. The formulas for the flash recovery area disk quota are suggested minimums, and depend upon the availability of space on tape to satisfy the retention policy. You can, however, increase your on-disk recovery window and reduce the need to restore archived redo logs and backups from tape during a database restore-and-recovery scenario, by allocating a larger flash recovery area disk quota than these formulas specify.

Configuring the RMAN Environment for Disk and Tape Backups

Use RMAN to configure the retention policy, backup optimization, and the control file autobackup. For example:

```
CONNECT TARGET SYS/oracle@trgt;
# Recovery window of 15 days ensures that the database is recoverable within 7
# days using backups on disk and tape
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 15 DAYS;
CONFIGURE BACKUP OPTIMIZATION ON;
# Because you do not use a recovery catalog, enable the autobackup feature
CONFIGURE CONTROLFILE AUTOBACKUP ON;
CONFIGURE DEVICE TYPE sbt PARMS='...' PARALLELISM 1; # PARMS are vendor-specific
```

Writing Backup Scripts for Disk and Tape Scenarios

As in the disk-only scenarios, the backup scripts in this section are categorized based on database workload.

Backup Scripts When Few Data Blocks Change

In this scenario, relatively few data blocks change frequently, so daily level 1 incremental backups will typically be small. The goal is to keep one level 0 incremental backup on disk, incrementally updated each day using a level 1 incremental backup, and then move all other files onto tape. This keeps flash recovery area usage to a minimum.

Initial Setup The only required setup is to create a flash recovery area with the required disk quota, and set the flash recovery area as a redo log archiving destination.

The formula for determining your disk quota depends upon the backup retention policy. It is the same, however, whether your retention policy is based on redundancy or recovery window. Use the following formula:

```
Disk Quota = Size of 1 copy of the database
             + size of 1 day's level 1 incremental backup
             + size of (Y+1) days of archived logs
```

where Y is the number of days that elapse between execution of `BACKUP RECOVERY AREA` in your backup scripts.

For this example, there is one backup script, executed every day, which creates a new incremental level 1 backup with that day's changes, rolls forward the level 0 backup using the level 1 backup from the previous day, and backs up all flash recovery area files to tape. Since the script backs up the flash recovery area to tape daily, the flash recovery area must be large enough to hold a copy of the database, the daily level 1 incremental backup, and two days' worth of archived redo logs.

The on-disk recovery window is one day. To recover to a point farther back than one day, RMAN must restore backups from tape.

Daily Script After the initial setup, a typical daily RMAN backup script would look like the following:

```
# Execute each day of the week
# Roll forward the copy to most recent incremental backup SCN, which will
# be yesterday's incremental level 1 backup
RECOVER COPY OF DATABASE WITH TAG "daily_backup";
# Take incremental backups to flash recovery area (using default disk channel)
BACKUP INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG "daily_backup" DATABASE;
# Back up flash recovery area to tape
```

```

BACKUP RECOVERY AREA;
# delete obsolete backups on tape
DELETE OBSOLETE DEVICE TYPE sbt;
    
```

The following table describes how the contents of the flash recovery area and tape change as this script is run each day. The script rolls forward the level 0 database copy tagged as `daily_backup` to the SCN of the preceding day's incremental level 1 backup, then creates a new level 1 incremental backup with the previous day's changes. For example, when you run the script on Thursday, RMAN rolls forward the level 0 backup to the SCN of the incremental backup on Wednesday.

Table A-5 Backup Timeline for Scenario When Few Blocks Change

Day	Action	Contents of Flash Recovery Area and SBT
Sun Feb 1	<ol style="list-style-type: none"> 1. Attempt roll-forward of level 0 backup; fail because there is no level 0 backup yet. 2. Create level 0 incremental backup to be rolled forward on future days. 	Level 0 incremental backup. Any other flash recovery area files such as archived redo logs are copied to tape and may be deleted from flash recovery area when space is needed.
Mon Feb 2	<ol style="list-style-type: none"> 1. Attempt roll-forward of level 0 backup; fail because there is no level 1 backup yet. 2. Create level 1 incremental backup, containing changes during Feb 2. 3. Back up flash recovery area to sbt. 	Level 0 backup from Feb 1, level 1 incremental backup containing changes from Feb 2. Any other flash recovery area files such as archived redo logs are copied to tape and may be deleted from flash recovery area when space is needed. Any obsolete files stored on tape are deleted.
All future days	<ol style="list-style-type: none"> 1. Roll forward incremental level 0 backup to SCN of previous level 1 incremental backup. 2. Create level 1 incremental backup containing changes from previous day. 3. Back up flash recovery area to sbt. 	Level 0 backup rolled forward to previous day, level 1 incremental backup containing changes from this day Any other flash recovery area files such as archived redo logs are copied to tape and may be deleted from flash recovery area when space is needed. Any obsolete files stored on tape are deleted.

Backup Scripts When Many Blocks Change

In this scenario, many or most of the data blocks are updated over the course of a week, as in a CRM environment. A strategy using daily incremental backups would not be recommended because the size of the incremental backups could be quite large and is hard to predict.

A better strategy in this case is to back up the database to tape weekly and the archived logs to tape every day, and to use a recovery window-based retention policy.

The elements of the strategy are as follows:

- Redo log files are archived in the flash recovery area.
- A full backup of the database is taken to the flash recovery area once each week.
- Once each day, any backup files (including archived redo logs) in the flash recovery area that is not currently stored on tape is backed up to tape using the

BACKUP RECOVERY AREA command, and any obsolete backups on tape are deleted.

If you use a recovery window-based retention policy, all backups required to perform point-in-time recovery within the window are retained as long as they are needed. This provides the necessary protection for your data, even with limited use of disk space.

Initial Setup Create your flash recovery area with the required disk quota and set up the flash recovery area as a redo log archiving destination. The formula for determining your disk quota depends upon the frequency of backups of the flash recovery area to tape, as follows:

$$\text{Disk Quota} = \text{Size of 1 copies of the database} \\ + \text{size of (Y+1) days of archived logs}$$

where Y is the number of days that elapse between execution of BACKUP RECOVERY AREA in your backup scripts.

For this example, assume that you take a full database backup once each week, and back up the flash recovery area to tape on all seven days of the week.

There are two scripts in this strategy. Use the first script at the beginning of each week (say, on Sunday) to create a full database backup, and the other script on the remaining days of the week (Monday through Saturday) to back up the flash recovery area contents (the archived redo logs for the day) to tape.

Note that both of the scripts in this strategy include the BACKUP RECOVERY AREA command, so that command is executed every day. For this example, Y=1, so the disk quota is the size of one copy of the database plus two days of archived redo logs.

Weekly Scripts This RMAN backup script is executed once each week, on Sunday:

```
# Execute only once a week
# Take copy of database to flash recovery area
BACKUP AS COPY DATABASE;
# Take backup of flash recovery area to tape
BACKUP RECOVERY AREA;
# Delete obsolete backups on tape
DELETE OBSOLETE DEVICE TYPE sbt;
EXIT;
```

Daily Script The RMAN script to be executed each day (Monday through Saturday) would look like the following:

```
# Execute 6 days/wk
# Take backup of flash recovery area to tape
BACKUP RECOVERY AREA;
# delete obsolete backups on tape
DELETE OBSOLETE DEVICE TPE sbt;
```

Because the entire flash recovery area is backed up to tape every day, the database may delete backups and archived redo log files from the flash recovery area whenever space is needed for new files. Between backups it may be difficult to predict exactly which files will still be in the flash recovery area, because files are deleted whenever space is needed.

Backup Scripts When Blocks Change Moderately

The following strategy is based on full database backups taken at regular intervals. It is useful when about half of the data blocks change during the interval between full

database backups, or when the number of changed data blocks varies widely between full database backups.

The strategy is based on an incrementally updated backups-based full backup of the database, rolled forward once each week. This requires a level 0 disk copy incremental backup of the database, and a level 1 incremental created each Sunday with the previous seven days of datafile changes. After the roll-forward, the contents of the flash recovery area are backed up to tape, so the incremental level 0 copy of the database and the level 1 weekly incremental backup are available both on disk and tape.

Each day, archived redo log files accumulate in the flash recovery area. Each time the daily script runs, those redo log files are backed up to tape. After that, they may be deleted from disk if space is needed in the flash recovery area.

Initial Setup The minimum flash recovery area disk quota for this scenario is calculated with the following formula:

```
Disk Quota = Size of 1 copy of the database
            + Size of 1 level 1 incremental backup with Y days of changes
            + Size of (Y+1) days of archived redo logs
```

where Y is the number of days between backing up the flash recovery area to tape.

For this example, Y=7, so the disk quota must be at least the size of one copy of the database, one seven-day level 1 incremental backup, and eight days' worth of archived redo logs.

Weekly Script The script that implement the Sunday backup for this strategy weekly parts of the backup is as follows:

```
# Execute once a week
# Roll forward the whole copy of the database to last incremental backup SCN
RECOVER COPY OF DATABASE WITH TAG "whole_db_copy";
# Make an incremental backup of the database to the flash recovery area.
BACKUP DEVICE TYPE DISK INCREMENTAL LEVEL 1
      FOR RECOVER OF COPY WITH TAG "whole_db_copy"
      DATABASE;
# Back up flash recovery area to tape
BACKUP RECOVERY AREA;
# delete obsolete backups on tape
DELETE OBSOLETE DEVICE TYPE sbt;
```

Daily Script The daily script for this strategy is shown here:

```
# Execute 6 days/wk
# Back up flash recovery area to tape
BACKUP RECOVERY AREA;
# delete obsolete backups on tape
DELETE OBSOLETE DEVICE TYPE sbt;
```

The following table illustrates how the scripts maintain the recovery window. Assume for the purposes of this example that the week begins on Sunday.

Table A-6 Backup Timeline for Scenario When Moderate Number of Blocks Change

Day	Script	Action	Contents of Flash Recovery Area and Tape After Script
Week 1, Sunday	Weekly	<ol style="list-style-type: none"> 1. Attempt RECOVER COPY OF DATABASE command. This command has no effect, because there is no level 0 backup of the database yet. 2. Create level 0 backup as result of the BACKUP... FOR RECOVER OF COPY command. 3. Back up all files in the flash recovery area to tape. 4. Delete obsolete backups from tape. 	<p>Flash recovery area contains the level 0 incremental backup of the database, to be rolled forward each week.</p> <p>Tape contains a backup of the level 0 incremental backup of the database.</p>
Week 1, Monday-Saturday	Daily	<ol style="list-style-type: none"> 1. Back up flash recovery area to tape. 2. Delete obsolete backups from tape. 	<p>Flash recovery area contains archived redo logs for the previous day, and the incremental level 0 database backup.</p> <p>Tape contains a backup of the level 0 incremental backup, and all redo logs for all days since Sunday of week 1.</p> <p>No backups are obsolete.</p>
Week 2, Sunday	Weekly	<ol style="list-style-type: none"> 1. Attempt RECOVER COPY OF DATABASE level 0 to most recent incremental SCN. This command has no effect, because there is no level 1 incremental backup to use in roll-forward. 2. Perform level 1 incremental backup. A level 1 incremental backup is created, including changes from week 1. 3. Back up flash recovery area to tape. 4. Delete obsolete backups from tape. 	<p>Flash recovery area contains incremental level 0 database backup at SCN of Sunday of week 1, level 1 incremental backup with changes from week 1.</p> <p>Tape contains level 0 incremental backup from Sunday of week 1, level 1 incremental backup with all changes from week 1, archived redo logs from week 1.</p> <p>No backups are obsolete.</p>
Week 2, Monday-Saturday	Daily	<ol style="list-style-type: none"> 1. Back up flash recovery area to tape. 2. Delete obsolete backups from tape. 	<p>Flash recovery area contains archived redo logs for the previous day, and the incremental level 0 database backup. May also contain other archived redo logs.</p> <p>Tape contains a backup of the level 0 incremental backup, and all redo logs for all days since Sunday of week 1.</p>

Table A-6 (Cont.) Backup Timeline for Scenario When Moderate Number of Blocks

Day	Script	Action	Contents of Flash Recovery Area and Tape After Script
Week 3, Sunday	Weekly	<ol style="list-style-type: none"> 1. Perform RECOVER COPY OF DATABASE level 0 to most recent incremental SCN. The level 0 incremental backup is rolled forward to the beginning of week 2. 2. Perform level 1 incremental backup. A level 1 incremental backup is created including changes from week 2. 3. Back up flash recovery area to tape. 4. Delete obsolete backups from tape. 	<p>Flash recovery area contains incremental level 0 database backup at SCN of Sunday from week 2, level 1 incremental backup with changes from week 2.</p> <p>Tape contains level 0 incremental backup rolled forward to SCN of Sunday of week 2, level 1 incremental backup with all changes from week 2, archived redo logs from week 2.</p> <p>Obsolete backups deleted from tape include archived redo logs from week 1, tape backup of level 0 incremental database backup from week 1.</p>

Backup Scripts When Not Enough Disk Space for a Database Backup

If the disk quota is not sufficient to keep a copy of database, then the flash recovery area should be used *only* as an archived log destination. The disk quota rules will automatically delete these logs from the flash recovery area when they are no longer needed (because they are obsolete or because they have been backed up to tape). Back up the archived logs to tape daily, to ensure that they can be deleted from the flash recovery area when space is needed there for other files.

For this strategy, the flash recovery area should be sized large enough to hold at least two days' worth of archived redo logs, plus one day's worth of incremental backup.

This strategy is based on level 0 and level 1 incremental backups (though it does not use incrementally updated backups). This strategy uses two scripts: one to be executed once a week (for example, on Sunday), and the other to be executed every day except for the day that the first script is executed (Monday through Saturday).

The script that runs once each week creates a level 0 incremental backup on tape, containing the full contents of the database.

The script that runs each day except the first day creates level 1 incremental backups containing the changes to the database each day.

Weekly Script Here is the script to run once at the beginning of each week

```
# Execute only once a week
# backup database to tape
BACKUP DEVICE TYPE sbt INCREMENTAL LEVEL 0 DATABASE;
# delete obsolete backups on tape
DELETE OBSOLETE DEVICE TYPE sbt;
# backup recovery file destination to tape
BACKUP RECOVERY AREA;
```

Daily Script This script executes each day of the week except for the first day, for example, Monday through Saturday:

```
# backup recovery file destination to tape
BACKUP RECOVERY AREA;
# Take incremental backups to flash recovery area
```

BACKUP DEVICE TYPE DISK INCREMENTAL LEVEL 1 DATABASE;

The following table illustrates how the scripts maintain the RMAN retention policy recovery window and how the disk quota is maintained. The recovery files that exist after the execution of the script each day also exist on tape, so Oracle can delete files from the flash recovery area as needed.

Table A-7 Backup Timeline for Limited Disk Space Scenario

Day	Script	Action	Contents of Tape After Script
Sun Feb 1	Once-a-week	<ol style="list-style-type: none"> 1. Back up level 0 to tape. 2. Delete obsolete from tape. 3. Back up flash recovery area to tape. 	Tape contains a level 0 backup of the whole database from February 1, and any archived redo logs.
Mon Feb 2 - Sat Feb 7	Daily	<ol style="list-style-type: none"> 1. Back up flash recovery area to tape. 2. Back up level 1 to disk. 	Tape contains a level 0 backup of the whole database from February 1, incremental level 1 backups for each day from Monday through the present day, and any archived redo logs for the whole week.
Sun Feb 8	Once-a-week	<ol style="list-style-type: none"> 1. Back up level 0 to tape. 2. Delete obsolete from tape. 3. Back up flash recovery area to tape. 	Tape contains a level 0 backup of the whole database from February 1, incremental level 1 backups for each day from Monday through the present day, and any archived redo logs for the whole week.

Glossary

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. After the LGWR process fills each online redo log with redo records, the archiver process copies the log to one or more redo log archiving destinations. This copy is the archived redo log. Note that RMAN does not distinguish between an original archived redo log and an image copy of an archived redo log; both are considered image copies.

ARCHIVELOG mode

The mode of the database in which Oracle copies filled online redo logs to disk. Specify the mode at database creation or with the ALTER DATABASE ARCHIVELOG statement.

See Also: [archived redo log](#), [NOARCHIVELOG mode](#)

archiving

The operation in which a filled online redo log file is copied to an offline log archiving destination. An offline copy of an online redo logs is called an [archived redo log](#). You must run the database in ARCHIVELOG mode to archive redo logs.

automatic channel allocation

The ability of RMAN to perform backup and restore tasks without requiring the use of the ALLOCATE CHANNEL command. You can use the CONFIGURE command to specify disk and tape channels. Then, you can issue commands such as BACKUP and RESTORE at the RMAN command prompt without manually allocating channels. RMAN uses whatever configured channels that it needs in order to execute the commands.

automatic undo management mode

A mode of the database in which undo data is stored in a dedicated [undo tablespace](#). The only undo management that you must perform is the creation of the undo tablespace. All other undo management is performed automatically.

auxiliary database

(1) A database created from target database backups with 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: [recovery set](#), [auxiliary set](#), [tablespace point-in-time recovery \(TSPITR\)](#)

auxiliary set

In TSPITR, the set of files that is not in the recovery set but which must be restored in the auxiliary database for the TSPITR operation to be successful.

See Also: [auxiliary database](#), [recovery set](#)

backup

(1) A backup copy of data, that is, a database, tablespace, table, datafile, control file, or archived redo log. Backups can be physical (at the database file level) or logical (at the database object level). Physical backups can be created by using RMAN to back up one or more datafiles, control files or archived redo log files. Logical backups can be created using one of the Oracle export utilities (Data Pump Export or Original Export).

(2) An RMAN command that creates a backup set, proxy copy, or disk-based image copy.

See Also: [copy](#), [backup set](#), [multiplexing](#), [RMAN](#)

backup, whole database

See [whole database backup](#)

backup and recovery

The set of concepts, procedures, and strategies involved in protecting the database against data loss due to media failure or users errors.

backup control file

A backup of the control file. You can back up the control file with the RMAN backup command or with the SQL statement `ALTER DATABASE BACKUP CONTROLFILE TO 'filename'`.

backup mode

The database mode (also called **hot backup mode**) initiated when you issue the `ALTER TABLESPACE . . . BEGIN BACKUP` or `ALTER DATABASE BEGIN BACKUP` command before taking an online backup. You take a tablespace out of backup mode when you issue the `ALTER TABLESPACE . . . END BACKUP` or `ALTER DATABASE END BACKUP` command.

You must use this command when you make a user-managed backup of datafiles in an online tablespace. RMAN does not require you to put the database in backup mode. In backup mode, updates to the database create more than the usual amount of redo. Each time a block in the buffer cache becomes dirty, Oracle must write an image of the changed block to the redo log file, in addition to recording the changes to the data.

See Also: [corrupt block](#), [online backup](#)

backup piece

The physical file format used to store RMAN backup sets.

See Also: [backup](#), [backup set](#), [RMAN](#)

backup retention policy

See [retention policy](#)

backup set

A backup of one or more datafiles, control files, SPFILEs and archived redo log files. Each backup set consists of one or more binary files called **backup pieces**. Backup

pieces are written in a proprietary format that can only be created or restored by RMAN.

Backup sets are produced by the RMAN BACKUP command. A backup set usually consists of only one backup piece. RMAN divides the contents of a backup set among multiple backup pieces only if you limit the backup piece size using the MAXPIECESIZE option of the ALLOCATE CHANNEL or CONFIGURE CHANNEL command.

See Also: [backup piece](#), [unused block compression](#), [multiplexing](#), [RMAN](#)

binary compression

A technique whereby RMAN applies a compression algorithm to data being backed up to backup sets.

block change tracking

A database option that causes Oracle to track datafile blocks affected by each database update. The tracking information is stored in a new type of file called the change tracking file. When block change tracking is enabled, RMAN uses the record of changed blocks from the change tracking file to improve incremental backup performance by only reading those blocks known to have changed, instead of reading datafiles in their entirety.

block media recovery

The recovery of specified blocks within a datafile with the Recovery Manager BLOCKRECOVER command. Block media recovery leaves the affected datafiles online and restores and recovers only the damaged or corrupted blocks.

channel

A connection between RMAN and the target database. Each allocated channel starts a new Oracle server session; the session then performs backup, restore, and recovery operations. A channel can either be a DISK channel (used to perform disk I/O) or an sbt channel (used to perform I/O through a third-party [media manager](#)).

See Also: [media manager](#), [target database](#)

checkpoint

A data structure that defines an SCN in the redo thread of a database. Checkpoints are recorded in the control file and each datafile header, and are a crucial element of recovery.

circular reuse records

Control file records containing information used by RMAN for backups and recovery operations. These records are arranged in a logical ring. When all available record slots are full, Oracle either expands the control file to make room for a new records or overwrites the oldest record. The CONTROL_FILE_RECORD_KEEP_TIME initialization parameter controls how many days records must be kept before they can be overwritten. The default for CONTROL_FILE_RECORD_KEEP_TIME is 7 days.

See Also: [noncircular reuse records](#)

closed backup

A backup of one or more database files taken while the database is closed. Typically, closed backups are whole database backups. If you closed the database cleanly, then all the files in the backup are consistent. Otherwise, the backups are inconsistent.

See Also: [consistent shutdown](#), [consistent backup](#)

cold backup

See [closed backup](#)

complete recovery

Recovery of one or more datafiles that applies all redo generated after the restored backup. Typically, you perform complete 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.

See Also: [incomplete recovery](#), [media recovery](#)

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 this backup for it to be consistent. (Note, however, that unless you apply the redo generated since the consistent backup was created, you lose all transactions since the time of the consistent backup.)

You can only take consistent backups after you have performed a [consistent shutdown](#) of the database. The database must not be re-opened until the backup has completed.

See Also: [fuzzy file](#), [inconsistent backup](#)

consistent shutdown

A database shut down with the `IMMEDIATE`, `TRASACTIONAL` or `NORMAL` options of the statement. A database shut down cleanly does not require recovery; it is already in a consistent state.

crash recovery

The automatic application of online redo records to a database after either a single-instance database crashes or all instances of an Oracle Real Applications Cluster configuration crash. Crash recovery only requires redo from the online logs: archived redo logs are not required.

See Also: [recover](#)

control file autobackup

The automatic backup of the current control file that RMAN makes in the situations:

- After every `BACKUP` command run at the RMAN prompt
- After a `BACKUP` command within a `RUN` block that is not followed by another `BACKUP` command

The control file autobackup has a default filename that allows RMAN to restore it even if the control file and recovery catalog are lost. You can override the default filename if desired.

copy

To back up a bit-for-bit image of an Oracle file (Oracle datafiles, control files, and archived redo logs) onto disk. You can copy in two ways:

- Using operating system utilities (for example, the UNIX `cp` or `dd`)
- Using the RMAN `BACKUP AS COPY` command

See Also: [backup](#)

corrupt block

An Oracle block that is not in a recognized Oracle format, or whose contents are not internally consistent. Typically, corruptions are caused by faulty hardware or operating system problems. Oracle identifies corrupt blocks as either logically corrupt (an Oracle internal error) or media corrupt (the block format is not correct).

You can repair a media corrupt block by recovering the block, or dropping the 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: [block media recovery](#)

crosscheck

A check to determine whether files on disk or in the media management catalog correspond to the data in the repository 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 RMAN repository can contain outdated information about backups. Run the CROSSCHECK command to perform a crosscheck. To determine whether you can restore a file, run VALIDATE BACKUPSET or RESTORE . . . VALIDATE.

See Also: [validation](#)

cumulative incremental backup

An [incremental backup](#) that backs up all the blocks changed since the most recent backup at level 0. When recovering with cumulative incremental backups, only the most recent cumulative incremental backup needs to be applied.

See Also: [differential incremental backup](#), [incremental backup](#)

current online redo log

The [online redo log](#) file in which the LGWR background process is currently logging redo records.

See Also: [redo log](#), [redo log group](#)

database checkpoint

The thread checkpoint that has the lowest SCN. All changes in all enabled threads prior to the database checkpoint are guaranteed to have been written to disk.

See Also: [checkpoint](#)

database identifier

See [DBID](#)

database point-in-time recovery (DBPITR)

The recovery of an entire database to a specified past target time, SCN, or log sequence number.

See Also: [incomplete recovery](#), [tablespace point-in-time recovery \(TSPITR\)](#)

datafile media recovery

The application of redo records to a restored datafile in order to roll it forward to a more current time. Unless you are doing [block media recovery](#), the datafile must be offline while being recovery.

DBID

An internal, uniquely generated number that differentiates databases. Oracle creates this number automatically when you create the database.

differential incremental backup

A type of [incremental backup](#) that backs up all blocks that have changed since the most recent backup at level 1 or level 0. For example, in a differential level 1 backup RMAN determines which level 1 or level 0 backup is most recent and then backs up all blocks changed since that backup. Differential backups are the default type of incremental backup. When recovering using differential incremental backups, RMAN must apply all differential incremental level 1 backups since the restored datafile backup.

See Also: [cumulative incremental backup](#), [incremental backup](#)

disk controller

A hardware component that is responsible for controlling one or more disk drives.

disk quota

A user-specified limit to the size of the [flash recovery area](#). When the disk quota is reached, Oracle automatically deletes files that are no longer needed.

duplicate database

A database created from target database backups using the RMAN duplicate command.

See Also: [auxiliary database](#)

export

The extraction of logical data (that is, not physical files) from a database into a binary file using one of the Oracle export utilities, such as Data Pump Export. You can then use a corresponding Oracle import utility to import the data into a database.

See Also: [logical backup](#)

flash recovery area

An optional disk location that you can use to store recovery-related files such as control file and online redo log copies, archived logs, flashback logs, and RMAN backups. Oracle and RMAN manage the files in the flash recovery area automatically. You can specify the [disk quota](#), which is the maximum size of the flash recovery area.

flashback logs

Oracle-generated logs used to perform flashback database operations. Oracle can only write flashback logs to the flash recovery area. They cannot be backed up to disk.

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.

full resynchronization

An RMAN operation that updates the [recovery catalog](#) with all changed metadata in the database's control file. You can initiate a full catalog [resynchronization](#) by issuing the RMAN command `RESYNC CATALOG`. (Note that it is rarely necessary to use

RESYNC CATALOG because RMAN automatically performs resynchronizations when needed.)

fuzzy file

A datafile that contains at least one block with an SCN greater than or equal to the checkpoint SCN in its header. For example, this situation occurs when Oracle updates a datafile that is in **backup mode**. A fuzzy file that is restored always requires recovery.

hot backup

See [online backup](#)

hot backup mode

See [backup mode](#)

image copy

A bit-for-bit **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 uses **unused block compression** and is in an RMAN-specific format)
- Generated with the RMAN BACKUP AS COPY command, an operating system command such as the UNIX cp, or by the Oracle archiver process

incarnation

A separate version of a database. The incarnation of the database changes when you open it with the RESETLOGS option, but you can recover backups from a prior incarnation so long as the necessary redo is available.

incomplete recovery

A synonym for [database point-in-time recovery \(DBPITR\)](#).

See Also: [complete recovery](#), [media recovery](#), [recover](#)

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 online database backups. You can also make an inconsistent backup by backing up datafiles while a database is closed, either:

- Immediately after the crash of an Oracle instance (or, in a RAC configuration, all instances)
- After shutting down the database using SHUTDOWN ABORT

Inconsistent backups are only useful if the database is in ARCHIVELOG mode and all archived redo logs created since the backup are available.

See Also: [consistent backup](#), [online 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 level 1 back up only blocks that have changed since previous incremental backups. Blocks that have not changed are not backed up. An incremental backup can be either a **differential incremental backup** or a **cumulative incremental backup**. A cumulative incremental backup backs up all blocks changed since the last level 1 incremental backup. A differential incremental backup backs up all blocks changed since the last level 0 or level 1 incremental backup.

instance failure

The termination of an Oracle instance due to a hardware failure, Oracle internal error, or SHUTDOWN ABORT statement. Crash or instance recovery is always required after an instance failure.

instance recovery

In a RAC configuration, the application of redo data to an open database by an instance when this instance discovers that another instance has crashed.

See Also: [recover](#)LogMiner

LogMiner

A utility that enables log files to be read, analyzed, and interpreted by means of SQL statements

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.

See Also: [log switch](#), [redo log](#)

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.

See Also: [redo log](#)

logical backup

A backup of database schema objects, such as tables. Logical backups are created and restored with the Oracle Data Pump Export utility or Original Export utility. You can restore objects from logical backups using the Oracle import utility that corresponds to the utility used to create the backup.

long-term backup

A backup that you want to exclude from a backup retention policy, but want to record in the recovery catalog. Typically, long-term backups are snapshots of the database that you may want to use in the future for report generation.

Mean Time To Recover (MTTR)

The time required to perform crash or media recovery on the database. A variety of factors influence MTTR for media recovery, including the speed of detection, the method used to perform media recovery, and the size of the database.

media failure

Damage to the disks containing any of the files used by Oracle, such as the datafiles, logfiles, or control file. When Oracle detects media failure, it takes the affected files offline.

See Also: [media recovery](#)

media manager

A third-party networked backup system that can be integrated with Recovery Manager so that database backups can be written directly to tertiary storage.

media recovery

The application of redo or incremental backups to a restored backup datafile or individual data block.

When performing media recovery, you can recover a database, tablespace, datafile, or set of blocks within a datafile. Media recovery can be either **complete recovery** (in which all changes in the redo logs are applied) or **incomplete recovery** (in which only changes up to a specified point in time are applied). Media recovery is only possible when the database is in ARCHIVELOG mode.

See Also: [block media recovery](#), [recover](#)

mirroring

Maintaining identical copies of data on one or more disks. 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. When mirroring files, Oracle writes once while the operating system writes to multiple disks; when **multiplexing** files, Oracle writes the same data to multiple files.

multiplexing

- **online redo logs**

The automated maintenance of more than one identical copy of the online redo log.

- **control file**

The automated maintenance of more than one identical copy of a database's control file.

- **backup set**

The RMAN technique of reading database files *simultaneously* from the disks and then writing the blocks to the *same* backup piece.

- **archived redo logs**

The Oracle archiver process is able to archive multiple copies of a redo log.

See Also: [mirroring](#)

NOARCHIVELOG mode

The mode of the database in which Oracle does not require filled online redo logs to be archived before they can be overwritten. Specify the mode at database creation or change it with the ALTER DATABASE NOARCHIVELOG command.

Note that running in NOARCHIVELOG mode severely limits the possibilities for recovery of lost or damaged data.

See Also: [archived redo log](#), [ARCHIVELOG mode](#)

noncircular reuse records

Control file records containing critical information needed by the Oracle database. These records are never automatically overwritten. Some examples of information in non-circular reuse records include the locations of datafiles and online redo logs.

See Also: [circular reuse records](#)

offline normal

When a tablespace is taken offline normal, it is taken offline using the `ALTER TABLESPACE . . . OFFLINE NORMAL` statement. The datafiles in the tablespace are checkpointed and do not require recovery before being brought online. If a tablespace is not taken offline normal, then its datafiles must be recovered before being brought online.

online backup

A backup of one or more datafiles taken while a database is open and the datafiles are online. When you make a user-managed backup while the database is open, you must put the tablespaces in **backup mode** by issuing an `ALTER TABLESPACE BEGIN BACKUP` command. (You can also use `ALTER DATABASE BEGIN BACKUP` to put all tablespaces in your database into backup mode in one step.)

Note that you should not put tablespaces in backup mode when performing backups with RMAN.

online redo log

The online redo log is a set of two or more files that record all changes made to the database. 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 each filled online redo log file must be copied to one or more archiving locations before LGWR can overwrite them.

See Also: [archived redo log](#)

online redo log group

The Oracle online redo log consists of two or more online redo log groups. Each group contains one or more identical online redo log members. An **online redo log member** is a physical file containing the redo records.

online redo log member

A physical online redo log file within an **online redo log group**. Each log group must have one or more members. Each member of a group is identical.

operating system backup

See [user-managed backup](#)

operating system backup and recovery

See [user-managed backup and recovery](#)

Oracle Flashback Database

The return of the whole database to a prior consistent SCN by means of the RMAN FLASHBACK command or SQL*Plus FLASHBACK statement. A database flashback is different from traditional media recovery because it does not involve the restore of physical files, instead restoring your current datafiles to past states using saved images of changed data blocks. The flashback database process uses **flashback logs** and archived redo logs.

Oracle-managed file

A database file managed by the Oracle Managed Files feature.

Oracle Managed Files (OMF)

A feature of the Oracle database which manages the creation, naming and deletion of Oracle database files within dedicated areas of disk, to minimize the need for DBAs to concern themselves with such specifics.

orphaned backups

Backups that belong to incarnations of the database that are not direct ancestors of the current incarnation, or that were created after the SCN where the ancestor incarnation branched toward the current incarnation. Such backups cannot be used in the current incarnation.

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 with the RECOVERY_PARALLELISM initialization parameter or options to the SQL*Plus RECOVER command.

parallelization

Allocating multiple channels for RMAN backup and recovery operations.

partial resynchronization

A type of **resynchronization** in which RMAN transfers data about archived logs, backup sets, and datafile copies from the target control file to the **recovery catalog**.

password file

A file created by the ORAPWD command, and required if you wish to connect using the SYSDBA or SYSOPER roles over a network. For details on password files, see the *Oracle Database Administrator's Guide*.

physical schema

The datafiles, control files, and redo logs in a database at a given time. Issue the RMAN REPORT SCHEMA command to obtain a list of tablespaces and datafiles.

point-in-time recovery

The incomplete recovery of database files to a noncurrent time. Time-based recovery is also known as **incomplete recovery** and **time-based recovery**.

See Also: [incomplete recovery](#), [incomplete recovery](#), [media recovery](#), [recover](#)

proxy copy

A backup in which the **media manager** manages the transfer of data between the media storage device and disk during RMAN backup and restore operations.

recover

To recover a database file or a database is typically to perform media recovery, crash recovery or instance recovery. Can also be used generically, as in "recover your data," to refer to reconstructing or re-creating lost data by any means.

See Also: [complete recovery](#), [incomplete recovery](#), [instance recovery](#), [crash recovery](#), [media recovery](#).

recovery

When used to refer to a database file or a database, 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.

See Also: [complete recovery](#), [incomplete recovery](#)

recovery catalog

A set of Oracle tables and views used by RMAN to store RMAN repository information about one or more Oracle databases. RMAN uses this data to manage the backup, restore, and recovery of Oracle databases.

Use of a recovery catalog is optional. The primary storage for RMAN repository information for a database is always in the control file of the target database. A recovery catalog is periodically updated with RMAN repository data from the control file. In the event of the loss of your control file, the recovery catalog can provide most or all of the lost information required for restore and recovery of your database. The recovery catalog can also store records of long-term backups and RMAN stored scripts for use with target databases.

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 the target database.

Recovery Manager (RMAN)

The primary utility for physical backup and recovery of Oracle databases. RMAN keeps records of Oracle databases in its own structure called an RMAN repository, manages storage of backups, validates backups. 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 third-party media management software 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 [tablespace point-in-time recovery \(TSPITR\)](#). After TSPITR, all database objects in the recovery set have been recovered to the same point in time.

See Also: [auxiliary set](#)

recovery window

A recovery window is one type of RMAN backup retention policy, in which the DBA specifies a period of time and RMAN ensures retention of backups and archived redo

logs required for point-in-time recovery to any time during the recovery window. The interval always ends with the current time and extends back in time for the number of days specified by the user.

For example, if the retention policy is set for a recovery window of seven days, and the current time is 11:00 AM on Tuesday, RMAN retains the backups required to allow point-in-time recovery back to 11:00 AM on the previous Tuesday.

See Also: [retention policy](#)

redo log

A redo log can be either an [online redo log](#) or an [archived redo log](#). The online redo log is a set of two or more redo log groups that records all changes made to Oracle datafiles and control files. An archived redo log is a copy of an online redo log that has been written to an offline destination.

See Also: [archived redo log](#), [online redo log](#)

redo log group

Each online redo log member (which corresponds to an online redo log file) belongs to a redo log group. Redo log groups contain one or more members. A redo log group with more than one member is called a multiplexed redo log group. The contents of all members of a redo log group are identical.

redo thread

The redo generated by an instance. If the database runs in a single instance configuration, then the database has only one thread of redo.

redundancy set

A set of backups enabling you to recover from the failure or loss of any Oracle database file.

registration

In RMAN, the execution of a REGISTER DATABASE command in order to record the existence of a target database in the recovery catalog. A target database is uniquely identified in the catalog by its DBID. You can register more than one database in the same catalog, and also register the same database in multiple catalogs.

See Also: [DBID](#)

RMAN repository

The record of RMAN metadata about backup and recovery operations on the target database. The authoritative copy of the RMAN repository is always stored in the control file of the target database. A [recovery catalog](#) can also be used for longer-term storage of the RMAN repository, and can serve as an alternate source of RMAN repository data if the control file of your database is lost.

See Also: [recovery catalog](#), [recovery catalog database](#), [resynchronization](#)

RESETLOGS

A method for opening a database that archives any current online redo logs (if using ARCHIVELOG mode), resets the log sequence number to 1, and clears the online redo logs. An OPEN RESETLOGS operation begins a new database incarnation. The starting SCN for the new incarnation, sometimes called the RESETLOGS SCN, is the incomplete recovery SCN of the media recovery preceding the OPEN RESETLOGS, plus one.

An OPEN RESETLOGS operation is required after incomplete recovery or recovery with a backup control file.

An OPEN RESETLOGS operation does not affect the recoverability of the database. Backups from before the OPEN RESETLOGS operation remain valid and can be used along with backups taken after the OPEN RESETLOGS operation to repair any damage to the database.

restore

The replacement of a lost or damaged file with a backup. You can restore files either with commands such as UNIX cp or the RMAN RESTORE command.

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. A **partial resynchronization** transfers information to the recovery catalog about archived redo logs, backup sets and datafile copies. RMAN performs resynchronizations automatically when needed.

retention policy

A user-defined policy for determining how long backups and archived logs need to be retained for media recovery. You can define a retention policy in terms of backup redundancy or a **recovery window**. RMAN retains the datafile backups required to satisfy the current retention policy, and any archived redo logs required for complete recovery of those datafile backups.

There should be a glossary entry for REDUNDANNCY.

RMAN

See [Recovery Manager \(RMAN\)](#)

rollback segments

Database segments that record the before-images of changes to the database.

rolling back

The use of rollback segments to undo uncommitted changes applied to the database during the **rolling forward** stage of **recover**.

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: [recover](#), [rolling back](#)

SBT

System Backup to Tape. Most commonly used to specify a destination for RMAN commands used to back up to tape, such as "BACKUP DEVICE TYPE sbt DATABASE".

snapshot control file

A copy of a database's control file created in an operating system specific location by Recovery Manager. RMAN creates the snapshot control file so that it has a consistent version of a control file to use when either resynchronizing the recovery catalog or backing up the control file.

standby database

A copy of a production database that you can use for disaster protection.

stored script

A sequence of RMAN commands stored in the [recovery catalog](#).

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.

SYSTEM tablespace

The tablespace that contains the Oracle data dictionary for a database, which is the metadata that describes the complete contents of the database. The `SYSTEM` tablespace is unlike other tablespaces in that all datafiles contained in the tablespace must be online for Oracle to function. If a media failure affects one of the datafiles in `SYSTEM`, then you must mount the database and recover.

tablespace point-in-time recovery (TSPITR)

The recovery of one or more non-`SYSTEM` tablespaces to a noncurrent time. You can use either RMAN or user-managed methods to perform TSPITR.

target database

In RMAN, the database that you are backing up or restoring.

tempfile

A file that belongs to a temporary tablespace, and is created with the `TEMPFILE` option. Temporary tablespaces cannot contain permanent database objects such as tables, and are typically used for sorting. Because tempfiles cannot contain permanent objects, RMAN does not back them up. However, RMAN does keep track of the locations of tempfiles in the control file, and during recovery the tempfiles will be re-created as needed at those locations.

transportable tablespace

A feature that transports a set of tablespaces from one database to another, or from one database to itself. Transporting a tablespace into a database is like creating a tablespace with preloaded data.

undo tablespace

A dedicated tablespace that stores only undo information when the database is run in [automatic undo management mode](#).

unused block compression

A method by which RMAN reduces the size of backup sets containing datafile backups by skipping some datafile blocks that are not currently in use. More details about when blocks can be skipped are in *Oracle Database Backup and Recovery Reference*.

user-managed backup

A backups made using a non-RMAN method, for example, using an operating system utility. For example, you can make a user-managed backup by running the `cp` command on UNIX or the `copy` command on Windows. A user-managed backup is also called an [operating system backup](#).

user-managed backup and recovery

A backup and recovery strategy for an Oracle database that does not use RMAN. This term is equivalent to [operating system backup and recovery](#). You can back up and restore database files using operating system utilities (for example, the `cp` command in UNIX), and recover using the SQL*Plus `RECOVER` command.

validation

A test that checks whether a backup can be restored. RMAN scans the backups and looks at the checksum to verify that the contents can be successfully restored.

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](#)

A

alert log
 control file record messages, 8-2
 monitoring overwriting of control file records, 8-3

ALTER DATABASE statement
 RENAME FILE clause, 6-18

archived log files
 backing up, 4-10
 deleting after backup, 4-11
 with other backups, 4-11

archived redo logs
 backing up
 using RMAN, 4-10, 4-11
 cataloging, 8-14
 generated during backups, 4-10
 restoring using RMAN, 6-19

archiving modes, 2-7

autobackups
 generating, 4-9

availability
 of RMAN backups, 8-13

AVAILABLE option
 of CHANGE, 8-13

avoiding dangerous backup techniques, 2-12

B

BACKUP
 ARCHIVELOG, 4-10
 DELETE ALL, 4-11
 DELETE INPUT, 4-11
 FOR RECOVER OF COPY WITH TAG, 4-16
 PLUS ARCHIVELOG, 4-11

backup and recovery
 definition, 1-1

BACKUP command, 4-1
 CURRENT CONTROLFILE option, 4-9
 DELETE ALL INPUT option, 4-11
 DELETE INPUT option, 4-11

backup methods, 1-15
 comparison, 1-15
 feature comparison, 1-15

backup pieces
 definition, 4-3

backup scenarios
 disk-and-tape, A-7
 disk-only, A-1

backup sets
 testing restore of, 6-11
 unused space compression, 4-3

backups
 after structural changes to database, 2-11
 archived redo logs
 using RMAN, 4-10, 4-11
 automatic log switch during, 4-10
 availability, 8-13
 consistent
 making using RMAN, 4-7
 control file
 using RMAN, 4-9, 4-10
 control file autobackups, 4-9
 crosschecking, 8-4
 cumulative incremental, 4-14
 Data Pump Export utility, 2-12
 datafile
 using RMAN, 4-8
 Date Pump Export utility, 2-12
 definition, 1-1
 exempt from retention policy, 8-13
 expired
 deleting, 8-7
 frequency, 2-10
 generating reports for, 4-22, 4-28
 guidelines, 2-5 to 2-13
 Data Pump Export utility, 2-12
 frequency, 2-10
 often-used tablespaces, 2-11
 storing old backups, 2-10
 structural changes, 2-11
 unrecoverable objects, 2-11
 inconsistent
 making using RMAN, 4-7
 incremental, 4-12
 differential, 4-13
 logical
 definition, 1-1
 methods
 feature comparison, 1-15
 noncumulative incremental, 4-14
 offline, 2-10

- online, 2-10
- online redo logs, 2-12
- physical
 - definition, 1-1
- planning before database creation, 2-5
- records of, 2-12
- recovering pre-RESETLOGS, 7-24
- Recovery Manager, 4-1
- reporting objects needing backups, 4-29
- server parameter files, 4-10
- storing, 2-10
- tablespace, 2-11
 - using RMAN, 4-8
- techniques to avoid, 2-12
- testing RMAN, 4-21
- validating, 4-21
- whole database
 - using RMAN, 4-7
- backupsets
 - compressed, 4-6
- block change tracking, 4-19
 - enabling and disabling, 4-19
- block corruptions
 - stored in V\$DATABASE_BLOCK_
 - CORRUPTION, 4-21

C

- CATALOG command, 8-14
- CHANGE command, 8-4
 - AVAILABLE option, 8-13
 - UNAVAILABLE option, 8-13
 - UNCATALOG option, 8-17
- CHANGE... KEEP
 - with DELETE OBSOLETE, 8-9, 8-14
- change tracking, 4-19
 - disk space used for, 4-20
 - enabling and disabling, 4-19
 - moving the change tracking file, 4-20
- channels
 - multiple
 - crosschecking and deleting, 8-9
- character sets
 - setting for use with RMAN, 3-2
- commands, Recovery Manager
 - BACKUP CURRENT CONTROLFILE, 4-9
 - CATALOG, 8-14
 - CHANGE, 8-4
 - CROSSCHECK, 8-4
 - DELETE, 8-15
 - LIST, 4-22
 - REPORT, 4-28
 - NEED BACKUP option, 4-29
 - RESTORE, 6-6
- compressed backupsets, 4-6
- configuring
 - Recovery Manager
 - autobackups, 3-11
 - backup retention policies, 3-18
- consistent backups

- using RMAN, 4-7
- control file autobackups
 - after structural changes to database, 3-11
 - configuring, 3-11
- control file copies
 - backing up, 4-10
- control file records
 - overwriting, 8-2
- control files
 - automatic backups
 - configuring, 3-11
 - backing up
 - using RMAN, 4-10
 - backups
 - including within database backup, 4-9
 - using RMAN, 4-9
 - copies
 - backing up, 4-10
 - overview, 1-4
 - overwriting records, 8-2
 - restoring, 6-15
- CONTROL_FILE_RECORD_KEEP_TIME
 - initialization parameter, 8-2
 - preventing overwrite of RMAN records, 8-2
- CONTROL_FILES initialization parameter, 6-15
- copies
 - crosschecking, 8-4
- crash recovery, 1-8
 - after instance failure, 1-8
 - instance failure, 1-8
 - overview, 1-8
 - read-only tablespaces, 6-8
- CROSSCHECK command, 8-4
- crosschecking, 8-4
 - backups and copies, 8-4
 - on multiple channels, 8-9
- cumulative incremental backups, 4-14

D

- Data Pump Export utility
 - backups, 2-12
- data structures
 - involved in recovery, 1-3
- database connections
 - Recovery Manager
 - without a catalog, 3-5
 - SYSDBA required for RMAN, 3-6
 - types in RMAN, 3-6
- database point-in-time recovery, 7-19
 - time-based recovery, 7-19
- database point-in-time recovery (DBPITR)
 - definition, 7-1
- database schema
 - generating reports, 4-31
- database writer process (DBWn)
 - media failure, 1-14
- databases
 - modes of archiving, 2-7
- datafile copies

- backing up using RMAN, 4-8
- datafiles
 - backing up
 - using Recovery Manager, 4-8
 - read-only
 - recovery, 6-8
- DBPITR, 7-19
- DELETE command, 8-7, 8-15
 - EXPIRED option, 8-7
- DELETE OBSOLETE
 - with CHANGE... KEEP, 8-9, 8-14
- deleting
 - backups, 8-6, 8-7
 - on multiple channels, 8-9
 - copies, 8-6
- differential incremental backups, 4-13
- disconnecting
 - from Recovery Manager, 3-2
- disk failures, 1-2, 1-13
- disk usage
 - monitoring, 3-18

E

- environment variables
 - NLS_DATE_FORMAT, 3-2
 - NLS_LANG, 3-2
- EXIT command, 3-2
- exiting RMAN, 3-2
- expired backups
 - deleting, 8-7
- EXPIRED option
 - of DELETE, 8-7

F

- failures
 - described, 1-2, 1-13
 - instance
 - recovery from, 1-8
 - media, 1-2, 1-13
 - safeguards provided, 1-3
 - See also* recovery
- flash recovery area
 - backup scenarios using, A-1
 - backup scenarios using with tape, A-7
 - monitoring disk usage with V\$RECOVERY_FILE_DEST and V\$FLASH_RECOVERY_AREA_USAGE, 3-18
- Flashback Database
 - and flashback logs, 1-12
 - overview, 1-12
- flashback database
 - determining the flashback database
 - window, 5-11
 - enabling, 5-9
 - logs
 - rules for retention and deletion, 5-11
 - performance tuning, 5-12
 - requirements, 5-9

- space management, 5-11
 - estimating disk space requirement, 5-10
- Flashback Drop, 7-5
 - and recycle bin, 1-12
 - overview, 1-12
 - recycle bin, 7-6
- flashback drop
 - enabling and disabling, 7-7
- Flashback Query
 - overview, 1-11
- Flashback Table
 - overview, 1-12
- flashback table
 - using, 7-4, 7-5
- FLASHBACK TABLE statement, 7-4, 7-5
- Flashback Transaction Query
 - overview, 1-12
- flashback transaction query, 7-3
- flashback undrop
 - object names when in recycle bin, 7-7
 - querying recycle bin, 7-8
 - restoring objects, 7-10
- Flashback Version Query
 - overview, 1-11
- FORCE option
 - DELETE command, 8-8

G

- guaranteed restore point, 5-4
 - compared to storage snapshots, 5-4
 - requirements, 5-6
- guaranteed restore points
 - space usage in flash recovery area, 5-7
- guidelines
 - backups
 - Data Pump Export utility, 2-12
 - frequency, 2-10
 - often-used tablespaces, 2-11
 - storing old backups, 2-10
 - structural changes, 2-11
 - unrecoverable operations, 2-11

H

- hardware configuration
 - keeping records of, 2-12

I

- image copies
 - testing restore of, 6-11
- INCLUDE CURRENT CONTROLFILE option
 - BACKUP command, 4-9
- incomplete recovery
 - defined, 7-19
- inconsistent backups
 - using RMAN, 4-7
- incremental backups, 4-12
 - and block change tracking, 4-19
 - differential, 4-13

- incrementally updated, 4-16
- incrementally updated backups, 4-16
- initialization parameters
 - CONTROL_FILE_RECORD_KEEP_TIME, 8-2
 - CONTROL_FILES, 6-15
- instance recovery, 1-8
 - definition, 1-8
 - instance failure, 1-8
 - overview, 1-8
 - read-only tablespaces, 6-8
- instances
 - recovery of, 1-8
- I/O errors
 - ignoring during deletions, 8-8

K

- KEEP option
 - of BACKUP, 8-13

L

- level 0 incremental backups, 4-13
- LIST command, 4-22
- lists
 - backups and copies, 4-27
- log switches
 - recovery catalog records, 8-14
- logical backups, 1-1

M

- managing RMAN metadata, 8-1
- media failures
 - overview, 1-2, 1-13
- media recovery
 - overview, 1-6
- metadata
 - managing RMAN, 8-1
- modes
 - archive log, 2-7
- multiplexing
 - recovery and, 1-13

N

- NLS_DATE_FORMAT environment variable, 3-2
- NLS_LANG environment variable, 3-2
- noncumulative incremental backups, 4-14

O

- offline backups, 2-10
- online backups, 2-10
- online redo logs
 - backing up, 2-12
 - media failure, 1-13
 - multiplexed, 1-13
- Oracle Flashback Database
 - overview, 1-12
- Oracle Flashback Drop

- overview, 1-12
- Oracle Flashback Query
 - overview, 1-11
- Oracle Flashback Table
 - overview, 1-12
- Oracle Flashback Transaction Query
 - overview, 1-12
- Oracle Flashback Version Query
 - overview, 1-11
- overwriting control file records, 8-2

P

- performing backups after unrecoverable operations, 2-11
- physical backups, 1-1
- point-in-time recovery
 - performing
 - with current control file, 7-22

Q

- QUIT command, 3-2
- quitting RMAN, 3-2

R

- recovery
 - crash, 1-8
 - crash recovery, 1-8
 - read-only tablespaces, 6-8
 - data structures used in, 1-3
 - database point-in-time, 7-19
 - failures requiring, 1-2, 1-13
 - generic procedures, 6-6
 - instance, 1-8
 - instance recovery, 1-8
 - instance failure, 1-8
 - read-only tablespaces, 6-8
 - media, 1-6
 - media recovery
 - enabled or disabled, 2-7
 - point-in-time
 - time-based, 7-19
 - preparing for, 6-6
 - tablespace, 6-5
- recovery catalog
 - cataloging
 - O/S backups, 8-14
 - deleting backups, 8-6
 - deleting records, 8-7
 - log switch record, 8-14
 - updating
 - after operating system deletions, 8-17
- Recovery Manager
 - archived redo logs
 - backups, 4-10
 - backups, 4-1
 - archived redo logs, 4-10, 4-11
 - control file, 4-10
 - control file copies, 4-10

- control files, 4-9
- datafile, 4-8
- incremental, 4-12
- tablespace, 4-8
- testing, 4-21
- validating, 4-21
- whole database, 4-7
- commands
 - CATALOG, 8-14
 - CHANGE, 8-4
 - DELETE, 8-15
 - REPORT, 4-29
 - RESTORE, 6-6
- database character set, 3-2
- database connections, 3-6
 - SYSDBA required for target, 3-6
 - without a catalog, 3-5
- datafile copies
 - backing up, 4-8
- disconnecting from, 3-2
- incremental backups
 - cumulative, 4-14
 - differential, 4-13
 - level 0, 4-13
- lists, 4-22
- metadata, 8-1
- reports, 4-28
 - database schema, 4-31
 - objects needing a backup, 4-29
 - obsolete backups, 4-30
- restoring
 - archived redo logs, 6-19
- retention policies
 - configuring, 3-18
 - setting time parameters, 3-2
- RECOVERY WINDOW parameter
 - CONFIGURE command, 3-19
- recovery windows
 - configuring for retention policy, 3-19
- Recycle Bin
 - and Flashback Drop, 1-12, 7-6
- recycle bin
 - enabling and disabling, 7-7
 - renamed objects, 7-7
 - restoring objects from, 7-10
 - viewing, 7-8
- redo logs
 - archiving modes, 2-7
- RENAME DATABASE clause
 - ALTER DATABASE statement, 6-18
- REPORT command, 4-22, 4-28
 - NEED BACKUP option, 4-29
- reports, 4-22, 4-28
 - database schema, 4-31
 - objects needing a backup, 4-29
 - obsolete backups, 4-30
 - unrecoverable backups, 4-30
- RESTORE command, 6-6
- restore point
 - guaranteed, 5-4

- compared to storage snapshots, 5-4
 - normal, 5-3
 - requirements, 5-6
- restore points
 - using, 5-6
- restore validation, 6-11
- restoring
 - control files, 6-15
 - server parameter files, 6-15
 - testing, 6-11
- retention policies
 - configuring, 3-18
 - configuring for redundancy, 3-19
 - disabling, 3-20
 - making backups exempt, 8-13
 - recovery windows, 3-19
- rollback segments, 1-5

S

- scenarios, Recovery Manager
 - recovering pre-resetlogs backup, 7-24
- server parameter files
 - backups, 4-10
 - configuring autobackups, 3-11
 - restoring, 6-15
- software configuration
 - keeping records of, 2-12
- SYSDBA option
 - implicitly assumed for RMAN connect to target, 3-6

T

- tables
 - FLASHBACK TABLE statement, 7-4
 - flashback transaction query, 7-3
- tablespace backups
 - using RMAN, 4-8
- tablespaces
 - backing up, 2-11
 - frequency, 2-11
 - backups using RMAN, 4-8
 - recovering accessible
 - when database is open, 6-5
- testing RMAN
 - backups, 4-21
- time parameters
 - setting for Recovery Manager use, 3-2
- time-based recovery, 7-19

U

- UNAVAILABLE option
 - of CHANGE, 8-13
- UNCATALOG option
 - deleting repository records, 8-17
 - of CHANGE, 8-17
- undo blocks, 1-5
- undo tablespaces, 1-5
- unrecoverable operations

- performing backups after, 2-11
- unused block compression, 4-3
- unused space compression, 4-3
- user errors, 1-14

V

- V\$DATABASE_BLOCK_CORRUPTION view, 4-21
- V\$FLASH_RECOVERY_AREA_USAGE, 3-18
- V\$RECOVERY_FILE_DEST, 3-18
- validating
 - backups, 4-21
- validation of restore, 6-11

W

- whole database backups
 - using RMAN, 4-7