

Table of Contents

1. Oracle Database Server	1
1.1 Oracle Instance	1
1.1.1 Background Processes	1
1.1.2 Memory Structure	1
1.2 Oracle Database	1
1.3 Processes	2
1.3.1 User Processes	2
1.3.2 Server Processes	2
1.4 Physical Structure Files	3
1.5 Memory Structure	3
1.5.1 System Global Area.....	4
1.5.2 Program Global Area.....	5
2. System Global Area (SGA)	6
3. Result Cache	8
3.1 Result Cache.....	8
3.2 Implementing SQL Query Result Cache.....	8
3.3 SQL Query Result Cache	8
3.4 PL/SQL Function Result Cache	8
4. FLASH CACHE.....	9
4.1 When to Configure the Flash Cache.....	9
4.2 Assign Sizing the Flash Cache	9
4.3 Flash cache disabled	9
4.4 Flash cache enabled.....	9
4.5. Flash Cache Initialization Parameters	10
5. Background Processes.....	11
5.1 Mandatory Processes	11
5.1.1 System Monitor	11
5.1.2 Process Monitor	11
5.1.3 Log Writer	11
5.1.4 Database Writer	12
5.1.5. Checkpoint Process	13
5.1.6. Recoverer	13
5.1.7 Listener Registration Process	13
5.2 Optional Processes.....	14
5.2.1 Archiver	14
5.2.2 Recovery Writer.....	14
5.2.3 Lock Monitor	14

5.2.4 Lock Manager Daemon	15
5.2.5 Lock processes	15
5.2.6 Block Server Process	15
5.2.7 Queue Monitor.....	15
5.2.8 Event Monitor	15
5.2.9 Shared Server Processes	15
5.2.10 Memory Manager	16
5.2.11 Parallel Execution Slaves.....	16
5.2.12 Trace Writer	16
5.2.13 DMON.....	16
5.2.14 Dispatcher	16
5.2.15 MMON	16
5.2.16 Input/Output Slaves.....	16
5.2.17 Wakeup Monitor Process	17
5.2.18 Memory Monitor Light.....	17
5.2.19 Job Queue Processes	17
5.2.20 RBAL	17
5.2.21 ARBx.....	17
5.2.22 ASMB.....	17
5.2.23 Change Tracking Writer.....	17
5.2.24 Job Queue Monitoring	17
5.2.25 Auto BMR Background Process.....	17
5.2.26 Automatic Block Media Recovery Slave Pool Process	18
5.2.27 Database Capture Process	18
5.2.28 Database Resource Manager Process	18
5.2.29 Diagnostic Capture Process.....	18
5.2.30 EMON Coordinator Process.....	18
5.2.31 RAT Masking Slave Process	18
5.2.32 Space Management Coordinator Process	18
5.2.33 Virtual Scheduler for Resource Manager Process	18
5.2.34 Virtual Keeper of Time Process.....	18
5.2.35 ASM Cluster File System CSS Process	18
5.3. Other Oracle Database Background Processes.....	19
6. Startup & Shutdown Modes.....	21
6.1 Starting Up a Database.....	21
6.1.1 Instance Stages.....	21
6.1.2 Startup Command Options.....	22
6.2 Restricted Mode	22
6.3 Database Shutdown.....	23
6.3.1 Shutdown Normal.....	23

6.3.2 Shutdown Transactional	23
6.3.3 Shutdown Immediate.....	23
6.3.4 Shutdown Abort	24
7. Database Creation.....	25
7.1 Understanding Oracle Tablespaces:.....	25
8. Tablespace Management	29
8.1 Dictionary Managed Tablespace.....	29
8.2 Locally Managed Tablespace.....	30
8.2.1 Benefits of LMTS.....	30
8.2.2 Allocation Types in LMTS.....	31
8.2.3 Checking space availability in LMTS.....	31
8.2.4 LMTS with Automatic Segment Space Management.....	32
8.3 BIGFILE Tablespaces - (From 10g)	32
8.3.1 Creating a BIGFILE Tablespace.....	33
8.3.2 Querying Oracle 10g tablespace properties and features	33
8.3.3 Database-Level Default Permanent and Temporary Tablespaces.....	33
8.3.4 Altering Database-Level Permanent and Temporary Tablespaces	34
8.4 SYSAUX Tablespace - (From 10g)	34
8.5 Tablespace Compression- (New in 11g R2)	35
8.5.1 Compression for Table Data.....	35
8.5.2 OLTP Table Compression	35
8.5.3 BASIC TABLESPACE COMPRESSION	36
8.5.5 OLTP TABLESPACE COMPRESSION	36
8.6 Online Move Datafile (12c)	36
8.6.1 Moving Online Datafile	36
9. Temporary Tablespace.....	38
9.1 Default Temporary Tablespace	38
9.2 Creating Locally Managed Temporary Tablespace	39
9.3 Altering Locally Managed Temporary Tablespace	39
9.4 Storage Parameters in Locally Managed Tablespaces.....	39
9.5 Creating Dictionary-Managed Temporary Tablespace.....	40
9.6 Altering Dictionary-Managed Temporary Tablespace	40
9.7 Temporary Tablespace Groups - (From 10g)	40
9.8 Creating Temporary Tablespace Groups	40
9.9 Temporary Undo(12c)	41
10. User Management (Users, Privileges & Roles).....	42
10.1 Users.....	42
10.1.1 Database Schema.....	42
10.1.2 Schema Objects	42
10.1.3 Alter User Command.....	43

10.1.4 Drop User Command	43
10.1.5 Related Views	43
10.2 Privileges.....	44
10.2.1 System Privileges.....	44
10.2.2 Revoking System Privileges.....	44
10.2.3 Object Privileges	44
10.3 Roles	45
10.3.1 Facts About Roles	45
10.3.2 Role Benefits	45
10.3.3 Creating Roles.....	46
10.3.4 Altering Roles	46
10.3.5 Granting Roles	46
10.3.6 Authentication.....	46
11.1 Overview	48
11.2 Authentication by the Operating System	48
11.3 Authentication by the Network.....	48
11.3.1 Authentication Using SSL.....	48
11.3.2 Authentication Using Third-Party Services	49
11.4 Authentication by Oracle Database.....	50
11.4.1 Password Encryption While Connecting	51
11.4.2 Account Locking	51
11.4.3 Password Lifetime and Expiration	51
11.4.4 Password History.....	51
11.4.5 Password Complexity Verification	51
11.5 Single sign-on.....	52
11.5.1 Server-Based Single sign-on.....	52
11.5.2 Middle Tier Single Sign-On.....	52
12. Schema Management	53
12.1 DDL Wait Option (New in 11g)	53
12.2 Invisible Indexes (New in 11g)	53
12.3 Virtual Columns (New in 11g)	54
12.4 Virtual columns (New in 11g)	54
12.5 Read-Only Tables (New in 11g)	54
12.6 Invisible Columns (New in12c).....	55
12.7 In-Database Archiving (New in12c)	55
12.8 Oracle Data Masking (New in12c)	56
12.8.1 Data Redaction using DBMS_REDACT.....	56
12.9 Identity Columns (New in12c)	56
12.10.1 Restrictions.....	57
13. Security Enhancements.....	58

13.1 Secure Password Support	58
13.1.1 Configuring Case-Sensitive Passwords	58
13.1.2 Case Sensitivity and Upgrading	58
13.1.3 Case Sensitivity and Password Files	59
13.1.4 New Password Management Function	60
13.1.5 New Security-Related Initialization Parameters	60
13.2 Encrypting Tablespaces	61
13.2.1 Creating the Oracle Wallet	61
13.2.2 Creating an Encrypted Tablespace	62
13.2.3 Restrictions on Tablespace Encryption	63
14. Storage Hierarchy in Oracle Database.....	64
14.1 Blocks.....	64
14.2 Extents	66
14.3 Segments	66
14.4 Tablespaces	67
14.5 Storage Hierarchy Summary.....	68
14.6 Storage Parameters.....	68
15. UNDO Management (From 9i)	69
15.1 Overview	69
15.2 Undo Retention	69
15.3 Sizing and Monitoring an Undo Tablespace	70
16. Oracle Managed Files	71
16.1 Benefits of Using Oracle-Managed Files	71
16.2 File Location.....	71
16.3 Managing Different Files in OMF	72
16.3.1 Managing Controlfiles Using OMF	72
16.3.2 Managing Redo Log Files Using OMF	72
16.3.3 Managing Datafiles and Tempfiles	72
16.3.4 Managing Tablespaces Using OMF	72
16.3.5 Default Temporary Tablespace.....	73
16.4 File Location for Controlfiles and Logfiles	73
17. Server Parameter File (SPFILE)	75
17.1 Overview	75
17.2 Benefits and Features of SPFILE	75
17.3 Which SPFILE on startup.....	76
17.4 On starting the Instance.....	76
17.5 Creating an SPFILE	77
18. Oracle Networking	78
18.1 Overview	78
18.1.1 Connectivity.....	78

18.1.2 Manageability	78
18.1.3 Performance and Scalability	78
18.1.4 Network Security	78
18.1.5 Diagnosability	79
18.2 Establishing Connectivity	79
18.3 Configuring the Server with Listener:.....	79
18.4 Configuring the Client with Alias:	80
19. Databases Links	81
19.1 Location Transparency.....	81
19.2 Database link	81
20. Materialized Views.....	83
20.1 Materialized View Logs	83
20.2 Refresh Clause	84
20.2.1 Fast Clause	84
20.2.2 Complete Clause.....	84
20.2.3 Force Clause	84
20.3 Refresh Modes.....	84
20.4 Out-of-Place refresh for Mviews (New in 12c)	85
20.4.1 Out-of-Place Materialized Views	85
21. Distributed Transactions	87
21.1 Overview	87
21.2 Advantages of RDBMS	87
21.3 Disadvantages of RDBMS	88
21.4 The Two-Phase Commit Mechanism	88
21.5 Recoverer Process (RECO).....	88
22. Two Phase Commit Mechanism	90
22.1 Overview	90
22.2 Phases	90
22.2.1 Prepare Phase	90
22.2.2 Commit Phase.....	92
22.2.3 Forget Phase	93
23. Control File Management	94
215.1 Overview	94
211.2 Multiplexing the Control File	94
215.3 Recreating the Controlfile	94
24. Redolog Files Management.....	95
24.1 Overview	95
24.2 Characteristics of Redo Log Files:	95
24.3 Redo Log File Organization – Multiplexing	95
24.3.1 Adding Online Redo Log Groups	96

24.3.2 Dropping a Redo Log Group	96
24.3.3 Adding Redo Log Members.....	96
24.3.4. Dropping a Redo Log Member.....	96
24.4 Redo Log File Usage	97
25. Archivelog Files Management	98
26. Exports & Imports	99
26.1 Import export modes	99
26.2 Export (exp).....	99
26.3 NLS_LANG settings	99
26.4 Import (imp).....	100
26.5 New Feature Oracle 11g R2 export and import	100
26.6 Reasons why we perform Logical Backup:.....	100
27. Data Pump.....	104
27.1 Characteristics of Data Pump	104
27.2 Performing a Full Database Export.....	106
27.3 Performing a Schema-Mode export.....	106
27.4 Importing objects of one Schema into other	106
27.5 Performing a Table-Mode export.....	106
27.6 Estimating the Disk space needed	106
27.7 Monitoring DataPump Operations	106
27.8 Data Pump Legacy Mode	107
27.8.1 Management of File Locations in Data Pump Legacy Mode	107
27.8.2 Multiple Schema object Exported.....	109
27.8.3 Execute Privilege for Directory Object	109
27.9 Data Pump Enhancements in 12c	109
27.15.1 NOLOGGING Option (DISABLE_ARCHIVE_LOGGING).....	110
27.11.2 LOGTIME Parameter.....	110
27.15.3 Export View as Table	111
27.15.4 Change Table Compression at Import.....	111
27.15.5 Transportable Database	111
28. Transportable Tablespaces.....	112
28.1 Overview	112
28.2 Limitations on Transportable Tablespace Use	112
28.3 Transporting Tablespaces Accross Platforms	112
29. Tablespace Transport for a Single Partition.....	114
30. Backups & Recovery Concepts	116
30.1 Basics of Backup and Recovery	116
30.2 Backup and Recovery Operations	116
30.3 Elements of Backup and Recovery Strategy	117
30.4 Key Data Structures for Backup and Recovery	117

30.4.1 Datafiles	117
30.4.2 Control Files	117
30.4.3 Online Redo Log Files.....	118
30.4.4 Archived Redo Log Files	118
30.4.5 Automatic Managed Undo	119
30.5 Understanding Basic Backup.....	119
30.5.1 What Types of Failures can occur?.....	119
30.5.2 What Information should be Backed Up?	120
30.5.3 Which Backup Method should be used?.....	121
30.6 Understanding Basic Recovery Strategy	122
30.6.1 How does Recovery Work?.....	123
30.6.2 What are the types of Recovery?.....	123
30.6.3 Which Recovery method should be used?.....	124
31. Cold Backup (Physical)	125
31.1 Overview	125
31.2 Complete Recovery & Incomplete Recovery	125
32. Recovery Matrix.....	126
33. Backup & Recovery Scenarios - 1	128
33.1 CASE 1: Full Database Recovery.....	128
33.2 Case 2: Loss of a Non-SYSTEM datafile.....	128
33.2.1 Method 1: Database Recovery.....	129
33.2.2 Method 2: Data File Recovery.....	129
33.2.3 Method 3: Tablespace Recovery	129
33.3 CASE 3: Loss of a System Data File	130
33.4 CASE 4: Loss of an Object and Point in Time Recovery.	130
33.5 CASE 5: Loss of an Un-archived Online Log File.....	131
33.6 CASE 6: Recovery with a Backup Control File	131
33.6.1 Method 1:	132
33.6.2 Method 2:	132
34. Backup & Recovery Scenarios - 2	133
34.1 CASE 7: Loss of a Non-SYSTEM Data File with Rollback Segments.....	133
34.2 CASE 8: Database Crash During Hot Backups	133
34.3 CASE 9: Recovery through RESETLOGS.....	134
34.4 CASE 10: Creating Data Files	136
35. Hot Backups	138
35.1 Why Hot Backups?	138
36. Recovery Manager (RMAN)	140
36.1 What is RMAN?	140
36.2 Overview of RMAN Functional Components	140
36.2.1 Target Database	140

36.2.2 RMAN Client.....	140
36.2.3 RMAN Repository.....	141
36.2.4 Flash Recovery Area	141
36.2.5 Recovery Catalog.....	141
36.2.6 Media Managers.....	141
36.3 Why use RMAN?	141
36.4 General Features & Benefits.....	142
37. Recovery Manager	145
37.1 Configure.....	145
37.1.2 Restrictions and usage notes	145
37.1.3 Comparisons	145
37.1.4 Examples.....	146
37.2 Crosscheck	147
37.3 Delete.....	147
37.4 Drop Catalog.....	148
37.5 Reset Database	148
38. RMAN Enhancements in Oracle Database 10g	150
38.1 Flash Recovery Area.....	150
38.2 Incrementally Updated Backups	150
38.3 Fast Incremental Backups	151
38.4 BACKUP for Backupsets and Image Copies	151
38.5 Cataloging Backup Pieces	152
38.6 Improved RMAN Reporting Through V\$ Views	152
38.7 Automatic Instance Creation for RMAN TSPITR	152
38.8 Cross-Platform Tablespace Conversion	153
38.9 Enhanced Stored Scripts Commands.....	153
38.10 Backupset Compression	154
38.11 Restore Preview	154
38.12 Managing Backup Duration and Throttling	154
38.13 RMAN Tablespace Point-in-Time Recovery (TSPITR)	155
38.14 Understanding RMAN TSPITR	155
38.14.1 RMAN TSPITR Concepts.....	155
38.14.2 How TSPITR Works With an RMAN-Managed Auxiliary Instance	156
38.14.3 Deciding When to Use TSPITR.....	156
38.14.4 Limitations of TSPITR	157
38.14.5 Limitations of TSPITR Without a Recovery Catalog	158
38.15 Performing Basic RMAN TSPITR	158
38.15.1 Fully Automated RMAN TSPITR	158
38.16 Miscellaneous	160
39. RMAN Enhancements in Oracle 11g.....	161

39.1 Data Recovery Advisor	161
39.2 ZLIB Compression	161
39.3 Virtual Private Catalog	162
39.4 Merging Catalogs.....	162
39.5 Proactive Health Checks.....	163
39.6 Parallel Backup of the Same Datafile	163
40. Duplicate Database Using RMAN	164
40.1 Starting and Configuring RMAN before Duplication	164
40.2 Duplicating a Database.....	165
40.2.1 Duplicating a Database to a Remote Host with the Same Directory Structure	165
40.2.2 Duplicating a Database to a Remote Host with a Different Directory Structure.....	166
40.2.3 Creating a Duplicate Database on the Local Host.....	166
41. RMAN Enhancements in Oracle 11g R2	167
41.1 Automatic Block Recovery.	167
41.2 Flexible Set Newname Directives.	167
41.3 Enhancement Tablespace Point In Time Recovery (TSPITR).....	167
41.4 Enhancement Compression Algorithm levels	167
41.5 Duplicate Without Connection to Target Database.....	168
41.5.1 Purpose of Database Duplication.....	168
41.5.2 Techniques for Duplicating a Database.....	168
41.5.3 Backup-Based Duplication without a Target Connection	169
41.5.4 Backup-Based Duplication without a Target Connection or Recovery Catalog Connection	170
41.5.5 Backup-Based Duplication with a Target Connection	170
41.5.6 Contents of a Duplicate Database	170
41.5.7 How RMAN Duplicates a Database	170
42. RMAN Enhancements in Oracle 12c.....	172
42.1 SQL Interface Improvements	172
42.2 Sysbackup Privilege.....	173
42.3 Recover a single Table.....	173
43. Performance Tuning	175
43.1 Hardware Level.....	175
43.1.1 CPU	175
43.1.2 MEMORY	175
43.1.3 I/O	175
43.2 Operating System Level.....	176
43.3 Performance Tuning at Database Level	176
43.3.1 Oracle Versions	176
43.3.2 Routine Maintenance:-	177
43.3.3 Special Features: -	177

44. SQL Tuning	179
44.1 Defining the Basic Problem.....	179
44.2 Understanding the Optimizer and Associated Tools	180
44.2.1 SQL Conventions	180
44.2.2 Managing the WHERE Clause.....	181
44.2.3 Why a Table Scan?	181
44.2.4 Data Distribution Statistics	181
44.2.5 Disabling an Index with a Bad WHERE	181
44.3 Asking Performance Questions.....	182
44.4 Tips on working Effective SQL Statements.....	182
44.4.1 Comparing Columns in the Same Table.....	183
44.4.2 Using Nonselective Indexes	183
44.4.3 Doing Math on a Column	183
44.4.4 Using Functions.....	184
44.4.5 Finding Ranges with BETWEEN	184
44.4.6 Matching with LIKE.....	185
44.4.7 Comparing to NULL.....	185
44.4.8 Negating with NOT	186
44.4.9 Converting Values	186
44.4.10 Using OR	187
44.4.11 Finding Sets of Values with IN	187
44.4.12 Using Multicolumn Indexes	188
44.4.13 Creating Covering Indexes.....	188
44.4.14 Joining Columns	189
44.4.15 Sorting with DISTINCT and UNION	190
44.4.16 WHERE	192
44.4.17 Choosing between HAVING and WHERE	193
44.4.18 Looking at Views	194
44.4.19 Forcing Indexes.....	195
44.5 SQL Tuning Enhancements in 12c	196
44.5.1 Adaptive SQL Plan Management (SPM).....	196
44.6 New types of histograms.....	196
44.7 Monitoring database operations	197
44.7.1 Concurrent statistics gathering.....	197
44.7.2. Reporting mode for DBMS_STATS statistics gathering functions.....	197
44.7.3 Reports on past statistics gathering operations	197
44.7.4 Automatic column group creation.....	197
44.7.5 Session-private statistics for global temporary tables	197
44.7.6 SQL Test Case Builder enhancements	197
44.7.7 Online statistics gathering for bulk loads.....	198

44.7.8 Reuse of synopses after partition maintenance operations	198
44.7.9 Automatic updates of global statistics for tables with stale or locked partition statistics	198
44.7.10 Cube query performance enhancements.....	198
44.10 SQL Tuning Sets	198
44.11 Adaptive Plans in Oracle Database 12c Release 1	199
44.15.1 Adaptive Join Method	199
44.11.2 Adaptive Parallel Distribution Method.....	199
44.12 Summary.....	200
45. Application Tuning	201
45.1 Explain-Plan	202
45.2 SQL Trace	202
46. AUTOTRACE	204
47. Memory Tuning	205
47.1 Flash Cache	206
47.1.1 Tuning Memory for the Flash Cache	206
47.2 Configuring the In-Memory Column Store (12c Enhancement).....	206
48. Parsing in Oracle	209
48.1 Types of Parsing	209
48.2 Parsing process	209
48.3 Identical statements	209
48.4 Reduce hard parsing.....	209
49. Automatic Workload Repository (AWR).....	212
49.1 Overview	212
49.2 Implementation	212
49.3 Using the Statistics	212
49.4 Time Model.....	213
49.5 Active Session History (ASH).....	213
49.6 Manual Collection	214
50. Automatic Database Diagnostic Monitor (ADDM)	215
50.1 Overview	215
50.2 Enterprise Manager	215
50.3 DBMS ADVISED	216
50.4 Related Views	216
51. Automatic Shared Memory Management (ASMM).....	217
51.1 How Do You Split the Pie?	217
51.2 Which Pools is not affected?.....	217
51.3 MMAN:	218
52. Row Migration & Row Chaining.....	219
52.1 Overview	219

52.2 Oracle Block.....	219
52.3 Row Migration (RM):.....	221
52.4 Row Chaining (RC):.....	221
52.5 How to Find RM/RC	221
52.6 How to avoid/eliminate RM/RC	222
53. Networking Tuning	223
53.1 Overview	223
53.2 Monitoring MTS Connections.....	224
54. Partitioned Tables and Indexes	226
54.1 Why Use Partitioning?	226
54.2 Partitioned Tables and Indexes.....	226
54.3 Key Features.....	226
54.4 Tables versus Index partitioning.....	226
54.5 Basic Partitioning Methods	227
54.6 Updatable Partition Keys	227
54.7 Equipartitioned Tables and Indexes	227
54.8 Partitioned Indexes	227
54.9 Types of Indexes	227
54.10 Prefixed Versus Non-prefixed Local Indexes.....	228
54.11 Benefits of Range Partitioning.....	228
55. Partitioning Tables Enhancements in Oracle 11g	229
55.1 Partition Key.....	229
55.2 Partitioned Tables.....	229
55.3 Partitioned Index-Organized Tables.....	230
55.4 Partitioning Methods	230
55.5 Interval Partitioning	230
55.6 Reference Partitioning	231
55.7 System Partitioning	231
55.8 Virtual Column-Based Partitioning	231
55.9 Partitioning Tables Enhancements in oracle 11g R2	231
55.15.1 LOCAL PARTITION.....	231
55.11.2 Allow Virtual Columns in the Primary Key or Foreign Key for Reference Partitioning	232
55.15.3 System-Managed Indexes for List Partitioning	232
56. Partitioning Enhancements in Oracle 12c	233
56.1 Asynchronous index maintenance	233
56.2 Multiple partition maintenance	233
56.3 ONLINE move partition – Enhancement in 12c.....	233
57. Indexes	234
57.1 Concepts and Facts	234

57.2 Loading Data.....	234
57.3 B-Tree Index Structure.....	234
57.3.1 Creating B-Tree Indexes	236
57.3.2 Indexes and Constraints	236
57.4 Key-Compressed Index	236
57.5 Other General Topics	237
57.5.1 Altering Index Storage Parameters	237
57.5.2 Creating an Index Online	237
57.5.3 Rebuilding Indexes	237
57.5.4 Coalescing Indexes	238
57.5.5 Checking Index Validity	238
57.5.6 Dropping an Index.....	239
57.5.7 Identifying Unused Indexes	239
57.5.8 Guidelines for Creating Indexes.....	239
57.7 Special Indexes.....	241
57.7.1 Bitmap Index Structure.....	241
57.8 Reverse Key Index:.....	242
57.8.1 Limitations of Reverse-Key Indexes:.....	242
57.9 Function-Based Indexes	242
57.10 Enhancements In Oracle 12c.....	243
57.11 Descending Indexes.....	243
57.12 Index Organized Tables (IOT):.....	244
57.11.1 Row Overflow Area	244
57.11.2 Advantages:	244
57.11.3 Restrictions:	244
58. Flashback Query	245
59. Flashback Version Query.....	246
59.1 Overview	246
59.2 Querying Changes to a Table	246
59.3 Finding Out Changes During a Period	248
60. Flashback Table.....	249
61. Flashback Database	250
61.1 Overview	250
61.2 Enabling The Flash Recovery Area	250
61.2.1 Setting up the Flash Recovery Area - closed database	251
61.3 Enabling Flashback Database	252
61.3.1 Flash Recovery status queries	253
61.4 Storing Backups In Flash Recovery Area	253
61.4.1 RMAN Daily Backup Scheme Using Image Copies	253
61.5 Flashback Database: An Example	255

61.6 Flashback Data Archive	256
61.6.1 Oracle Flashback Technology	257
61.7 Architecture	258
61.8 Understanding Flashback Data Archive.....	258
61.8.1 Viewing Flashback Data Archive Data	259
61.9 Flashback Data Archive Enhancement in Oracle 11g R2.....	259
61.10 Flashback Data Archive Enhancement in Oracle 12C R1	260
61.10.1 User context tracking.....	260
61.11 Database Hardening	260
62. SQL*Loader.....	261
62.1 Overview	261
62.2 Key Features.....	261
62.3 File Types	261
62.4 Load Methods	261
62.5 Important Points	262
62.6 Oracle 12C SQL Loader Express Mode Basic Example	264
62.6.1 SQL Loader - Oracle 12C - Express Mode	264
62.6.2 Express Mode - Basic Definition	264
Working Example - Express Mode	264
63. Database Auditing	267
63.1 Overview	267
63.2 Types of Auditing:.....	267
63.3 Unified and Conditional Auditing	269
64. Log Miner.....	274
65. Automatic Storage Management (ASM).....	278
65.1 What is ASM?	278
65.1.1 Dynamic Performance Views	280
65.2 Important Points	280
66. ASM Enhancements in 11G R1	282
66.1 Automatic Storage Management Overview	282
66.2 Automatic Storage Management New Features	283
66.3 Improved Scalability and Performance	283
66.4 VLDB Support	284
66.5 Multiple Allocation Unit size	285
66.6 Faster Rebalance with Restricted Mount Option	285
66.7 ASM Rolling Upgrades and Patching.....	286
66.8 Manageability Enhancements.....	287
66.8.1 Disk Group Compatibility Attributes.....	287
66.9 New ASMCMD commands	287
66.10 SYSASM role	289

66.11 Mount/Drop FORCE disk group	290
66. ASM Enhancements in 11G R2	291
66.1 Disk Group Rename	291
66.2 Specifying the Sector Size for Disk Drives	292
66.3 Oracle ASM Configuration Assistant (ASMCA).....	293
66.4 ASMCMD Enhancements	293
66.5 Intelligent Data Placement.....	294
67. ASM Enhancements in 12C R1	295
67.1 Oracle Flex ASM	295
67.2 Oracle ASM Disk Scrubbing.....	297
67.3 Oracle ASM Disk Resync Enhancements	298
67.4 Even Read For Disk Groups	298
67.5 Shared Oracle ASM Password File in a Disk Group.....	298
67.6 Updated Key Management Framework	298
67.7 Oracle ASM Rebalance Enhancements.....	298
68. Introduction to Oracle Data Guard	299
68.1 Data Guard Configurations	299
68.1.1 Primary Database.....	299
68.1.2 Standby Databases	299
68.1.3 Configuration Example	300
69. Data Guard	301
69.1 Overview	301
69.2 Concepts	302
69.2.1 No Data Loss.....	302
69.2.2 No Data Divergence	302
69.3 Overview of Oracle Data Guard Functional Components	302
69.3.1 Data Guard Configuration	302
69.3.2 Role Management.....	302
69.4 Data Guard Protection Modes	303
69.5 Data Guard Broker.....	303
69.6. Data Guard Architecture	303
69.7 What's New in Oracle Data Guard 10g Release 1?	304
69.7.1 Real Time Apply	304
69.7.2 Integration with Flashback Database.....	304
69.7.3 Simplified Browser-based Interface	304
69.8 What's New in Oracle Data Guard 10g Release 2?	305
69.8.1 Fast-Start Failover.....	305
69.8.2 Improved Redo Transmission	305
69.8.3 Easy conversion of a physical standby database to a reporting database	305
69.8.4 Automatic deletion of applied archived redo log files in logical standby DB's.....	305

69.8.5 Fine-grained monitoring of Data Guard configurations	305
69.9 Data Guard Benefits	305
69.15.1 Disaster recovery and high availability	305
69.11.2 Complete data protection	305
69.15.3 Efficient utilization of system resources	306
69.15.4 Flexibility in data protection to balance availability against performance requirements	306
69.15.5 Protection from communication failures	306
69.15.6 Centralized and simple management.....	306
69.15.7 Integrated with Oracle database	306
70. Data Guard Protection Modes	307
71. Starting Up and Shutting Down a Physical Standby Database.....	310
71.1 Starting Up a Physical Standby Database.....	310
71.2 Shutting Down a Physical Standby Database	310
71.3 Opening a Physical Standby Database	310
72. Standby Database Types	311
72.1 A standby database can be one of these types:	311
72.1.1 Physical Standby Databases	311
72.2 Benefits of a Physical Standby Database	311
72.3 Data protection.....	311
72.4 Reduction in primary database workload.....	311
72.5 Performance.....	311
72.5.1 Logical Standby Databases	312
72.6 Benefits of a Logical Standby Database:.....	312
72.7 Protection against additional kinds of failure	312
72.8 Efficient use of resources.....	312
72.9 Workload distribution	312
72.10 Optimized for reporting and decision support requirements.....	312
72.11 Minimizing downtime on software upgrades	313
72.15.1 Snapshot Standby Databases.....	313
72.12 Benefits of a Snapshot Standby Database	313
72.13 User Interfaces for Administering Data Guard Configurations.....	314
73. Monitoring Stand by Databases	315
73.1 Primary DB Changes That Require Manual Intervention at a Physical Standby.....	315
73.1.1 Adding a Datafile or Creating a Tablespace	316
73.1.2 Dropping Tablespaces and Deleting Datafiles	318
73.1. 3 Using DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES	318
73.1.4 Using Transportable Tablespaces with a Physical Standby Database	318
73.1.5 Renaming a Datafile in the Primary Database	319
73.1.6 Add or Drop a Redo Log File Group.....	320

73.1.7 NOLOGGING or Unrecoverable Operations.....	320
73.1.8 Refresh the Password File	320
73.1.9 Reset the TDE Master Encryption Key	320
73.2 Recovering Through the OPEN RESETLOGS Statement	321
73.3 Monitoring Primary, Physical Standby, and Snapshot Standby Databases.....	321
74. Dataguard Services	323
74.1 Redo Transport Services	323
74.2 Apply Services	323
74. 3 Role Transitions	324
74. 3.1 Introduction to Role Transitions.....	325
74. 3.2 Preparing for a Role Transition	325
74. 3.3 Choosing a Target Standby Database for a Role Transition	325
74. 3.4 Switchovers	326
74. 3.5 Failovers	328
74. 3.6 Preparing for a Failover.....	329
74. 3.7 Role Transition Triggers	329
74. 3.8 Role Transitions Involving Physical Standby Databases	330
75. Redo Apply Services	338
75.1 Introduction to Apply Services.....	338
75.2 Apply Services Configuration Options	338
75.2.1 Specifying a Time Delay for the Application of Archived Redo Log Files.....	339
75.3 Applying Redo Data to Physical Standby Databases	340
75.3.1 Starting Redo Apply.....	340
75.3.2 Stopping Redo Apply.....	340
75.4 Applying Redo Data to Logical Standby Databases	340
75.4.1 Starting and Stopping SQL Apply.....	341
76. Redo Transport Services	342
76.1 Introduction to Redo Transport Services	342
76.2 Configuring Redo Transport Services.....	342
76.2.1 Redo Transport Security.....	342
76.2.2 Configuring an Oracle Database to Send Redo Data	343
76.2.3 Configuring an Oracle Database to Receive Redo Data	345
76.3 Monitoring Redo Transport Services	346
76.3.1 Monitoring Redo Transport Status	347
76.3.2 Monitoring Synchronous Redo Transport Response Time	348
76.3.3 Redo Gap Detection and Resolution.....	348
76.4 Manual Gap Resolution	349
76.4.1 Redo Transport Services Wait Events	350
77. Logical Standby Dataguard	351
77.1 Introduction	351

77.2 When to choose Logical Standby database	351
77.3 Prerequisite Conditions for creating a Logical Standby Database:	352
77.3.1 Improvements in Oracle Data Guard in Oracle 10gr2:.....	353
78. Cloning Oracle Database	354
79. Oracle Data Guard Broker.....	357
79.1 Overview of Oracle Data Guard and the Broker	357
79.1.1 Data Guard Configurations and Broker Configurations	357
79.1.2 Oracle Data Guard Broker.....	357
79.2 Benefits of Data Guard Broker.....	359
79.3 Data Guard Broker Management Model.....	362
79.3 Data Guard Broker User Interfaces	363
79.3.1 Oracle Enterprise Manager	363
79.3.2 Data Guard Command-Line Interface (DGMGRL).....	363
79.4 Data Guard Monitor	365
79.4.1 Data Guard Monitor (DMON) Process	365
79.4.2 Configuration Management.....	365
79.4.3 Database Property Management	366
80. What's New in Oracle Data Guard?	367
81. Active Dataguard	369
81.1 Traditional Data Guard.....	369
81.2 Oracle Active Data Guard	369
81.3 Unique Advantages of Oracle Active Data Guard.....	370
82. Snapshot Standby Databases	371
82.1 Benefits of a Snapshot Standby Database	371
82.2 Using a Snapshot Standby Database.....	371
83. Data Guard Enhancements in Oracle 11 G R2	373
83.1 Redo Apply and SQL Apply.....	373
83.2 Redo Apply.....	373
83.3 SQL Apply	373
83.4 Compressed Table Support in Logical Standby Databases and Oracle LogMiner.....	374
83.5 Configurable Real-Time Query Apply Lag Limit.....	374
83.6 Support Up to 30 Standby Database	374
83.7 Automatic Repair of Corrupt Data Blocks	374
83.8 Manual Repair of Corrupt Data Blocks	374
84. Data Guard Enhancements in Oracle 12C	375
84.1 Far Sync.....	375
84.1 Creating a Far Sync Instance	376
84.1.1 Creating and Configuring a Far Sync Instance	377
84.2 Configuring an ALTERNATE Destination	379
84.3 Additional Configurations	379

84.3.1 Maintaining Protection After a Role Change	380
84.3.2 Far Sync Instance High Availability	381
84.4 Supported Protection Modes for Far Sync Instances	382
84.4.1 Far Sync Instances in Maximum Availability Mode Configurations.....	382
84.4.2 Far Sync Instances in Maximum Performance Mode Configurations	382
84.5 Real Time Cascading	383
84.6 Other Feautures	385
85. OEM Jobs & Events	386
85.1 Overview	386
85.2 Creating Jobs without Programs.....	386
85.3 Associating Jobs with Programs	388
85.4 Classes, Plans, and Windows.....	389
85.5 Monitoring	390
85.6 Administration	391
86. Glossary.....	393
VOLUME -2: Multitenant Architecture.....	399
87. Introduction to the Multitenant Architecture	401
87.1 Containers in Multitenancy.....	403
87.2 CONTAINER Database.....	404
87.3 PLUGGABLE Databases	407
88. User Management	410
88.1 Introduction	410
88.2 Create Users	410
88.2.1 Create Common Users.....	410
88.2.2 Create Local Users	410
88.3 Create Roles	411
88.3.1 Create Common Roles.....	411
88.3.2 Create Local Roles	411
89. Data Pump Enhancements in 12c	412
90. RMAN Enhancements in 12c	413
90.1 PDB subset Cloning.....	414
90.2 PDB Metadata Clone	414
90.2.1 Restrictions.....	414
90.3 PDB Remote Clone.....	415

1. Oracle Database Server

Architecture includes physical components, memory components, processes, and logical structures. Oracle database server consists of Oracle Instance and Oracle Database. The database includes several different types of files:

- Datafiles
- Control files
- Redo log files
- Archive redo log files (Optional)

The Oracle server also access parameter files and password files. This set of files has several purposes.

- Enable system users to process SQL statements.
- Improve system performance.
- Ensure the database can be recovered if there is a software/hardware failure.

The DB server must manage large amounts of data in multi-user environment. The server must manage concurrent access to the same data. The server must deliver high performance. This generally means fast response times.

1.1 Oracle Instance

An Oracle Instance consists of two different sets of components

- Background Processes
- Memory Structure

1.1.1 Background Processes

Background processes (PMON, SMON, RECO, DBW0, LGWR, CKPT, D000 and others). These will be covered later in detail, but basically each background process is a computer program. These processes perform input/output and monitor other Oracle processes to provide good performance and database reliability.

1.1.2 Memory Structure

When an instance starts up, a memory structure called the System Global Area (SGA) is allocated. At this point the background processes also start. The Oracle Instance provides access to an Oracle database. An Oracle Instance opens one and only one database.

1.2 Oracle Database

An Oracle database consists of files.

- The control files are used to synchronize all database activities.
- The redo log files are used to recover the database in the event of application program failures, instance failures and other minor failures.
- The database files that store the database information that a firm or organization needs in order to operate.
- The archived redo log files are used to recover the database if a disk fails.
- Other files
- The required parameter file that is used to specify parameters for configuring an Oracle instance when it starts up.
- The optional password file authenticates special users of the database – these are termed privileged users and include database administrators.
- Alert and Trace Log Files – these files store information about errors and actions taken that affect the configuration of the database.

1.3 Processes

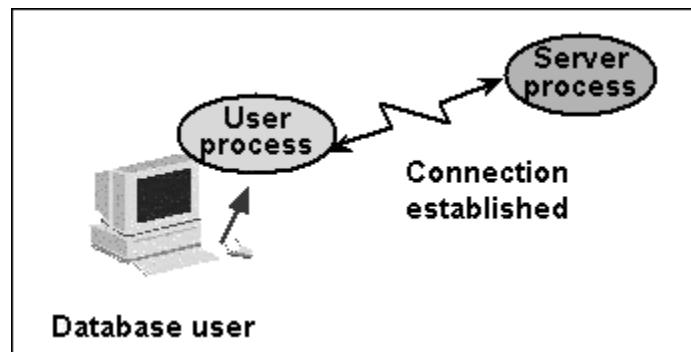
You need to understand three different types of Processes:

- User Process: Starts when a database user requests to connect to an Oracle Server.
- Server Process: Establishes the Connection to an Oracle Instance when a User Process requests connection makes the connection for the User Process.
- Background Processes: These start when an Oracle Instance is started up.

1.3.1 User Processes

In order to use Oracle, you must obviously connect to the database. This must occur whether you're using SQLplus, an Oracle tool such as Designer or Forms, or an application program.

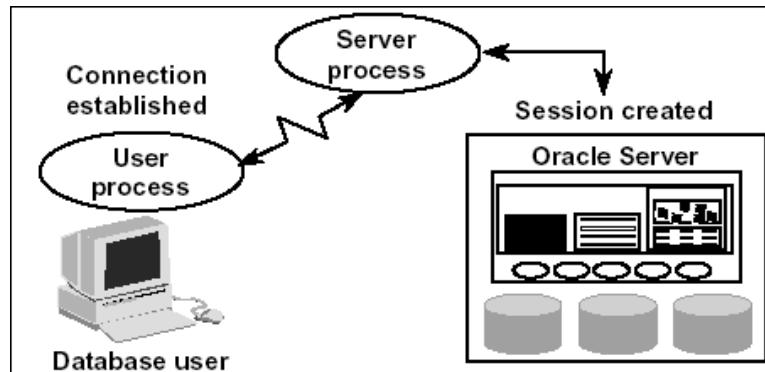
- A program that requests interaction with the Oracle Server
- Must first establish a connection
- Doesn't interact directly with the Oracle Server



This generates a User Process (a memory object) that generates programmatic calls through your user interface (SQLplus, Integrated Developer Suite, or application program) that creates a session and causes the generation of a Server Process that is either dedicated or shared.

1.3.2 Server Processes

- A program that directly interacts with the Oracle Server
- Fulfills calls generated and return results
- Can be Dedicated or Shared Server



As you have seen, the Server Process is the go-between for a User Process and the Oracle Instance. In a Dedicated Server environment, there is a single Server Process to serve each User Process. In a Shared Server environment, a Server Process can serve several User Processes, although with some performance reduction.

System users can connect to an Oracle database through SQLplus or through an application program like the Integrated Developer Suite (the program becomes the system user).

This connection enables users to execute SQL statements. The act of connecting creates a communication pathway between a user process and an Oracle Server. As is shown in the figure above, the User Process communicates with the Oracle Server through a Server Process. The User Process executes on the client computer. The Server Process executes on the server computer, and actually executes SQL statements submitted by the system user.

The figure shows a one-to-one correspondence between the User and Server Processes. This is called a Dedicated Server connection. An alternative configuration is to use a Shared Server where more than one User Process shares a Server Process.

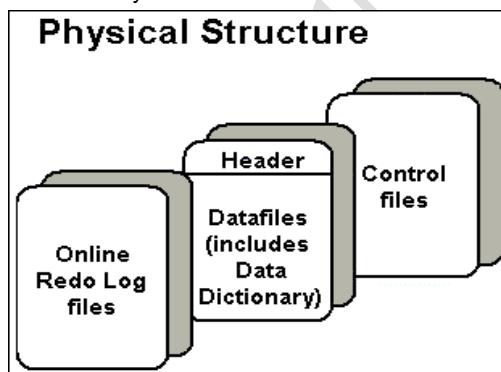
Sessions

When a user connects to an Oracle server, this is termed a Session. The session starts when the Oracle server validates the user for connection. The session ends when the user logs out (disconnects) or if the connection terminates abnormally (network failure or client computer failure). A user can typically have more than one concurrent session, e.g., the user may connect using SQLplus and also connect using Internet Developer Suite tools at the same time. The limit of concurrent session connections is controlled by the DBA. If a system user attempts to connect and the Oracle Server is not running, the system user receives the Oracle Not Available error message.

1.4 Physical Structure Files

Oracle database consists of physical files. These physical structures include three types of files:

- Datafiles - these contain the organization's actual data.
- Redo log files - these contain a record of changes made to the database, and enable recovery when failures occur.
- Control files - these are used to synchronize all database activities.



Other key files as noted above include:

- Parameter File - there are two types of parameter files.
- The init.ora file (also called the PFILE) is a static parameter file. It contains parameters that specify how the database instance is to start up. For example, some parameters will specify how to allocate memory to the various parts of the system global area.
- The spfile.ora is a dynamic parameter file. It also stores parameters to specify how to startup a database; however, its parameters can be modified while the database is running.
- Password File - specifies which *special* users are authenticated to startup/shut down an Oracle Instance.
- Archived Redo Log Files - these are copies of the redo log files and are necessary for recovery in an online, online transaction processing (OLTP) environment in the event of a disk failure.

1.5 Memory Structure

The memory structures include two areas of memory:

- System Global Area (SGA) – this is allocated when an Oracle Instance starts up.
- Program Global Area (PGA) – this is allocated when a Server Process starts up.

1.5.1 System Global Area

The SGA is an area in memory that stores information shared by all database processes and by all users of the database. This information includes both organizational data and control information used by the Oracle Server. The SGA is allocated in virtual memory. The size of the SGA is established by the parameter SGA_MAX_SIZE in the parameter file.

The SGA is allocated when an Oracle instance (database) is started up based on values specified in the initialization parameter file (either PFILE or SPFILE).

The SGA (also called the shared global area) has the following mandatory memory structures:

- Shared Pool – includes two components:
- Library Cache
- Data Dictionary Cache
- Database Buffer Cache
- Redo Log Buffer
- Other structures (for example, lock and latch management, statistical data)
- Additional optional memory structures in the SGA include:
 - Large Pool
 - Java Pool
 - Streams Pool

Oracle 8i and earlier versions of the Oracle Server used a Static SGA. This meant that if modifications to memory management were required, the database had to be shutdown, modifications were made to the init.ora parameter file, and then the database had to be restarted.

Oracle 9i and 12c use a Dynamic SGA. Memory configurations for the system global area can be made without shutting down the database instance. The advantage is obvious. This allows the DBA to resize the Database Buffer Cache and Shared Pool dynamically.

Several initialization parameters are set that affect the amount of random access memory dedicated to the SGA of an Oracle Instance. These are:

- DB_CACHE_SIZE: This is the size of the Database Buffer Cache in database blocks. Block sizes vary among operating systems. We use 8KB block sizes. The total blocks in the cache defaults to 48 MB on LINUX/UNIX and 52 MB on Windows operating systems.
- LOG_BUFFER: This is the number of bytes allocated for the Redo Log Buffer.
- SHARED_POOL_SIZE: This is the number of bytes of memory allocated to shared SQL and PL/SQL. The default is 16 MB. If the operating system is based on a 64-bit configuration, then the default size is 64 MB.
- LARGE_POOL_SIZE: Since this is an optional memory object, the size of the Large Pool defaults to zero. If the init.ora parameter PARALLEL_AUTOMATIC_TUNING is set to TRUE, then the default size is automatically calculated.
- JAVA_POOL_SIZE: This is another optional memory object. The default is 24 MB of memory.

The size of the SGA cannot exceed the parameter SGA_MAX_SIZE minus the combination of the size of the additional parameters, DB_CACHE_SIZE, LOG_BUFFER, SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE.

Memory is allocated to the SGA as contiguous virtual memory in units termed granules. Granule size depends on the estimated total size of the SGA, which as was noted above, depends on the SGA_MAX_SIZE parameter. Granules are sized as follows:

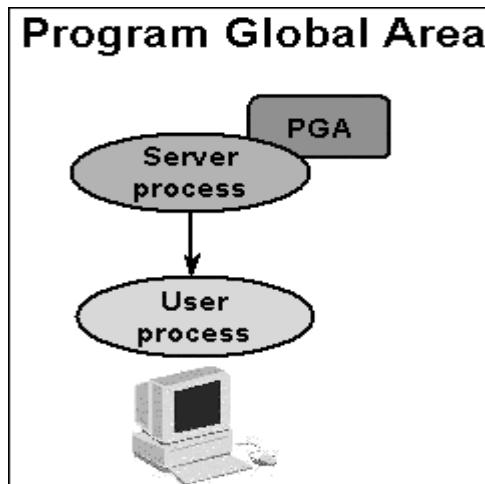
- If the SGA is less than 128MB in total, each granule is 4MB.
- If the SGA is greater than 128MB in total, each granule is 16MB.

Granules are assigned to the Database Buffer Cache and Shared Pool, and these two memory components can dynamically grow and shrink. Using contiguous memory improves system performance. Granules are allocated when the Oracle server starts a database instance in order to provide memory-addressing space to meet the SGA_MAX_SIZE parameter. The minimum is 3 granules: one each for the fixed SGA, Database Buffer Cache, and Shared Pool.

1.5.2 Program Global Area

The Program Global Area is also termed the Process Global Area (PGA) and is a part of memory allocated that is outside of the Oracle Instance. The PGA stores data and control information for a single Server Process or a single Background Process. It is allocated when a process is created and the memory is scavenged by the operating system when the process terminates. This is NOT a shared part of memory – one PGA to each process only.

- Memory reserved for each user connecting to an oracle database
- Allocated when a process is created
- Deallocated when the process is terminated
- Used by only one process



The content of the PGA varies, but generally includes the following:

- Private SQL Area: Data for binding variables and runtime memory allocations. A user session issuing SQL statements has a Private SQL Area that may be associated with a Shared SQL Area if the same SQL statement is being executed by more than one system user. This often happens in OLTP environments where many users are executing and using the same application program.
 - Dedicated Server environment – the Private SQL Area is located in the Program Global Area.
 - Shared Server environment – the Private SQL Area is located in the System Global Area.
- Session Memory: Memory that holds session variables and other session information.
- SQL Work Area: Memory allocated for sort, hash-join, bitmap merge, and bitmap create types of operations.
 - Oracle 9i enables automatic sizing of the SQL Work Areas by setting the WORKAREA_SIZE_POLICY = AUTO parameter (this is the default!) and PGA_AGGREGATE_TARGET = n (where n is some amount of memory established by the DBA). However, the DBA can let Oracle 11g determine the appropriate amount of memory.
 - pga_aggregate_limit=2GB Oracle 12c gives some limited size of PGA.
 - Oracle 8i and earlier required the DBA to set the following parameters to control SQL Work Area memory allocations:
 - SORT_AREA_SIZE.
 - HASH_AREA_SIZE.
 - BITMAP_MERGE_AREA_SIZE.
 - CREATE_BITMAP_AREA_SIZE.

2. System Global Area (SGA)

Simply stated, the system global area (SGA) is just shared memory structures that are created at instance startup, hold information about the instance and control its behavior. The following table gives a brief synopsis of the particular components of the SGA, the variables that control the size of memory allocated; some of the areas of the Oracle server the particular component have an influence on, and then a very brief description. What can be seen from this simple list is that there are plenty of options available for us to tweak the SGA and without a complete understanding of what our applications are doing in the background, our ability to guess the appropriate amount of memory to give each of these individual components is not always optimal. What we do not want to have happen in this process of allocation of memory is to waste it.

SGA Component	Size Controlled By	Areas Of Influence	Simple Descriptions
Shared Pool Oracle 6 thru 12c	SHARED_POOL_SIZE	Library Cache <ul style="list-style-type: none"> ▪ Shared SQL areas ▪ Private SQL areas ▪ PL/SQL procedures and packages ▪ Various control structures 	Oracle needs to allocate & deallocate memory as SQL or procedural code is executed based on the individual needs of users' sessions and in accordance to the LRU algorithm.
		Dictionary Cache <ul style="list-style-type: none"> ▪ Row cache 	Highly accessed memory structures that provide information on object structures to SQL statements being parsed.
Redo Log Buffer Oracle 6 thru 12c	LOG_BUFFER	<ul style="list-style-type: none"> ▪ Redo entries 	Holds changes made to data and allows for the reconstruction of data in the case of failure.
Database Buffer Cache Oracle 6 thru 12c	DB_2K_CACHE_SIZE DB_4K_CACHE_SIZE DB_8K_CACHE_SIZE DB_16K_CACHE_SIZE DB_32K_CACHE_SIZE DB_KEEP_CACHE_SIZE DB_RECYCLE_CACHE_SIZE	<ul style="list-style-type: none"> ▪ Write list ▪ LRU list 	Holds copies of data requested by SQL and reduces requests to disk by having data in memory. You may have many different buffer caches that help segregate on usage patterns.
Large Pool From Oracle 8	LARGE_POOL_SIZE	<ul style="list-style-type: none"> ▪ Shared server ▪ Oracle XA ▪ I/O server processes ▪ Backup & restore 	For large memory allocations
Java Pool From Oracle 8i	JAVA_POOL_SIZE	<ul style="list-style-type: none"> ▪ Run state ▪ Methods ▪ Classes ▪ Session code ▪ Data in JVM 	Memory available for the Java memory manager to use for all things Java.
Streams Pool From Oracle 11g	STREAMS_POOL_SIZE	<ul style="list-style-type: none"> ▪ Stream activity 	New to Oracle 11g, memory available for stream processing.

Inmemory From Oracle 12c	INMEMORY_SIZE	▪ new section of the SGA	New to Oracle 12c to store specific groups of columns, whole tables
--------------------------	---------------	--------------------------	---

You can look at the size of your SGA by looking at the initialization parameters that control its size. Here is a simple query and its output.

```
SQL> select name, value from v$parameter where name in
  ('shared_pool_size', 'java_pool_size', 'streams_pool_size', 'log_buffer',
  'db_cache_size', 'db_2k_cache_size', 'db_4k_cache_size',
  'db_8k_cache_size', 'db_16k_cache_size', 'db_32k_cache_size',
  'db_keep_cache_size', 'db_recycle_cache_size', 'large_pool_size');
```

NAME	VALUE
shared_pool_size	629145600
large_pool_size	0
java_pool_size	4194304
streams_pool_size	0
db_cache_size	50331648
db_2k_cache_size	16777216
db_4k_cache_size	0
db_8k_cache_size	0
db_16k_cache_size	0
db_32k_cache_size	0
db_keep_cache_size	0
db_recycle_cache_size	0
log_buffer	5185536

```
SQL> Show SGA
```

Total System Global Area	1233125376 bytes
Fixed Size	2934360 bytes
Variable Size	633342376 bytes
Database Buffers	67108864 bytes
Redo Buffers	5451776 bytes
In-Memory Area	524288000 bytes

Switching over to the Automatic Shared Memory Tuning is as easy as setting an initialization parameter. How this will behave under load is yet to be determined but since these numbers are driven by the various advisories. We suggest you take a snapshot of your initialization parameters before letting Oracle take control and then compare the end settings that Oracle has implemented. It is always easy to switch back, just reset the SGA_TARGET parameter and set the individual components back to their original values.

3. Result Cache

3.1 Result Cache

The result cache is composed of the SQL query result cache and PL/SQL function result cache, which share the same infrastructure. The DBMS_RESULT_CACHE package provides administration subprograms, which, for example, flush all cached results and turn result-caching on or off system-wide. The dynamic performance views V\$RESULT_CACHE_* allow the developer and DBA to determine, for example, the cache-hit success for a certain SQL query or PL/SQL function. Similar to the result cache, the client result cache also caches results, except that the caching is done on the client side.

3.2 Implementing SQL Query Result Cache

The new SQL Query Result Cache enables explicit caching of queries and query fragments in an area of the shared pool called Result Cache Memory. When a query is executed the result cache is built up and the result is returned. The database can then use the cached results for subsequent query executions, resulting in faster response times. Cached query results become invalid when data in the database object(s) being accessed by the query is (are) modified.

You can enable Query Result Cache at the database level using the RESULT_CACHE_MODE initialization parameter in the database initialization parameter file. The same parameter can also be used at the session level using the ALTER SESSION command. RESULT_CACHE_MODE can be set to:

AUTO the optimizer will decide based on a number of factors whether or not to cache the result. Decision factors include the frequency of query execution, the cost of building the result and the frequency of changes against the underlying database objects.

MANUAL (default) you have to add the RESULT_CACHE hint to your queries in order for results to be cached or to be served out of the cache. The RESULT_CACHE hint can also be added in sub queries and in-line views.

FORCE results are always stored in the Result Cache Memory if possible.

The use of the SQL Query Result Cache introduces the Result Cache operator in the query execution plan.

Perform the following steps to understand the use of Query Result Cache

3.3 SQL Query Result Cache

Results of queries and query fragments can be cached in memory in the SQL query result cache. The database can then use cached results to answer future executions of these queries and query fragments. Because retrieving results from the SQL query result cache is faster than rerunning a query, frequently run queries experience a significant performance improvement when their results are cached. Users can annotate a query or query fragment with a result cache hint to indicate that results are to be stored in the SQL query result cache.

You can set the RESULT_CACHE_MODE initialization parameter to control whether the SQL query result cache is used for all queries (when possible), or only for queries that are annotated. The database automatically invalidates a cached result whenever a transaction modifies the data or metadata of any of the database objects used to construct that cached result.

3.4 PL/SQL Function Result Cache

A PL/SQL function is sometimes used to return the result of a computation whose inputs are one or several parameterized queries issued by the function. In some cases, these queries access data (for example, the catalog of wares in a shopping application) that changes very infrequently compared to the frequency of calling the function. You can include syntax in the source text of a PL/SQL function to request that its results be cached and, to ensure correctness, that the cache be purged when any of a list of tables experiences DML. The look-up key for the cache is the combination of actual arguments with which the function is invoked. When a particular invocation of the result-cached function is a cache hit, then the function body is not executed; instead, the cached value is returned immediately.

4. FLASH CACHE

The database buffer cache, also called the buffer cache, is the memory area that stores copies of data blocks read from data files. A buffer is a main memory address in which the buffer manager temporarily caches a currently or recently used data block. All users concurrently connected to a database instance share access to the buffer cache.

Oracle Database uses the buffer cache to achieve the following goals:

- Optimize physical I/O

The database updates data blocks in the cache and stores metadata about the changes in the redo log buffer. After a COMMIT, the database writes the redo buffers to disk but does not immediately write data blocks to disk. Instead, database writer (DBWn) performs lazy writes in the background.

- Keep frequently accessed blocks in the buffer cache and write infrequently accessed blocks to disk

When Database Smart Flash Cache (flash cache) is enabled, part of the buffer cache can reside in the flash cache. This buffer cache extension is stored on a flash disk device, which is a solid state storage device that uses flash memory. The database can improve performance by caching buffers in flash memory instead of reading from magnetic disk.

The flash cache is typically more economical than additional main memory, and is an order of magnitude faster than disk drives.

4.1 When to Configure the Flash Cache

Consider adding the flash cache when all of the following are true:

- Your database is running on the Solaris or Oracle Linux operating systems. The flash cache is supported on these operating systems only.
- The Buffer Pool Advisory section of your Automatic Workload Repository (AWR) report or STATSPACK report indicates that doubling the size of the buffer cache would be beneficial.
- db file sequential read is a top wait event.
- You have spare CPU.

4.2 Assign Sizing the Flash Cache

As a general rule, size the flash cache to be between 2 times and 10 times the size of the buffer cache. Any multiplier less than two would not provide any benefit. If you are using automatic shared memory management, make the flash cache between 2 times and 10 times the size of SGA_TARGET. Using 80% of the size of SGA_TARGET instead of the full size would also suffice for this calculation.

When the number of clean or unused buffers is low, the database must remove buffers from the buffer cache. The algorithm depends on whether the flash cache is enabled:

4.3 Flash cache disabled

The database re-uses each clean buffer as needed, overwriting it. If the overwritten buffer is needed later, then the database must read it from magnetic disk.

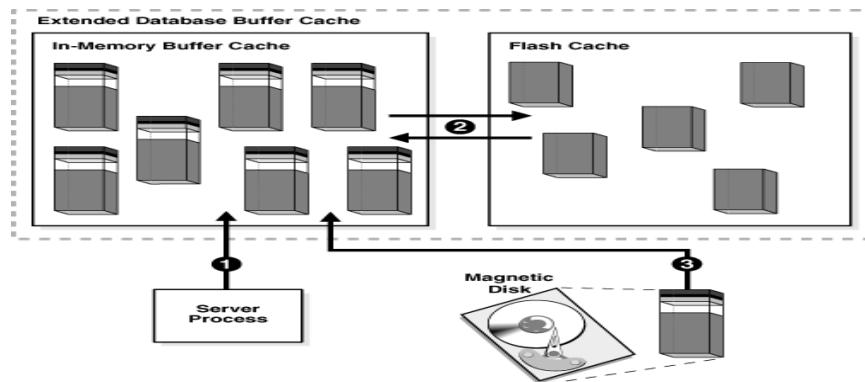
4.4 Flash cache enabled

DBWn can write the body of a clean buffer to the flash cache, enabling reuse of its in-memory buffer. The database keeps the buffer header in an LRU list in main memory to track the state and location of the buffer body in the flash cache. If this buffer is needed later, then the database can read it from the flash cache instead of from magnetic disk.

When a client process requests a buffer, the server process searches the buffer cache for the buffer. A cache hit occurs if the database finds the buffer in memory. The search order is as follows:

1. The server process searches for the whole buffer in the buffer cache. If the process finds the whole buffer, then the database performs a logical read of this buffer.
2. The server process searches for the buffer header in the flash cache LRU list. If the process finds the buffer header, then the database performs an optimized physical read of the buffer body from the flash cache into the in-memory cache.
3. If the process does not find the buffer in memory (a cache miss), then the server process performs the following steps:
 1. Copies the block from a data file into memory (a physical read)
 2. Performs a logical read of the buffer that was read into memory

The extended buffer cache includes both the in-memory buffer cache, which contains whole buffers, and the flash cache, which contains buffer bodies. In the figure, the database searches for a buffer in the buffer cache and, not finding the buffer, reads it into memory from magnetic disk.



Accessing data through a cache hit is faster than through a cache miss. The buffer cache hit ratio measures how often the database found a requested block in the buffer cache without needing to read it from disk.

The database can perform physical reads from either a data file or a [temp file](#). Reads from a data file are followed by logical I/Os. Reads from a temp file occur when insufficient memory forces the database write data to a [temporary table](#) and read it back later. These physical reads bypass the buffer cache and do not incur a logical I/O.

4.5. Flash Cache Initialization Parameters

Parameter	Description
db_flash_cache_file	Specifies the path and file name for the file to contain the flash cache, in either the operating system file system or an Oracle Automatic Storage Management disk group. If the file does not exist, the database creates it during startup. The file must reside on a flash disk device. If you configure the flash cache on a disk drive (spindle), performance may suffer. The following is an example of a valid value for db_flash_cache_file:/dev/fioa1
db_flash_cache_size	Specifies the size of the flash cache. Must be less than or equal to the physical memory size of the flash disk device. Expressed as nG, indicating the number of gigabytes (GB). For example, to specify a 16 GB flash cache, set db_flash_cache_size to 16G.

5. Background Processes

An Oracle instance runs on two types of processes - **Server** and **Background**. Server processes are created to handle requests from sessions connected to the instance. Background processes, as the name says, are processes running behind the scene and are meant to perform IO activities or to deal with abnormal conditions arising in the lifetime of the instance.

There are two types of background processes for an instance.

Mandatory - These background processes must always be running when the instance is up.

Optional - These background processes may or may not be used depending on which optional Oracle features are being used in the database.

Each background process is meant for a specific purpose and its role is well defined. Background processes are visible as separate operating system processes in Unix/Linux. In Windows, these run as separate threads within the same process. Background processes are invoked automatically when the instance is started.

5.1 Mandatory Processes

5.1.1 System Monitor

Process Name: SMON

Max Processes: 1

The system monitor performs recovery when a failed instance starts up again. In a Real Application Clusters database, the SMON process of one instance can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use. These transactions are eventually recovered by SMON when the tablespace or file is brought back online. It also coalesces contiguous free extents in dictionary-managed tablespaces that have PCTINCREASE set to a non-zero value.

SMON checks the SCN in all datafile headers when the database is started. Everything is OK if these entire SCNs match the SCN found in the controlfile. If the SCNs don't match, the database is in an inconsistent state.

5.1.2 Process Monitor

Process Name: PMON

Max Processes: 1

This process monitor performs process recovery when a user process fails. It will rollback uncommitted transactions. PMON is also responsible for cleaning up the database buffer cache and freeing resources that were allocated to a process. PMON also registers information about the instance and dispatcher processes with network listener. PMON also checks on the dispatcher processes and server processes and restarts them if they have failed.

5.1.3 Log Writer

Process Name: LGWR

Max Processes: 1

The log writer process writes data from the redo log buffers to the redo log files on disk. The writer is activated under the following conditions:

When a transaction is committed, a System Change Number (SCN) is generated and tagged to it. Log writer puts a commit record in the redo log buffer and writes it to disk immediately along with the transactions redo entries. Changes to actual data blocks are deferred until a convenient time (Fast-Commit Mechanism).

Every 3 seconds.

When the redo log buffer is 1/3 full.

When DBWn signals the writing of redo records to disk. All redo records associated with changes in the block buffers must be written to disk first (The write-ahead protocol). While writing dirty buffers, if the DBWn process finds that some redo information has not been written, it signals the LGWR to write the information and waits until the control is returned.

Log writer will write synchronously to the redo log groups in a circular fashion. If any damage is identified with a redo log file, the log writer will log an error in the LGWR trace file and the system Alert Log. Sometimes, when additional redo log buffer space is required, the LGWR will even write uncommitted redo log entries to release the held buffers. LGWR can also use group commits (multiple committed transaction's redo entries taken together) to write to redo logs when a database is undergoing heavy write operations.

5.1.4 Database Writer

Process Name: DBWR

Max Processes: 100

The **database writer process (DBW)** writes the contents of database buffers to data files. DBW processes write modified buffers in the database buffer cache to disk.

Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes—DBW1 through DBW9, DBWa through DBWz, and BW36 through BW99—to improve write performance if your system modifies data heavily. These additional DBW processes are not useful on uniprocessor systems.

The DBW process writes dirty buffers to disk under the following conditions:

When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBW to write. DBW writes dirty buffers to disk asynchronously if possible while performing other processing.

DBW periodically writes buffers to advance the checkpoint, which is the position in the redo thread from which instance recovery begins. The log position of the checkpoint is determined by the oldest dirty buffer in the buffer cache.

In many cases the blocks that DBW writes are scattered throughout the disk. Thus, the writes tend to be slower than the sequential writes performed by LGWR. DBW performs multiblock writes when possible to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

When a checkpoint is issued. Please see checkpoint process below.

When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers.

Every 3 seconds time-out.

When a log switch occurs

When Dirty-Blocks are becoming LRU Blocks

When Dirty-Blocks reaches to threshold value.

When Database shuts down.

5.1.5. Checkpoint Process

Process Name: CKPT

Max processes: 1

Checkpoint process signals the synchronization of all database files with the checkpoint information. It ensures data consistency and faster database recovery in case of a crash. CKPT ensures that all database changes present in the buffer cache at that point are written to the data files, the actual writing is done by the Database Writer process.

The datafile headers and the control files are updated with the latest SCN (when the checkpoint occurred) this is done by the log writer process. The CKPT process is invoked under the following conditions:

When a log switch is done.

When the time specified by the initialization parameter LOG_CHECKPOINT_TIMEOUT exists between the incremental checkpoint and the tail of the log; this is in seconds.

When the number of blocks specified by the initialization parameter LOG_CHECKPOINT_INTERVAL exists between the incremental checkpoint and the tail of the log; these are OS blocks.

The number of buffers specified by the initialization parameter FAST_START_IO_TARGET required to perform roll-forward is reached.

Oracle 9i onwards, the time specified by the initialization parameter FAST_START_MTTR_TARGET is reached; this is in seconds and specifies the time required for a crash recovery. The parameter FAST_START_MTTR_TARGET replaces LOG_CHECKPOINT_INTERVAL and FAST_START_IO_TARGET, but these parameters can still be used.

When the ALTER SYSTEM SWITCH LOGFILE command is issued.

When the ALTER SYSTEM CHECKPOINT command is issued.

Incremental Checkpoints initiate the writing of recovery information to datafile headers and control files. Database writer is not signaled to perform buffer cache flushing activity here.

5.1.6. Recoverer

Process Name: RECO

Max processes: 1

The Recoverer process is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions.

5.1.7 Listener Registration Process

Process Name: LREG

The Listener Registration Process (LREG) registers information about the database instance and dispatcher processes with the Oracle Net Listener. When an instance starts, LREG polls the listener to determine whether it is running. If the listener is running, then LREG passes it relevant parameters. If it is not running, then LREG periodically attempts to contact it.

NOTE: In releases before Oracle Database 12c, PMON performed the listener registration.

5.2 Optional Processes

5.2.1 Archiver

Process Name: ARC0 through ARC9

Max Processes: 10

The ARCh process is responsible for writing the online redo log files to the mentioned archive log destination after a log switch has occurred. ARCh is present only if the database is running in Archivelog mode and automatic archiving is enabled. The log writer process is responsible for starting multiple ARCh processes when the workload increases. Unless ARCh completes the copying of a redo log file, it is not released to log writer for overwriting.

The number of Archiver processes that can be invoked initially is specified by the initialization parameter LOG_ARCHIVE_MAX_PROCESSES. The actual number of Archiver processes in use may vary based on the workload. Arch can be stopped and started dynamically with alter system archive log stop/start or the SQL*PLUS command ARCHIVE LOG. In order to permanently have arch started when the database is started, log_archive_start must be set to true.

ARCh copies the online redo log a bit more intelligently than how the operating system command cp or copy would do: if a log switch is forced, only the used space of the online log is copied, not the entire log file.

5.2.2 Recovery Writer

Process Name: RVWR

This process is new optional process of the Oracle 10g database, is responsible for maintaining the before images of blocks in the flash recovery area used with the FLASHBACK DATABASE command.

Flashback a database means going back to a previous database state.

The Flashback Database feature provides a way to quickly revert an entire Oracle database to the state it was in at a past point in time.

This is different from traditional point in time recovery.

A new background process Recovery Writer (RVWR) introduced which is responsible for writing flashback logs which stores pre-image(s) of data blocks

One can use Flashback Database to back out changes that:

Have resulted in logical data corruptions.

Are results of user error?

This feature is not applicable for recovering the database in case of media failure.

The time required for flashback a database to a specific time in past is DIRECTLY PROPORTIONAL to the number of changes made and not on the size of the database.

5.2.3 Lock Monitor

Process Name: LMON

Max Processes: 1

Meant for Parallel server setups, Lock Monitor manages global locks and resources. It handles the redistribution of instance locks whenever instances are started or shutdown. Lock Monitor also recovers instance lock information prior to the instance recovery process. Lock Monitor co-ordinates with the Process Monitor to recover dead processes that hold instance locks.

5.2.4 Lock Manager Daemon

Process Name: LMDn

Meant for Parallel server setups, LMDn processes manage instance locks that are used to share resources between instances. LMDn processes also handle deadlock detection and remote lock requests.

5.2.5 Lock processes

Process Name: LCK0 through LCK9

Max Processes: 10

Meant for Parallel server setups, the instance locks that are used to share resources between instances are held by the lock processes.

5.2.6 Block Server Process

Process Name: BSP0 through BSP9

Max processes: 10

Meant for Parallel server setups, Block server Processes have to do with providing a consistent read image of a buffer that is requested by a process of another instance, in certain circumstances.

5.2.7 Queue Monitor

Process Name: QMN0 through QMN9

Max Processes: 10

This is the advanced Queuing Time manager process. QMNn monitors the message queues. Failure of QMNn process will not cause the instance to fail.

5.2.8 Event Monitor

Process Name: EMN0/EMON

Max Processes: 1

This process is also related to Advanced Queuing, and is meant for allowing a publish/subscribe style of messaging between applications.

5.2.9 Shared Server Processes

Process Name: Snnn

Max Processes: 1000

Intended for Shared server setups (MTS). These processes pickup requests from the call request queue process them and then return the results to a result queue. The number of shared server processes to be created at instance startup can be specified using the initialization parameter SHARED_SERVERS.

5.2.10 Memory Manager

Process Name: MMAN

This process is new optional background process of the Oracle 10g database. Memory manager used for internal database tasks. MMAN can **dynamically** adjust the sizes of the following SGA components:

- Database buffer cache
- Large pool
- Shared pool
- Java pool

5.2.11 Parallel Execution Slaves

Process Name: Pnnn

These processes are used for parallel processing. It can be used for parallel execution of SQL statements or recovery. The Maximum number of parallel processes that can be invoked is specified by the initialization parameter PARALLEL_MAX_SERVERS.

5.2.12 Trace Writer

Process Name: TRWR

Max Processes: 1

This process is new optional background process of the Oracle 10g database. Trace writer writes trace files from an Oracle internal tracing facility. When one of the Oracle background processes (such as dbwr, lgwr, pmon, smon and so on) encounters an exception, they will write a trace file. These trace files are also recorded in the alert.log. Trace files are also created for diagnostic dump events. An ORA-00600 error also produces a trace

5.2.13 DMON

A DMON process is started when the data guard monitor is started.

5.2.14 Dispatcher

Process Name: Dnnn

Intended for Shared server setups (MTS). Dispatcher processes listen to and receive requests from connected sessions and places them in the request queue for further processing. Dispatcher processes also pickup outgoing responses from the result queue and transmit them back to the clients. Dnnn are mediators between the client processes and the shared server processes. The maximum number of Dispatcher process can be specified using the initialization parameter MAX_DISPATCHERS.

5.2.15 MMON

This process is new optional background process of the Oracle 10g database and is responsible in associated with the Automatic Workload Repository new features used for automatic problem detection and self-tuning.

MMON writes out the required statistics for AWR on a scheduled basis. This process makes a snapshot of the database 'health' (statistics) and stores this information in the automatic workload repository. Performs various manageability-related background tasks.

5.2.16 Input/Output Slaves

Process Name: Innn

These processes are used to simulate asynchronous I/O on platforms that do not support it. The initialization parameter DBWR_IO_SLAVES is set for this purpose.

5.2.17 Wakeup Monitor Process

Process Name: WMON

This process was available in older versions of Oracle to alarm other processes that are suspended while waiting for an event to occur. This process is obsolete and has been removed.

5.2.18 Memory Monitor Light

Process Name: MMON

The Memory Monitor Light (MMNL) process works with the Automatic Workload Repository new features (AWR) to write out full statistics buffers to disk as needed. Performs frequent and lightweight manageability-related tasks, such as session history capture and metrics computation.

5.2.19 Job Queue Processes

Process Name: J000 through J999 (Originally called SNPn processes)

Max Processes: 1000

Job queue processes carry out batch processing. All scheduled jobs are executed by these processes. The initialization parameter JOB_QUEUE_PROCESSES specifies the maximum job processes that can be run concurrently.

If a job fails with some Oracle error, it is recorded in the alert file and a process trace file is generated. Failure of the Job queue process will not cause the instance to fail.

5.2.20 RBAL

This is the ASM related process that performs rebalancing of disk resources controlled by ASM.

5.2.21 ARBx

These processes are managed by the RBAL process and are used to do the actual rebalancing of ASM controlled disk resources. The number of ARBx processes invoked is directly influenced by the ASM_POWER_LIMIT parameter.

5.2.22 ASMB

The ASMB process is used to provide information to and from the Cluster Synchronization Services used by ASM to manage the disk resources. It is also used to update statistics and provide a heartbeat mechanism.

5.2.23 Change Tracking Writer

Process Name: CTWR

This is a new process Change Tracking Writer (CTWR) which works with the new block changed tracking features in 10g for fast RMAN incremental backups.

5.2.24 Job Queue Monitoring

Process Name: CJQn

This is the job queue monitoring process which is initiated with the JOB_QUEUE_PROCESSES parameter.

5.2.25 Auto BMR Background Process

Process Name: ABMR

Coordinates execution of tasks such as filtering duplicate block media recovery requests and performing flood control.

5.2.26 Automatic Block Media Recovery Slave Pool Process

Process Name: BMRn

Fetches blocks from a real-time readable standby database

5.2.27 Database Capture Process

Process Name: CPnn

Captures database changes from the redo log by using the infrastructure of LogMiner.

5.2.28 Database Resource Manager Process

Process Name: DBRM

Sets resource plans and performs other Resource Manager tasks.

5.2.29 Diagnostic Capture Process

Process Name: DIAG

Performs diagnostic dumps and executes global oradebug commands.

5.2.30 EMON Coordinator Process

Process Name: EMNC

Coordinates the event management and notification activity in the database, including Streams Event Notifications, Continuous Query Notifications, and Fast Application Notifications. Spawns ENNN processes.

5.2.31 RAT Masking Slave Process

Process Name: RM

This background process is used with Data Masking and Real Application Testing.

5.2.32 Space Management Coordinator Process

Process Name: SMCO

Coordinates the execution of various space management related tasks, such as proactive space allocation and space reclamation.

5.2.33 Virtual Scheduler for Resource Manager Process

Process Name: VKRM

Serves as centralized scheduler for Resource Manager activity

5.2.34 Virtual Keeper of Time Process

Process Name: VKTM

Responsible for providing a wall-clock time (updated every second) and reference-time counter (updated every 20ms and available only when running at elevated priority).

5.2.35 ASM Cluster File System CSS Process

Process Name: ACFS

Tracks the cluster membership in CSS and informs the file system driver of membership changes

5.3. Other Oracle Database Background Processes

There are several other background processes that might be running. These can include the following:

ACMS (atomic controlfile to memory service) per-instance process is an agent that contributes to ensuring a distributed SGA memory update is either globally committed on success or globally aborted in the event of a failure in an Oracle RAC environment.

ASMB is present in a database instance using an Automatic Storage Management disk group. It communicates with the Automatic Storage Management instance.

DBRM (database resource manager) process is responsible for setting resource plans and other resource manager related tasks.

DIA0 (diagnosability process 0) (only 0 is currently being used) is responsible for hang detection and deadlock resolution.

DIAG (diagnosability) process performs diagnostic dumps and executes global oradebug commands.

DSKM (slave diskmon) is used as the conduit between RDBMS and ASM instances and the Master Diskmon daemon to communicate I/O Fencing information, I/O Resource Manager Plans, and Transaction Commit Cache information to SAGE storage. It is also used to implement the skgxp ANT protocol between the host nodes and SAGE storage servers. If no SAGE storage is used, the slave diskmon process will exit silently after startup of the instance.

EMNC (event monitor coordinator) is the background server process used for database event management and notifications.

FBDA (flashback data archiver process) archives the historical rows of tracked tables into flashback data archives. Tracked tables are tables which are enabled for flashback archive. When a transaction containing DML on a tracked table commits, this process stores the pre-image of the rows into the flashback archive. It also keeps metadata on the current rows.

FBDA is also responsible for automatically managing the flashback data archive for space, organization, and retention and keeps track of how far the archiving of tracked transactions has occurred.

GMON maintains disk membership in ASM disk groups.

GTDX0-j (global transaction) processes provide transparent support for XA global transactions in an Oracle RAC environment. The database auto tunes the number of these processes based on the workload of XA global transactions. Global transaction processes are only seen in an Oracle RAC environment.

KATE performs proxy I/O to an ASM metafile when a disk goes offline.

MARK marks ASM allocation units as stale following a missed write to an offline disk.

MMAN is used for internal database tasks.

MMNL performs frequent and light-weight manageability-related tasks, such as session history capture and metrics computation.

MMON performs various manageability-related background tasks, for example:

Issuing alerts whenever a given metrics violates its threshold value

Taking snapshots by spawning additional process (MMON slaves)

Capturing statistics value for SQL objects which have been recently modified

ORBn performs the actual rebalance data extent movements in an Automatic Storage Management instance. There can be many of these at a time, called ORB0, ORB1, and so forth.

PSP0 (process spawner) spawns Oracle processes.

RBAL coordinates rebalance activity for disk groups in an Automatic Storage Management instance. It performs a global open on Automatic Storage Management disks.

SMCO (space management coordinator) process coordinates the execution of various space management related tasks, such as proactive space allocation and space reclamation. It dynamically spawns slave processes (Wnnn) to implement the task.

VKTM (virtual keeper of time) is responsible for providing a wall-clock time (updated every second) and reference-time counter (updated every 20 ms and available only when running at elevated priority).

ABMRP(Auto BMR Background Process)

Coordinates execution of tasks such as filtering duplicate block media recovery requests ABMRand performing flood control.

Xnnn (ASM Disk Expel Slave Process) Performs Oracle ASM post-rebalance activities.

VUBG (Volume drive Umbilicus Background) Relays messages between Oracle ASM instance and Oracle ASM Proxy instance that is used by ADVM (for ACFS).

VBGn (Volume Background Process) Communicates between the Oracle ASM instance and the operating system volume driver.

RPOP (Instant Recovery Repopulation Daemon) Performs monitoring management tasks related to Data Guard on behalf of DMON.

RCBG (Result Cache Background Process) Handles result cache messages.

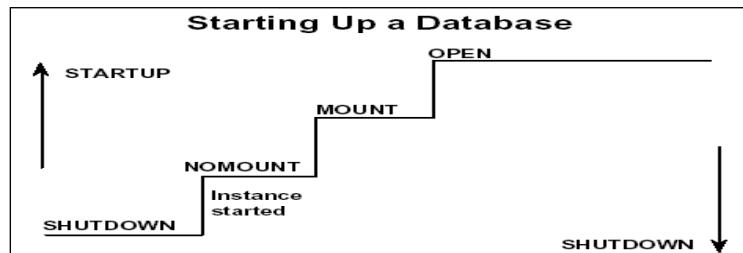
PSP0(Process Spawner Process) Spawns Oracle background processes after initial instance startup.

6. Startup & Shutdown Modes

6.1 Starting Up a Database

6.1.1 Instance Stages

Databases can be started up in various states or stages. The diagram shown below illustrates the stages through which a database passes during startup and shutdown.



6.1.1.1 NOMOUNT

This stage is only used when first creating a database or when it is necessary to recreate a database's control files. Startup includes the following tasks.

Read the spfileSID.ora or spfile.ora or initSID.ora.

Allocate the SGA.

Startup the background processes.

Open a log file named alert_SID.log and any trace files specified in the initialization parameter file.

Example startup commands for creating the Oracle database and for the database belonging to USER1 are shown here.

```

SQL> STARTUP NOMOUNT PFILE=$ORACLE_HOME/dbs/initoracle.ora
SQL> STARTUP NOMOUNT PFILE=$HOME/inituser1.ora
  
```

6.1.1.2 MOUNT

This stage is used for specific maintenance operations. The database is mounted, but not open. We can use this option if we need to:

Rename datafiles.

Enable/disable redo log archiving options.

Perform full database recovery.

When a database is mounted it

- Is associated with the instance that was started during NOMOUNT stage.
- Locates and opens the control files specified in the parameter file.
- Reads the control file to obtain the names/status of and redo log files, but it does not check to verify the existence of these files.

Example startup commands for maintaining the Oracle database and for the database belonging to USER1 are shown here.

```

SQL> STARTUP MOUNT PFILE=$ORACLE_HOME/dbs/initoracle.ora
SQL> STARTUP MOUNT PFILE=$HOME/inituser1.ora
  
```

Mount Options

EXCLUSIVE: Permits only the current instance to access the Database.

PARALLEL: Allows multiple instances to access the Database.

SHARED: Same as PARALLEL, different name

RETRY:(num) specifies that a parallel instance should retry to startup a (num) second intervals

6.1.1.3 OPEN

This stage is used for normal database operations. Any valid user can connect to the database. Opening the database includes opening and redo log files. If any of these files are missing, Oracle will return an error. If errors occurred during the previous database shutdown, the SMON background process will initiate instance recovery. An example command to startup the database in OPEN stage is shown here.

```
SQL> STARTUP PFILE=$ORACLE_HOME/dbs/initoracle.ora
SQL> STARTUP PFILE=$HOME/inituser1.ora
```

If the database initialization parameter file is in the default location at \$ORACLE_HOME/dbs, then we can simply type the command STARTUP and the database associated with the current value of ORACLE_SID will startup.

6.1.2 Startup Command Options

We can force a restart of a running database that aborts the current instance and starts a new normal instance with the FORCE option.

```
SQL> STARTUP FORCE PFILE=$HOME/inituser1.ora
```

Sometimes we will want to startup the database, but restrict connection to users with the RESTRICTED SESSION privilege so that we can perform certain maintenance activities such as exporting or importing part of the database.

```
SQL> STARTUP RESTRICT PFILE=$HOME/inituser1.ora
```

We may also want to begin media recovery when a database starts where our system has suffered a disk crash.

```
SQL> STARTUP RECOVER PFILE=$HOME/inituser1.ora
```

On a LINUX server, we can automate startup/shutdown of an Oracle database by making entries in a special operating system file named oratab located in the /var/opt/oracle directory.

Note: If an error occurs during a STARTUP command, we must issue a SHUTDOWN command prior to issuing another STARTUP command.

6.2 Restricted Mode

Earlier we learned to startup the database in a restricted mode with the RESTRICT option. If the database is open, we can change to a restricted mode with the ALTER SYSTEM command as shown here. The first command restricts logon to users with restricted privileges. The second command enables all users to connect.

```
SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;
SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

One of the tasks we may perform during restricted session is to kill current user sessions prior to performing a task such as the export of objects (tables, indexes, etc.). The ALTER SYSTEM KILL SESSION 'integer1, integer2' command is used to do this. The values of integer1 and integer2 are obtained from the SID and SERIAL# columns in the V\$SESSION view.

The first six SID values shown below are for background processes and should be left alone! Notice that the users SYS and USER1 are connected. We can kill the session for user account name DBOCKSTD.

```

SQL> SELECT sid, serial#, status, username FROM v$session WHERE
      username='DBOCKSTD';

  SID      SERIAL#      STATUS      USERNAME
-----  -----  -----
  260        1352    INACTIVE    DBOCKSTD

SQL> ALTER SYSTEM KILL SESSION '260, 1352';

System altered.

```

Now when DBOCKSTD attempts to select data, the following message is received.

```

SQL> select * from newspaper;
select * from newspaper
*
ERROR at line 1:
ORA-00028: your session has been killed

```

When a session is killed, PMON will rollback the user's current transaction and release all table and row locks held and free all resources reserved for the user.

6.3 Database Shutdown

The SHUTDOWN command is used to shutdown a database instance. We must be connected as either SYSOPER or SYSDBA to shutdown a database.

6.3.1 Shutdown Normal

This is the default shutdown mode.

No new connections are allowed.

The server waits for all users to disconnect before completing the shutdown.

Database and redo buffers are written to disk.

The SGA memory allocation is released and background processes terminate.

The database is closed and dismounted.

The shutdown command is:

SHUTDOWN (or) SHUTDOWN NORMAL

6.3.2 Shutdown Transactional

This prevents client computers from losing work.

No new connections are allowed.

No connected client can start a new transaction.

Clients are disconnected as soon as the current transaction ends.

Shutdown proceeds when all transactions are finished.

The shutdown command is:

SHUTDOWN TRANSACTIONAL

6.3.3 Shutdown Immediate

This can cause client computers to lose work.

No new connections are allowed.

Connected clients are disconnected and SQL statements in process are not completed.

Oracle rolls back active transactions.

Oracle closes/dismounts the database.

The shutdown command is:

SHUTDOWN IMMEDIATE

6.3.4 Shutdown Abort

This is used if the normal or transactional or immediate options fail. This is the LEAST favored option because the next startup will require instance recovery and we CANNOT backup a database that has been shutdown with the ABORT option.

Current SQL statements are immediately terminated.

Users are disconnected.

Database and redo buffers are not written to disk.

Uncommitted transactions are not rolled back.

The Instance is terminated without closing files.

The database is not closed or dismounted.

Database recovery by SMON must occur on the next startup.

There are different modes in Oracle Startup procedure.

Different Modes Of Startup

STARTUP NOMOUNT	Mounts the instance. (to create controlfiles or to create a database)
STARTUP MOUNT	Mounts the instance and mounts the database. (To perform Media recovers or to "STOP" or "START" Archivelog.)
STARTUP	Mounts the instance, mounts the database and opens the database.
STARTUP MOUNT EXCLUSIVE	Mounts the database in exclusive mode (Oracle Parallel Server).

Different Modes Of Shutdown

SHUTDOWN	Shuts down normally. (Waits for all users to disconnect)
SHUTDOWN TRANSACTIONAL	Shuts down when all the users active
SHUTDOWN IMMEDIATE	Rolls back any uncommitted transactions and shuts down. In the industry "normal" shutdown will never work you have to use "immediate" all the time. But both "normal" & "immediate" will ensure redo-log entries (committed ones) are written to DATABASE Files.
SHUTDOWN ABORT	Abruptly shuts down the database, which demands a crash recovery when the Inst. coming up next time (SMON). Usually we should never go for "abort" unless "immediate" also doesn't work.

7. Database Creation

Consider the following points before creating a brand-new Oracle DATABASE.

What is the application for this DATABASE? How many applications it's going to support? Advantages of having each application on each DATABASE.

If one DATABASE is down, only that application's end-users are down, where the other DATABASE is still up and people are using it.

If the company grows big (same with DATABASE), then we can easily move one DATABASE from this Sever to another.

Whenever we are going to take a cold backup, we need to bring down one DATABASE only, and have users & jobs run on the other DATABASE, where if we have 2 APPS on same DATABASE, then both users have to go down.

One greatest adv. in having more DATABASES is, if the size of DATABASE is too big, at times we may not be able to backup in 8 hours span. So by having two databases, we may have to backup these databases on alternative days.

Advantages of having a Common Database for all Applications:

We need to worry about 1 DATABASE Only. So that our log files are not many to monitor.

Integration is much easier (although it's possible even with Multiple, still we are hoping the other DATABASE is up), since all APPS are located in one DATABASE.

When upgrading the Oracle Versions, we need to worry about only one DATABASE.

So we need to decide on what is going on with this DATABASE and how many users we need to support.

7.1 Understanding Oracle Tablespaces:

Every Oracle DATABASE has to have at least two Tablespaces called "SYSTEM" and "SYSAUX". In the SYSTEM TS, Oracle stores all the information of Oracle-Base-Tables (e.g. users, tables, indexes etc.) and in the SYSAUX tablespace, objects of non-essential Schema.

For every user action, SYSTEM Tablespace is getting hit and this is why we never prefer to keep user tables in SYSTEM-TS (since it's already busy in providing information to all connected users from OBT).

So for performance reasons, we create

```
SYSTEM-TS      (for OBT) --- gets created automatically when the DATABASE
is created
SYSAUX-TS      (for non-essential schema objects) --- gets created
automatically with the DB
USER_DATA      (for User Tables)
USER_INDEX      (for User Indexes)
UNDO_TS        (for keeping UNDO SEGMENTS)
TEMP_TS        (for Temporary Segments -- gets created when sorting
happens)
```

If you create the above Tablespaces on different HD, you'll get better performance results since you are avoiding unnecessary disk-contentions.

In the **init<SID>.ora** we need to set a parameter called "PROCESSES". Each user will have his/her own background-processes (at least in dedicated server process).

SGA Size: This is the MEM required by Oracle for this Instance and it is up to OS to offer or not depending on avail of Physical - Memory and Kernel-Configuration of OS. But once OS allocates the MEM to Oracle-Instance, it can't be taken back - not even swappable by OS, until the Instance comes down-nothing but "shutting down" the database. (SGA -> System Global Area). This SGA is

unique to each Oracle-Instance and can't be shared between Oracle-Instances. In calculating www.wilshiresoft.com

SGA size we always go with

```
DB_CACHE_SIZE, DB_nK_CACHE_SIZE (where n=2, 4, 8, 16, 32)
SHARED_POOL
LOG_BUFFER
JAVA_POOL_SIZE
LARGE_POOL_SIZE
```

Apart from the above five parameters, "PROCESSES" also has its own effect on SGA Size. All above-mentioned parameters must be defined in INIT<SID>.ORA file, which is different for each Instance.

```
How many Apps: So that we can come out our "Tablespace" naming
conventions. For e.g. in this DATABASE, we have 2 Applications: PAY &
ACCT. So our naming conventions apart from SYSTEM_TS are:
PAY_USER_DATA      (Payroll)
PAY_USER_INDX       (Payroll)
ACCT_USER_DATA      (ACCT)
ACCT_USER_INDX      (ACCT)
```

How much SGA per each DATABASE: This depends on various factors like:

How many users for each DATABASE

How much memory available in the server

Is the server dedicated to Oracle or not?

How big the Oracle BLOCK_SIZE should be: This depends on the ROW_SIZE. So we usually have 10-15% tables, which are active on daily-basis. So on these Active-Tables, we do

```
SQL> ANALYZE TABLE EMP COMPUTE STATISTICS;
SQL> SELECT AVG_ROW_LEN FROM USER_TABLES WHERE TABLE_NAME = 'EMP';
To analyze all tables of the schema we can perform:
SQL> EXECUTE DBMSUTILITY.ANALYZE_SCHEMA ('SCOTT', 'COMPUTE');
```

Analyze the table and find out what is "AVG_ROW_LEN". If the AVG_ROW_LEN is above our ORACLE BLOCK SIZE, then this row is obtaining more than 1 block, meaning that Oracle has to get more than 1 block into SGA. So we come out with a better OBS, which is big enough to fit the AVG_ROW_LEN. Oracle Block Size cannot be changed after the DATABASE got created.

The only way you can change it is, drop the DATABASE and re-create it with modified OBS. The default OBS (on most Platforms) is 8K. You can choose OBS --> 2k, 4k, 8k, 16k, 32k (nothing but 2-power-n where n ranges from 1-5). For most applications 4K is more than sufficient. But in big apps like SAP, since there are 50-60 columns in each row, they may demand the Oracle Block Size of 8K or 16K.

In a server where you are running multiple Databases, each DATABASE can have different Oracle Block Size (depending on the average row length of each DATABASE).

The most commonly used block size should be picked as the STANDARD BLOCK SIZE. This is done typically by specifying the DB_BLOCK_SIZE initialization parameter.

To use NON-STANDARD BLOCK SIZES, you must configure sub-caches within the buffer cache area of the SGA memory for all of the non-standard block sizes that you intend to use. The ability to specify multiple block sizes is especially useful if you are transporting tablespaces between databases.

The DB_CACHE_SIZE initialization parameter specifies the size of the cache of standard block size buffers. The sizes and numbers of non-standard block size buffers are specified by the following init<SID>.ora parameters

```
DB_2K_CACHE_SIZE = 4M
DB_4K_CACHE_SIZE = 4M
```

Do we have a separate Development Server where we'll have development DATABASE (we should never let developers touch our Production DATABASE-Server)?

How big is our Tape drive?

What is the potential of this App? (Meaning the growth-rate)

DATABASE-Size estimations (depending on number of records coming-in daily)

Type of the DATABASE: Is it Data warehousing or Transactional? Similarly is it Insert oriented or Update oriented?

How often our developers are releasing new versions? (Updates to DATABASE - DDL)

Is it a Corporate-only-DATABASE or same DATABASE is existing in multiple locations (all belong to the same company).

Do we have any distributed transactions?

Any tables need to be replicated to other sites?

Is it 24/7 DB (or can you take it down in nights for Backup purposes)

What support-level do you have from Oracle (Bronze, Silver, Gold, and Platinum?)

How this DATABASE is accessed? Is it from Dumb-Terminals or Client/Server or through WEB?

Do we need to configure our Clients with SQL*Net or are they Browser-Based?

```
Using Dumb-Term      ---> Local computing / Host-based
Client/Server ---> VB App      ---> using ODBC drivers
Web based           ---> Java App --> using JDBC Drivers
```

After startup the instance in nomount state then oracle creates some files. They are

Alert<SID>.log --> a logfile contains all-important info about the instance including switch times and startup & shutdown events and also if any rollback segment problems etc.

Trace files --> these are generated by Oracle BPs and are denoted in the alert file whenever there is problem with any process.

Oracle has so many sub-directories in ORACLE_HOME. For e.g. ORACLE_HOME/bin, / dbs, /rdbms, /sqlplus, /network, etc.

NOTE: For easy management, Oracle11g recommends to create the UNDO TABLESPACE and TEMPORARY TABLESPACE at the time of database creation only.

After completion of database creation and required OBT which are defined in \$ORACLE_HOME/rdbms/admin/sql.bsq file. So all the required tables are already there and DATABASE is in 100% functional condition. Which means users can be created and they can login and create tables etc. But the environment is not so user-friendly (e.g. you don't have SELECT * FROM TAB ;), since the synonym "TAB" is not yet created.

You have only OBT, but not any views or synonyms. These OBT are very hard to understand even for a DBA. So what we do? We run 'CATALOG.SQL' which created the required views & synonyms that are very user-friendly.

By default Oracle will not create any of the PL*SQL packages (e.g. DBMS_JOBS). In order to take advantage of Oracle's pre-created packages, we need to run 'CATPROC.SQL' and 'CATBLOCK.SQL' also

Finally if you want to create profiles, you may run 'PUPBLD.SQL'

This completes basic database creation. So the actual problems start now. Since we know our applications that we are going to support on this DATABASE, let's plan out our TS structures with multiple block sizes. By default when you create a tablespace in 11g, it takes the extent management as local. Tablespace is something where we are carrying our Tables, Indexes, and Temporary Segments & UNDO Segments.

```
SQL> CREATE TABLESPACE USER_DATA
      DATAFILE '/disk1/oradata/ORCL/user_data_01.dbf' size 10M;
```

Similarly we can create other Tablespaces. In case if we are running out of space in any TS, we can add a second datafile to the same TS. Each datafile can only represent one & only one TS, where as a TS can be made up of at least 1 or more. If you don't like to add more & more, you can take advantage of Oracles new feature called "Dynamic Datafile Extension" which means, the datafile can be extended depending upon the need (here, one thing should be kept in your mind which is the space you have at OS-Level - in the UNIX-File system).

So you can limit the growth of this datafile to a certain point. After that it'll stop extending. But it is 1-way traffic (although using some other commands we can RESIZE the datafile and make it smaller). Here is an example for adding a second datafile to the existing TS.

```
SQL> ALTER TABLESPACE USER_DATA ADD
      DATAFILE '/disk1/oradata/ORCL/user_data_02.dbf' SIZE 10M;
```

As a DBA, we are interested in knowing how our TS's are doing every day. So we take our unix-cron help to print out these kinds of reports on daily-basis. Some of the important view types we need to know are: (1. ALL_, 2. DBA_, 3. USER_).

Every user has access to USER_ and also ALL_ views). Apart from these views, we have dynamic

performance views --> V\$). We also have some tables which end with "\$" and are known as "Oracle Base Tables" e.g. TAB\$, TS\$, VIEW\$...

Starting the Oracle Instance: You have set your ORACLE_SID environment variable and fire-up sqlplus /nolog, and then 'connect / as sysdba' and then start the Instance. Instance means 3 things:

DB-Files , BPs, SGA.

8. Tablespace Management

A **Tablespace** is a logical storage unit within an Oracle database. It is logical because a table space is not visible in the file system of the machine on which the database resides. A table space, in turn, consists of at least one datafile which, in turn, are physically located in the file system of the server. A datafile belongs to exactly one tablespace.

Each table, index and so on that is stored in an Oracle database belongs to a tablespace. The tablespace builds the bridge between the Oracle database and the filesystem in which the table's or index' data is stored.

8.1 Dictionary Managed Tablespace

Tablespaces that record extent allocation in the dictionary are called **Dictionary Managed Tablespaces (DMT)**. With this approach the data dictionary contains tables that store information that is used to manage extent allocation and deallocation manually.

Extents are managed in the data dictionary

Each segment stored in the tablespace can have different storage clause

Coalescing is required

Example

```
SQL>CREATE TABLESPACE user_data
  DATAFILE
  '/disk1/oradata/wilshire/userdata01.dbf'
  SIZE 50M EXTENT MANAGEMENT DICTIONARY
  DEFAULT STORAGE (INITIAL 50K NEXT 50K MINEXTENTS 2 MAXEXTENTS 50
  PCTINCREASE 0);
```

The DEFAULT STORAGE clause enables you to customize the allocation of extents. This provides increased flexibility, but less efficiency than locally managed tablespaces.

The tablespace will be in a single, 50M datafile.

The EXTENT MANAGEMENT DICTIONARY clause specifies the management.

All segments created in the tablespace will inherit the default storage parameters unless their storage parameters are specified explicitly to override the default.

The storage parameters specify the following:

INITIAL – size in bytes of the first extent in a segment.

NEXT – size in bytes of second and subsequent segment extents.

PCTINCREASE – percent by which each extent after the second extent grows.

- SMON periodically coalesces free space in a dictionary-managed tablespace, but only if the PCTINCREASE setting is NOT zero.

Use ALTER TABLESPACE <tablespacename> COALESCE to manually coalesce adjacent free extents.

MINEXTENTS – number of extents allocated at a minimum to each segment upon creation of a segment.

MAXEXTENTS – number of extents allocated at a maximum to a segment – you can specify UNLIMITED.

8.2 Locally Managed Tablespace

Tablespaces that record extent allocation in the tablespace header are called **Locally Managed Tablespaces (LMT)**. The extents allocated to a locally managed tablespace are managed through the use of **bitmaps**. Locally Managed Tablespace (LMT) is one of the key features in Oracle database. These have been made available since Oracle 8i. It is worth using LMTS considering the benefits in doing so.

Each bit corresponds to a block or group of blocks (an extent).

The bitmap value (on or off) corresponds to whether or not an extent is allocated or free for reuse.

Reduced contention on data dictionary tables

No undo generated when space allocation or deallocation occurs

No coalescing requiredLocal management is the default for the SYSTEM tablespace beginning with Oracle 12C.

If the SYSTEM tablespace is locally managed, the other tablespaces must also be either locally managed or read-only.

Local management reduces contention for the SYSTEM tablespace because space allocation and deallocation operations for other tablespaces do not need to use data dictionary tables.

The LOCAL option is the default so it is normally not specified.

With the LOCAL option, you cannot specify any DEFAULT STORAGE, MINIMUM EXTENT, or TEMPORARY clauses.

UNIFORM – a specification of UNIFORM means that the tablespace is managed in uniform extents of the SIZE specified.

Use UNIFORM to enable exact control over unused space and when you can predict the space that needs to be allocated for an object or objects.

Use K, M, G, etc to specify the extent size in kilobytes, megabytes, gigabytes, etc. The default is 1M; however, you can specify the extent size with the SIZE clause of the UNIFORM clause.

AUTOALLOCATE – a specification of AUTOALLOCATE instead of UNIFORM, then the tablespace is system managed and you cannot specify extent sizes.

AUTOALLOCATE is the default - this simplifies disk space allocation because the database automatically selects the appropriate extent size.

AUTOALLOCATE - this does waste some space but simplifies management of tablespace.

- Tablespaces with AUTOALLOCATE are allocated minimum extent sizes of 64K – dictionary-managed tablespaces have a minimum extent size of two database blocks.

8.2.1 Benefits of LMTS

Below are the key benefits offered by LMTS. Not all are achievable when migrating to LMTS.

- Dictionary contention is reduced.Extent management in DMTs is maintained and carried out at the data dictionary level. This requires exclusive locks on dictionary tables. Heavy data processing that results in extent allocation/deallocation may sometimes result in contentions in the dictionary. Extents are managed at the datafile level in LMTS. Dictionary tables are no longer used for storing extent allocation/deallocation information. The only information still maintained in the dictionary for LMTS is the tablespace quota for users.
- Space wastage removed.In DMTs, there is no implied mechanism to enforce uniform extent sizes. The extent sizes may vary depending on the storage clause provided at the object level or the tablespace level, resulting in space wastage and fragmentation. Oracle enforces the uniform extents allocation in the LMTS (when created with UNIFORM SIZE clause). Space wastage is removed, as this would result in all the same sized extents in the tablespace.
- No Rollback generated.
 - In DMTs, all extent allocations and deallocations are recorded in the data dictionary. This generates undo information thus using vital resources and may compete with other processes.
 - In LMTS, no rollback is generated for space allocation and deallocation activities.

ST enqueue contention reduced. In DMTs, Space Transaction (ST) enqueue is acquired when there is a need for extent allocations in DMTs. It is also exclusively acquired by SMON process for coalescing free space in DMTs. Only one such enqueue exists per instance, and may sometimes

result in contention and performance issues if heavy extent processing is being carried out. The following error is common in such scenario.

ORA-01575: timeout warning for space management resource
As ST enqueue is not used by LMTS it reduces the overall ST enqueue contention.

Recursive space management operations removed.
In DMTs, SMON process wakes up every 5 minutes for coalescing free space in DMTs. Optionally, the ALTER TABLESPACE <tablespace name> COALESCE command is also used to coalesce DMTs and reduce fragmentation.

On the other hand, LMTS avoid recursive space management operations and automatically track adjacent free space, thus eliminating the need to coalesce free extents. This further reduces fragmentation.

Fragmentation reduced.
Fragmentation is reduced in LMTS but not completely eliminated. Since adjacent free spaces are automatically tracked, there is no need to do coalescing, as is required in the case of DMTs.

8.2.2 Allocation Types in LMTS

Allocation type plays a very important role in how the LMT is behaving. It specifies how the extent is being allocated by the system. There are three types of allocating extents in LMTS- USER, SYSTEM and UNIFORM.

USER-The LMT behaves as DMT, allocating extents as per the storage clause provided with the object or defaulted at tablespace level. The advantage is that allocation of extents is managed at the datafile level and such tablespaces will not compete for ST enqueue.

The disadvantage is that such tablespaces are not subject to uniform extent allocation policy. DMTs that are converted to LMTS fall under this type.

SYSTEM-Oracle manages the space. The extents are auto allocated by the system based on an internal algorithm. Allocation of extents is managed at the datafile level and such tablespaces will not compete for ST enqueue.

Such tablespaces would have extents of varying sizes and would result in fragmentation and some space being wasted. This is a good alternative if the extent sizes of the various objects to be placed in the tablespace cannot be determined.

UNIFORM-All extents are of fixed size in the system. The size is provided when creating the LMT. This type gives all the benefits offered by LMT and one should aim at achieving this.

8.2.3 Checking space availability in LMTS

The existing DBA_FREE_SPACE is still available for checking available space in LMT AND DMT tablespaces. Specifically, two more views were introduced by Oracle - DBA_LMT_FREE_SPACE and DBA_DMT_FREE_SPACE. These views show the available blocks that should be multiplied with the block size to get the total bytes.

```
SQL> select name, (sum (a.blocks * 8192))/1024/1024 "size MB"
      from dba_lmt_free_space a, v$tablespace b
     where a.tablespace_id = b.ts#
   group by name;
SQL> select name, (sum(a.blocks * 8192))/1024/1024 "size MB"
      from dba_dmt_free_space a, v$tablespace b
     where a.tablespace_id = b.ts#
   group by name;
```

8.2.4 LMTS with Automatic Segment Space Management

When you create a locally managed tablespace using the CREATE TABLESPACE statement, the SEGMENT SPACE MANAGEMENT clause allows you to specify how free and used space within a segment is to be managed. You can either specify:

MANUAL: Specifying these keywords tells Oracle that you want to use free lists for managing free space within segments. Free lists are lists of data blocks that have space available for inserting rows.

AUTO: This keyword tells Oracle to use bitmaps to manage the free space within segments. A bitmap is a map that describes the status of each data block within a segment with respect to the amount of space in the block available for inserting rows.

Free lists have been the traditional method of managing free space within segments. Bitmaps, however, provide a simpler and more efficient way of managing segment space. They provide better space utilization and completely eliminate any need to specify and tune the PCTUSED, FREELISTS, and FREELISTS GROUPS attributes for segments created in the tablespace. AUTO is the default.

Note: For LOBs, you cannot specify automatic segment-space management.

- To create a locally managed permanent tablespace with auto allocate feature:

```
SQL> create tablespace lmtauto
      Datafile '/disk1/oradata/ORCL/lmtauto01.dbf' size 10m
      Extent management local autoallocate;
```

- To create a locally managed temporary tablespace with uniform feature:

```
SQL> create temporary tablespace lmttemp
      Tempfile '/disk1/oradata/ORCL/lmttemp01.dbf' size 10m
      Extent management local uniform size 64k;
```

To view information about tablespaces query dba_tablespaces:

```
SQL> select tablespace_name, extent_management, allocation_type
      from dba_tablespaces;
```

To view information about files related to temporary tablespace query v\$tempfile;

```
SQL> select * from v$tempfile;
```

8.3 BIGFILE Tablespaces - (From 10g)

As the quality and quantity of storage media continues to grow, Oracle 10g has kept pace as well. One of the major improvements in Oracle 11g is the introduction of new tablespace storage capacity. A tablespace is limited to a maximum of 65,536 (64K), but each datafile's size had been limited by the maximum OS file size. The new BIGFILE tablespace overcomes this limitation by allowing the creation of a tablespace with only one datafile that could be as big as 128 terabytes (TB).

A BIGFILE tablespaces datafile can now hold up to 4,294,967,296 (over 4 billion) blocks of data storage space. This means that a tablespace could conceivably hold up to 128 TB of data if the maximum Oracle tablespace block size of 32K is used. Oracle does recommend that Automatic Segment Space Management (ASSM) should be enabled.

In addition, if the BIGFILE tablespaces datafile is sized at its maximum of 128TB, then the UNIFORM extent management method will most likely yield better performance, especially when a large extent size is chosen. BIGFILE tablespaces (also known as BFTs) are therefore obviously aimed at Storage Area Network (SAN) environments on which Oracle's new Automatic Storage Management (ASM) feature has been enabled. The resulting increase in a database's potential size is staggering: If enough disk space is available, Oracle can now handle up to 8 exabytes of data (65,536 BIGFILE x 128TB per datafile).

Since BIGFILE tablespaces are allowed to own only one datafile, there is now a one-to-one correspondence between tablespace and datafile, and database space management actually becomes much simpler because all space management operations that were only limited to now extend automatically to BIGFILE tablespaces. Now you can resize BIGFILE tablespace by issuing

the `ALTER TABLESPACE <bft_name> RESIZE <new_size>;` command.

Migrating segments to a BIGFILE tablespace is accomplished using the same methods already available to move tables and indexes to different tablespaces. To migrate a table, the DBA only has to issue the `ALTER TABLE <table_name> MOVE TABLESPACE <bft_name>;` command. Issuing the `ALTER INDEX <index_name> TABLESPACE <bft_name> REBUILD ONLINE;` command will migrate an index to a BIGFILE tablespace.

Finally, note that a pre-Oracle 10g "normal" tablespace is now called a SMALLFILE tablespace and that by default Oracle will continue to create SMALLFILEs unless otherwise directed. The below example shows how to create a BIGFILE tablespace with the maximum number of allowable blocks, as well as change the database to create BIGFILEs instead of SMALLFILEs for all future tablespaces.

8.3.1 Creating a BIGFILE Tablespace

Set the database default to create BIGFILE tablespaces (SMALLFILE is the initial setting unless changed)

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

Create a sample BIGFILE tablespace. Note that the BIGFILE directive isn't necessary since the default tablespace type has been set for the database

```
SQL> DROP TABLESPACE big_lmpt INCLUDING CONTENTS AND DATAFILES;
SQL> CREATE TABLESPACE big_lmpt
      DATAFILE '/disk1/oradata/wilshire/big_lmpt01.dbf'
      SIZE 20M REUSE
      EXTENT MANAGEMENT LOCAL
      UNIFORM SIZE 100K
      SEGMENT SPACE MANAGEMENT AUTO;
```

Moreover, the below example shows the changes that have been made to the appropriate dynamic and static data dictionary views so that it is easy to determine the status of BIGFILE vs. SMALLFILE tablespaces.

8.3.2 Querying Oracle 10g tablespace properties and features

Show the default tablespace type (BIGFILE vs. SMALLFILE) for the database

```
SQL> SELECT property_name,property_value
      FROM database_properties
      WHERE property_name = 'DEFAULT_TBS_TYPE';
```

Are there any existing BIGFILE tablespaces?

```
SQL> SELECT tablespace_name,status,contents,bigfile,extent_management
      FROM dba_tablespaces
      ORDER BY tablespace_name;
SQL> SELECT name,bigfile,flashback_on
      FROM v$tablespace
      ORDER BY name;
```

8.3.3 Database-Level Default Permanent and Temporary Tablespaces

If you would have to spend time cleaning up after your colleagues when they have forgotten to assign a default permanent tablespace or (even worse!) a default temporary tablespace to a newly created user account. If this happened in prior releases, Oracle would assign the SYSTEM tablespace as the default and temporary tablespaces if both were omitted during user account creation.

Fortunately, Oracle 10g fill this gap by offering the ability to specify the default permanent and temporary tablespace for any newly created user accounts. If you are creating a new database using the Database Configuration Assistant (DBCA), Oracle will prompt you for these values and automatically create appropriate default permanent and temporary tablespaces. Of course, these tablespaces can also be specified in database creation scripts; for a converted Oracle 10g database, they can be updated via the appropriate `ALTER DATABASE` commands. See the below

example for database creation script for a new database and examples of modifying these settings for an existing database.

8.3.4 Altering Database-Level Permanent and Temporary Tablespaces

Default permanent tablespace and default temporary tablespace can be assigned to a database at the time its creation with CREATE DATABASE command or later we change the default tablespaces (Permanent or Temporary).

Alter an existing database to specify a default Permanent Tablespace. The tablespace must already exist!

```
SQL> ALTER DATABASE DEFAULT TABLESPACE userdata;
```

Alter an existing database to specify a default Temporary Tablespace. The tablespace must already exist!

```
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp;
```

8.4 SYSAUX Tablespace - (From 10g)

One of the major changes to the database in Oracle 10g is probably one of the more unnoticed unless you have had to convert a pre-Oracle 10g database to 10g, that is! Oracle 10g shifted a majority of schemas and their corresponding database objects to a new tablespace, SYSAUX, which must be created before a 10g conversion can proceed.

Prior to Oracle 10g, overabundance of different tablespaces was created whenever certain special features were requested for inclusion in an Oracle database. These included the database objects to enable capabilities like OLAP, Data Mining, XML Database (XMLDB), Oracle Enterprise Manager (OEM), and the Recovery Catalog.

If an Oracle DBA decided to install the optional STATSPACK features, another new schema (PERFSTAT) was created, and Oracle also recommended creating yet another new tablespace to hold the volume of raw data that facilitated STATSPACK reporting from multiple statistical snapshots.

Oracle 10g overcomes this confusion with the SYSAUX tablespace, a centralized repository that stores all metadata and database objects for all special database features. The SYSAUX tablespace is created automatically when using the Database Configuration Assistant (DBCA) to create a new database, and it can be added to an existing pre-10g Oracle database during its upgrade to 10g with the Database Upgrade Assistant (DBUA) as well as via standard SQL commands.

Since Oracle 10g features like Automatic Database Diagnostic Monitor (ADDM), Automatic Workload Repository (AWR), and the various Advisors all require regular and timely collection of database resource usage information, the SYSAUX tablespace facilitates centralized storage of usage statistics. Most importantly, however, is a reduction in the load on the SYSTEM tablespace. Prior to Oracle 10g, many of the optional features actually used the SYSTEM tablespace to store objects and related metadata.

With the implementation of SYSAUX, however, the primary functions of the SYSTEM tablespace - handling requests from the cost-based optimizer for data dictionary statistics and the storage of PL/SQL system-level objects - are now no longer overloaded by requests for other non-critical metadata and objects.

The loss of the SYSAUX tablespace's datafile is not a critical media failure, by the way. Unlike the loss of **either the SYSTEM or UNDO tablespaces', the database will continue to run without SYSAUX**. However, since so many utility applications depend on the objects stored within SYSAUX - especially the SYSMAN schema, which contains all of the objects needed to interface to Oracle 10g's Enterprise Manager - it is important to include SYSAUX in your database disaster recovery planning.

Don't Recreate Tablespace, Rename It!

Prior to Oracle 10g, if you want to move objects from, say, a locally managed tablespace with UNIFORM extent management to one with AUTOMATIC extent management, you had to follow a rather indirect path:

- Create a staging tablespace of similar size to the current tablespace.
- Move all objects from the current tablespace to the staging tablespace.
- Drop the current tablespace.
- Recreate the current tablespace with its new name and storage parameters.
- Move all objects from the staging tablespace back to the newly recreated tablespace.
- Drop the staging tablespace.
- Oracle 10g has done away with a third of this process with the new tablespace rename command, `ALTER TABLESPACE <old_tablespace_name> RENAME TO <new_tablespace_name>`.
- Create a new tablespace with the desired storage parameters.
- Move all objects from the current tablespace to the new tablespace.
- Drop the current tablespace.
- Rename the new tablespace to the same name as the original tablespace.

Warning: Before proceeding with a tablespace rename operation, it is important to remember these restrictions:

- SYSTEM and SYSAUX. These two tablespaces can never be renamed, for obvious reasons!
- Offline Datafiles. For a tablespace rename operation to succeed, all of the targeted tablespace's must be online.
- Tablespaces in READ ONLY Mode. A tablespace rename operation does not affect the datafile header of a read-only tablespace.

With these notable exceptions, any tablespace - whether permanent or temporary - can be renamed. Oracle automatically updates any references to the renamed tablespace in the tablespace's datafile headers, in the control file, and in the data dictionary. If the renamed tablespace was assigned as a default or temporary tablespace to a user account, Oracle also renames it for that account. Finally, an online UNDO tablespace can also be renamed; once the UNDO tablespace has been renamed, Oracle also handles the propagation of the renamed undo tablespace's new name to the server parameter file (SPFILE).

8.5 Tablespace Compression- (New in 11g R2)

You can specify that all tables created in a tablespace are compressed by default. You specify the type of table compression using the `DEFAULT` keyword, followed by one of the compression type clauses used when creating a table.

8.5.1 Compression for Table Data

Oracle has been a pioneer in database compression technology. Oracle Database 9i introduced Basic Table Compression several years ago that compressed data that was loaded using bulk load operations. Oracle Database 11g Release 1 introduced a new feature called OLTP Table Compression that allows data to be compressed during all types of data manipulation operations, including conventional DML such as `INSERT` and `UPDATE`.

In addition, OLTP Table Compression reduces the associated compression overhead of write operations making it suitable for transactional or OLTP environments as well. OLTP Table Compression, therefore, extends the benefits of compression to all application workloads.

It should be noted that Basic Table Compression is a base feature of Oracle Database 11g Enterprise Edition (EE). OLTP Table Compression is a part of the Oracle Advanced Compression option, which requires a license in addition to the Enterprise Edition.

8.5.2 OLTP Table Compression

Oracle's OLTP Table Compression uses a unique compression algorithm specifically designed to work with OLTP applications. The algorithm works by eliminating duplicate values within a database block, even across multiple columns. Compressed blocks contain a structure called a symbol table that maintains compression metadata. When a block is compressed, duplicate values are eliminated by first adding a single copy of the duplicate value to the symbol table.

Each duplicate value is then replaced by a short reference to the appropriate entry in the symbol table. Through this innovative design, compressed data is self-contained within the database block as the metadata used to translate compressed data into its original state is stored in the block. When compared with competing compression algorithms that maintain a global database symbol table, Oracle's unique approach offers significant performance benefits by not introducing additional I/O when accessing compressed data.

The following statement indicates that all tables created in the tablespace are to use OLTP compression, unless otherwise specified:

CREATE TABLESPACE ... DEFAULT COMPRESS FOR OLTP.

8.5.3 BASIC TABLESPACE COMPRESSION

```
SQL> CREATE TABLESPACE TS1 DATAFILE '/disk2/oradata/client/ts1_01.dbf'
      SIZE 5M
      DEFAULT COMPRESS BASIC;

SQL>CREATE TABLESPACE TS2 DATAFILE '/disk2/oradata/client/ts2_01.dbf'
      SIZE 5M
      DEFAULT COMPRESS;
```

8.5 OLTP TABLESPACE COMPRESSION

```
SQL>CREATE TABLESPACE TS3 DATAFILE '/disk2/oradata/client/ts3_01.dbf'
      SIZE 5M
      DEFAULT COMPRESS FOR OLTP;
```

8.6 Online Move Datafile (12c)

Prior to Oracle 12c, moving datafiles has always been an offline task. There were certain techniques you could employ to minimize that downtime, but you couldn't remove it completely. Oracle 12c includes an enhancement which uses the command ALTER DATABASE MOVE DATAFILE in order to rename, relocate, or copy a datafile when the datafiles are online. One single step performs the required action while the databases remains entirely available in read and write for users, without any data loss.

You should however be aware of some rules:

By default, Oracle automatically deletes old data file after moving them and prevents the user from overwriting an existing file.

When you move a data file, Oracle first makes a copy of the datafile. Then, when the file is successfully copied, pointers to the datafile are updated and the old file is removed from the file system. This is why the operation requires twice the size of the files to be copied as free space.

8.6.1 Moving Online Datafile

```
SQL> alter database move datafile
      '/disk1/oradata/wstcdb/C_system01.dbf' to
      '/disk1/oradata/wstcdb/C_system02.dbf';
```

Database altered.

The available options are:

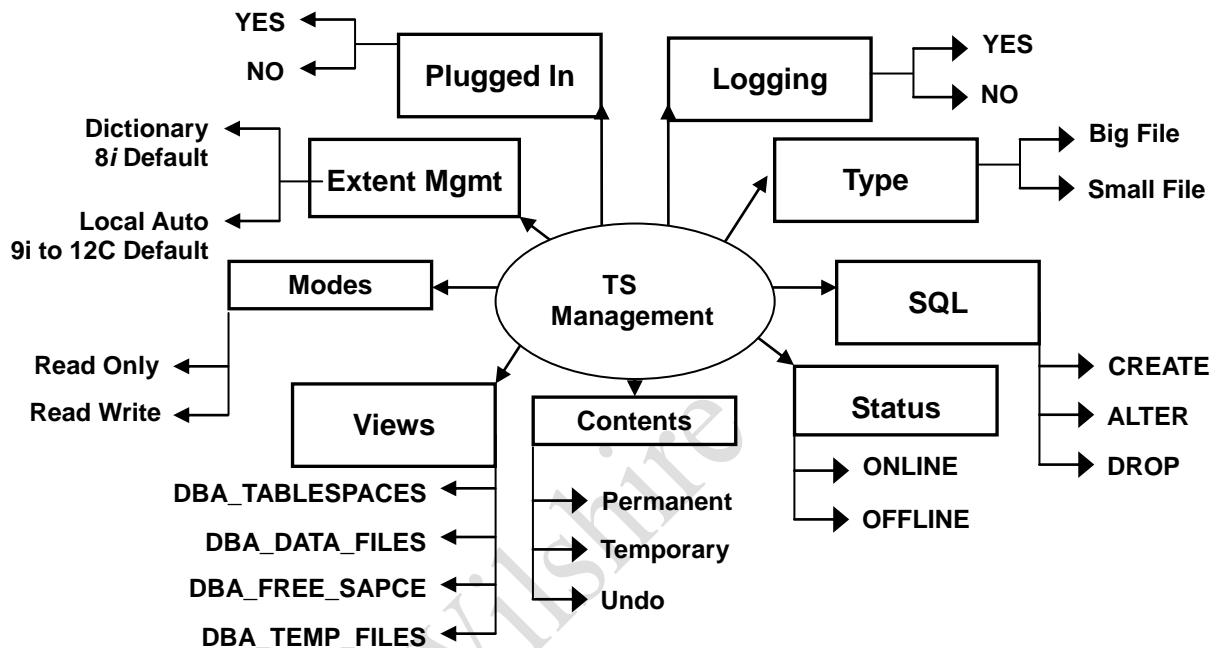
KEEP: to keep the old datafile, used to make a copy of the file. Note that the pointer will be updated to the new file, the old file only remains as unused on the filesystem. Note that on Windows, independently of the fact that the KEEP option is used or not, the old data file is not automatically deleted by Oracle. The user has to delete it manually after the copy is successfully performed.

REUSE: to overwrite an existing file.

Moving a datafile online works in both ARCHIVE and NOARCHIVE log mode. It does not work with an OFFLINE datafile.

Some files like temporary files, redo log files, and control files cannot be moved using this command.

Tablespace Management



S

9. Temporary Tablespace

A TEMPORARY tablespace is used to manage space for sort operations. Sort operations generate segments, sometimes large segments or lots of them depending on the sort required to satisfy the specification in a SELECT statement's WHERE clause. Sort operations are also generated by SELECT statements that join rows from within tables and between tables. The use of TEMPFILE instead of a DATAFILE specification for a temporary tablespace.

Used for sort operations

Cannot contain any permanent objects

Locally managed extents are recommended

```
SQL> CREATE TEMPORARY TABLESPACE temp
      TEMPFILE '/disk1/oradata/temp01.dbf' SIZE 100M
      EXTENT MANAGEMENT LOCAL UNIFORM SIZE 2M;
```

Tempfiles are also in a NOLOGGING mode. They also cannot be made read only or be renamed.

Tempfiles are required for read-only databases and are not recovered during database recovery operations.

UNIFORM SIZE parameter needs to be a multiple of the SORT_AREA_SIZE to optimize sort performance.

AUTOALLOCATE clause is not allowed for temporary tablespaces. The default extent SIZE parameter is 1M.

9.1 Default Temporary Tablespace

Each DB needs to have a specified DEFAULT TEMPORARY TABLESPACE. If one is not specified, then any user account created without specifying TEMPORARY TABLESPACE clause is assigned a temporary tablespace (TS) in SYSTEM TS. This should raise a red flag as you don't want system users to execute SELECT commands that cause sort operations to take place within the SYSTEM TS. If a default temporary TS is not specified at the time of database is created, a DBA can create one by altering the database.

```
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp;
```

After this, new system user accounts are automatically allocated to temp as their temporary TS. If you ALTER DATABASE to assign a new default temporary TS, all system users will automatically be reassigned to the new default TS for temporary operations.

Limitations:

A default temporary tablespace cannot be dropped unless a replacement is created. This would usually only be done if you were moving the tablespace from one disk drive to another.

You cannot take a default temporary tablespace offline – this is done only for system maintenance or to restrict access to a tablespace temporarily. None of these activities apply to default temporary tablespaces.

You cannot alter a default temporary tablespace to make it permanent.

To improve the concurrence of multiple sort operations, reduce their overhead, or avoid Oracle space management operations altogether, we create temporary table spaces. A temporary tablespace can be shared by multiple users and can be assigned to users with the CREATE USER statement when you create users in the database. Within a temporary tablespace, all sort operations for a given instance and tablespace share a single **sort segment**. Sort segments exist for every instance that performs sort operations within a given tablespace. The sort segment is created by the first statement that uses a temporary tablespace for sorting, after startup, and is released only at shutdown.

An extent cannot be shared by multiple transactions. You can view the allocation and deallocation of space in a temporary tablespace sort segment using the V\$SORT_SEGMENT view, and the V\$SORT_USAGE view identifies the current sort users in those segments. You cannot explicitly create objects in a temporary tablespace.

9.2 Creating Locally Managed Temporary Tablespace

Because space management is much simpler and more efficient in locally managed tablespaces, they are ideally suited for temporary tablespaces. Locally managed temporary tablespaces use **tempfiles**, which do not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data. Therefore, they can be used in standby or read-only databases. You also use different views for viewing information about tempfiles than you would for. The **V\$TEMPFILE** and **DBA_TEMP_FILES** views are analogous to the **V\$DATAFILE** and **DBA_DATA_FILES** views.

To create a locally managed temporary tablespace, you use the **CREATE TEMPORARY TABLESPACE** statement, which requires that you have the **CREATE TABLESPACE** system privilege. The following statement creates a temporary tablespace in which each extent is 100K. Each 100K extent (which is the equivalent of 50 blocks when the standard block size is 2K) is represented by a bit in the bitmap for the file.

```
SQL> CREATE TEMPORARY TABLESPACE lmtemp
      TEMPFILE '/disk1/oradata/ORCL/lmtemp01.dbf' SIZE 10M
      EXTENT MANAGEMENT LOCAL UNIFORM SIZE 200K;
```

9.3 Altering Locally Managed Temporary Tablespace

Except for adding a tempfile, you can't use **ALTER TABLESPACE** statement for a LM temporary tablespace.

```
SQL> ALTER TABLESPACE lmtemp
      ADD TEMPFILE '/disk1/oradata/ORCL/lmtemp02.dbf' SIZE 2M;
```

The following statements take offline and bring online temporary files:

```
SQL> ALTER DATABASE TEMPFILE '/disk1/oradata/ORCL/lmtemp02.dbf' OFFLINE;
```

Note: In some operating systems, Oracle doesn't allocate space for tempfile until the tempfile blocks are actually accessed. This delay in space allocation results in faster creation and resizing of tempfiles, but it requires sufficient disk space is available when the tempfiles are later used. You can't use **ALTER TABLESPACE** statement, with the **TEMPORARY** keyword, to change a locally managed (LM) permanent tablespace into a LM temporary tablespace. You must use **CREATE TEMPORARY TABLESPACE** statement to create a locally managed temporary tablespace.

```
SQL> ALTER DATABASE TEMPFILE '/disk1/oradata/ORCL/lmtemp02.dbf' ONLINE;
```

The following statement resizes a temporary file:

```
SQL> ALTER DATABASE TEMPFILE '/disk1/oradata/ORCL/lmtemp02.dbf' RESIZE
      20M;
```

The following statement drops a temporary file and deletes the operating system file:

```
SQL> ALTER DATABASE TEMPFILE '/disk1/oradata/ORCL/lmtemp02.dbf' DROP
      INCLUDING DATAFILES;
```

The TS to which tempfile belonged remains. A message is written to alert file for the datafile that was deleted. If an OS error prevents the deletion of file, the statement still succeeds, but a message describing the error is written to the alert file. It is also possible, but not shown, to AUTOEXTEND a tempfile, and to rename (RENAME FILE) a tempfile.

9.4 Storage Parameters in Locally Managed Tablespaces

You cannot specify default storage parameters for locally managed tablespaces, nor can you specify **MINIMUM_EXTENT**. If **AUTOALLOCATE** is specified, the tablespace is system managed with the smallest extent size being 64K. If **UNIFORM SIZE** is specified, then the tablespace is managed with uniform size extents of the specified **SIZE**. The default **SIZE** is 1M. When you allocate segments (create objects) in a locally managed tablespace, the storage clause specified at create time is interpreted differently than for dictionary-managed tablespaces. When an object is created in a locally managed tablespace, Oracle uses its **INITIAL**, **NEXT**, and **MINEXTENTS** parameters to calculate the initial size of the object's segment.

9.5 Creating Dictionary-Managed Temporary Tablespace

To identify a tablespace as temporary during tablespace creation, specify TEMPORARY keyword in CREATE TABLESPACE statement. You can't specify EXTENT MANAGEMENT LOCAL for temporary TS created in this fashion. To create a LM temporary tablespace, use CREATE TEMPORARY TABLESPACE statement, which is preferred method of creating a temporary tablespace. The following statement creates temporary dictionary-managed TS:

```
SQL> CREATE TABLESPACE sort DATAFILE '/disk1/oradata/ORCL/sort01.dbf'
  SIZE 10M DEFAULT STORAGE (INITIAL 100K NEXT 100K MINEXTENTS 1
  PCTINCREASE 0)
  EXTENT MANAGEMENT DICTIONARY TEMPORARY;
```

9.6 Altering Dictionary-Managed Temporary Tablespace

You can issue the ALTER TABLESPACE statement against a dictionary-managed temporary tablespace using many of the same keywords and clauses as for a permanent dictionary-managed tablespace. You can change an existing permanent dictionary-managed tablespace to a temporary tablespace, using the ALTER TABLESPACE statement.

```
SQL> ALTER TABLESPACE tbsa TEMPORARY;
```

9.7 Temporary Tablespace Groups - (From 10g)

Temporary tablespace groups are primarily used as sort work areas when there are insufficient resources for sorting a result set directly in memory, it is not uncommon for a database to run out of space in a temporary tablespace at the most unfortunate moments.

Oracle also uses temporary tablespaces to create and store an instance of each global temporary table (GTT) for each user session that invokes any PL/SQL using GTTs. Oracle 10g now permits assignment of at least one (but certainly more than one!) temporary tablespaces to a tablespace group. This tablespace group can then be assigned to a user account as if the tablespace group was a normal temporary tablespace.

And since a temporary tablespace can also be a BIGFILE tablespace, this significantly lessens the chance that any one user session or process might consume all temporary space during a huge sort - something that is especially useful during initial loading or updates of a data warehouse's tables via extraction, transformation and loading (ETL) steps.

The nice thing about temporary tablespace groups is that Oracle handles their creation (and destruction!) automatically. When a new tablespace group is first specified, Oracle implicitly creates it; likewise, when the last temporary tablespace is removed from an existing tablespace group, Oracle implicitly drops the tablespace group.

The below example shows how to create new temporary tablespace groups, add existing temporary tablespaces to existing tablespace groups, migrate a user account to an existing tablespace group, and how to remove tablespace groups implicitly from an Oracle 11g database.

9.8 Creating Temporary Tablespace Groups

Create a new temporary tablespace and assign it to a new temporary tablespace group named SORTGRP1. Oracle will create the temporary tablespace group if it doesn't exist

```
CREATE TEMPORARY TABLESPACE sortwk01
  TEMPFILE '/disk1/oradata/wilshire/sortwk01.tmp'
  SIZE 10M REUSE
  TABLESPACE GROUP sortgrp1;
Create another new temporary tablespace and assign it to an existing temporary tablespace group (SORTGRP1)
CREATE TEMPORARY TABLESPACE sortwk02
  TEMPFILE '/disk1/oradata/wilshire/sortwk02.tmp'
  SIZE 10M REUSE TABLESPACE GROUP sortgrp1;
Create another new temporary tablespace, but don't assign it to any temporary tablespace group
```

```
CREATE TEMPORARY TABLESPACE sortwk03
  TEMPFILE '/disk1/oradata/wilshire/sortwk03.tmp'
  SIZE 10M REUSE;
```

Add temporary tablespace SORTWK03 to sort group SORTGRP1

```
ALTER TABLESPACE sortwk03 TABLESPACE GROUP sortgrp1;
```

Shift a user to the SORTGRP1 temporary tablespace group

```
ALTER USER user1 TEMPORARY TABLESPACE sortgrp1;
```

Now drop all these temporary tablespaces. Once they are all removed, Oracle will implicitly delete tablespace group SORTGRP1. Oracle will not permit them to be dropped, however, until any user that's been assigned to the temporary tablespace group is migrated to another existing temporary tablespace or temporary tablespace group!

```
ALTER USER user1 TEMPORARY TABLESPACE temp;
DROP TABLESPACE sortwk03;
DROP TABLESPACE sortwk02;
DROP TABLESPACE sortwk01;
```

9.9 Temporary Undo(12c)

Each Oracle database contains a set of system related tablespaces, such as SYSTEM, SYSAUX, UNDO & TEMP, and each are used for different purposes within the Oracle database.

Pre Oracle 12C, undo records generated by the temporary tables used to be stored in undo tablespace, much similar to a general/persistent table undo records.

However, with the temporary undo feature in 12c R1, the temporary undo records can now be stored in a temporary table instead of stored in undo tablespace. The prime benefits of temporary undo includes: reduction in undo tablespace and less redo data generation as the information won't be logged in redo logs. You have the flexibility to enable the temporary undo option either at session level or database level.

Enabling temporary undo

To be able to use the new feature, the following needs to be set:

Compatibility parameter must be set to 12.0.0 or higher

Enable TEMP_UNDO_ENABLED initialization parameter

Since the temporary undo records now stored in a temp tablespace, you need to create the temporary tablespace with sufficient space

For session level, you can use:

```
SQL>ALTER SESSION SET TEMP_UNDO_ENABLE=TRUE;
```

Query temporary undo information

The dictionary views listed below are used to view/query the information/statistics about the temporary undo data:

To disable the feature, you simply need to set the following:

```
SQL> ALTER SYSTEM|SESSION SET TEMP_UNDO_ENABLED=FALSE;
```

10. User Management (Users, Privileges & Roles)

10.1 Users

In Oracle terminology, a user is someone who can connect to a database (if granted enough privileges) and optionally (again, if granted the appropriate privileges) can create objects (such as tables) in the database.

The objects a user owns are collectively called schema. A schema, on its part, is always bound to exactly one user. Because there is obviously a 1 to 1 relationship between a user and a schema, these two terms are often used interchangeable.

The CREATE USER command creates a Database user as shown here.

```
SQL> CREATE USER USER1 IDENTIFIED BY USER1;
```

- The IDENTIFIED BY clause specifies the user password.
- In order to create a user, a User must have the CREATE USER system privilege.
- Users also have a privilege domain – initially the user account has NO privileges – it is empty.
- In order for a user to connect to Oracle, you must grant the user the CREATE SESSION system privilege.
- Each username must be unique within a database. A username cannot be the same as the name of a role (roles are described in a later module).

Each user has a schema for the storage of objects within the database (see the figure below).

- Two users can name objects identically because the objects are referred to globally by using a combination of the username and object name.
- Example: USER1.Employee – each user account can have a table named Employee because each table is stored within the user's schema.

10.1.1 Database Schema

- A schema is a named collection of objects.
- A user is created, and a corresponding schema is created.
- A user can be associated only with one schema.
- Username and schema are often used interchangeably.

10.1.2 Schema Objects

- Tables
- Triggers
- Constraints
- Indexes
- Views
- Sequences
- Stored Program Units
- Synonyms
- User-defined data types
- Database Links

A complete example of the CREATE USER command.

```
SQL> CREATE USER USER1 IDENTIFIED BY new_password
      DEFAULT TABLESPACE users
      TEMPORARY TABLESPACE temp01
      QUOTA 10M ON users
      QUOTA 5M ON temp01
      QUOTA 5M ON data01
      PROFILE accountant
      ACCOUNT UNLOCK
      PASSWORD EXPIRE;
```

USER1 has two tablespaces identified, one for DEFAULT storage of objects and one for TEMPORARY objects.

USER1 has a quota set on 3 different tablespaces. USER1 has the resource limitations allocated by the PROFILE named accountant. The account is unlocked (the default – alternatively the account could be created initially with the LOCK specification).

The PASSWORD EXPIRE clause requires USER1 to change the password prior to connecting to the database. After the password is set, when the user logs on using Sqlplus or any other software product that connects to the database, the user receives the following message at logon, and is prompted to enter a new password:

```
ERROR:
ORA-28001: the account has expired
Changing password for USER1
Old password:
New password:
Retype new password:
Password changed
```

10.1.3 Alter User Command

DBA can use the ALTER USER command to change User password.

To make any other use of the command, a user must have the ALTER USER system privilege - something the DBA should not give to individual users.

Changing a user's security setting with the ALTER USER command changes future sessions, not a current session to which the user may be connected.

Example ALTER USER command:

```
SQL> ALTER USER USER1 IDENTIFIED by new_password
      DEFAULT TABLESPACE data01
      TEMPORARY TABLESPACE temp02
      QUOTA 100M ON data01
      QUOTA 0 ON inventory_tbs
      PROFILE HR;
```

10.1.4 Drop User Command

The DROP USER command is used to drop a user. This is an example:

```
SQL>DROP USER USER1;
SQL> DROP USER USER2 CASCADE;
```

- Dropping a user causes the user and the user schema to be immediately deleted from the database.
- If the user has created objects within their schema, it is necessary to use the CASCADE option in order to drop a user.
- If you fail to specify CASCADE when user objects exist, an error message is generated and the user is not dropped.

CAUTION: You need to exercise caution with the CASCADE option to ensure that you don't drop a user where views or procedures exist that depend upon tables that the user created. In those cases, dropping a user requires a lot of detailed investigation and careful deletion of objects.

If you want to deny access to the database, but do not want to drop the user and the user's objects, you should revoke the CREATE SESSION privilege for the user temporarily.

You cannot drop a user who is connected to the database - you must first terminate the user's session with the ALTER SYSTEM KILL SESSION command.

10.1.5 Related Views

DBA_TS_QUOTAS	USER_TS_QUOTAS	DBA_USERS	ALL_USERS
V\$SESSION_CONNECT_IN FO	USER_USERS	V\$SESSION	USER_PASSWORD_LIMITS

10.2 Privileges

Authentication means to authenticate a database user account ID for access to an Oracle database. Authorization means to verify that a system user account ID has been granted the right, called a privilege, to execute a particular type of SQL statement or to access objects belonging to another system user account. In order to manage system user access and use of various system objects, such as tables, indexes, and clusters. Oracle provides the capability to grant and revoke privileges to individual user accounts.

Example privileges include the right to:

- Connect to a database
- Create a table
- Select rows from another user's table
- Execute another user's stored procedure

Excessive granting of privileges can lead to situations where security is compromised. Privileges are broadly classified into two types:

- System Privileges allow a database user to perform a specific type of operation or set of operations. Typical operations are creating objects, dropping objects, and altering objects.
- Object Privileges allow a system user to perform a specific type of operation on a specific schema object. Typical objects include tables, views, procedures, functions, sequences, etc.

10.2.1 System Privileges

As Oracle has matured as a product, the number of system privileges has grown. The current number is over 100. A complete listing is available by querying the view named SYSTEM_PRIVILEGE_MAP.

- There are more than 100 distinct system privileges
- The ANY keyword in privileges signifies that users have the privileges in any schema.
- The GRANT command adds a privilege to a user or a group of users.
- The REVOKE command deletes the privileges.

The command to grant a system privilege is GRANT command. Some example GRANT commands are shown here.

```
SQL> GRANT ALTER TABLESPACE, DROP TABLESPACE TO USER1;
SQL> GRANT CREATE SESSION TO USER2 WITH ADMIN OPTION;
```

In general, you can grant a privilege to either a user or to a role. You can also grant a privilege to PUBLIC - this makes the privilege available to every system user.

10.2.2 Revoking System Privileges

The REVOKE command can be used to revoke privileges from a system user or from a role. Only privileges granted directly with a GRANT command can be revoked. There are no cascading effects when a system privilege is revoked. For example, the DBA grants the SELECT ANY TABLE WITH ADMIN OPTION to system user1, and then system user1 grants the SELECT ANY TABLE to system user2, then if system user1 has the privilege revoked, system user2 still has the privilege.

10.2.3 Object Privileges

Schema object privileges authorize the system user to perform an operation on the object, such as selecting or deleting rows in a table. A user account automatically has all object privileges for schema objects created within his/her schema. Any privilege owned by a user account can be granted to another user account or to a role.

```
SQL> GRANT SELECT, ALTER ON USER1.orders TO PUBLIC;
SQL> GRANT SELECT, DELETE ON USER1.order_details TO USER2;
```

10.2.3.1 Revoking Schema Object Privileges

Object privileges are revoked the same way that system privileges are revoked. Several example REVOKE commands are shown here. Note the use of ALL (to revoke all object privileges granted to a system user) and ON (to identify the object).

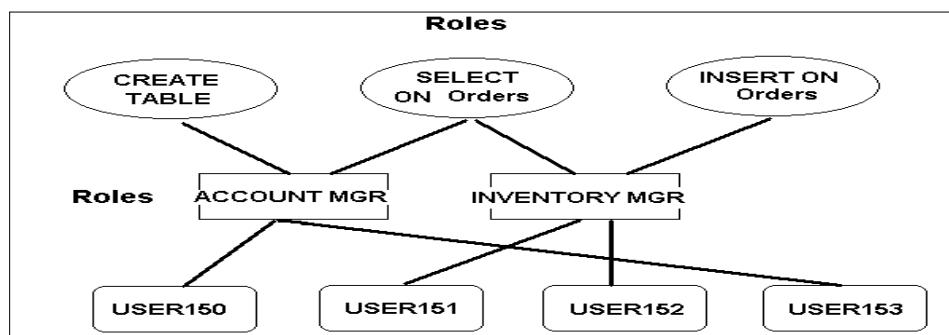
```
SQL> REVOKE SELECT ON dblock.orders FROM USER1;
SQL> REVOKE ALL ON USER1.Order_Details FROM USER2;
```

10.3 Roles

The Role database object is used to improve the management of various system objects, such as tables, indexes, and clusters by granting privileges to access these objects to roles. There are two types of privileges, System and Object. Both types of privileges can be allocated to roles. The concept of a role is a simple one - a role is created as a container for groups of privileges that are granted to system users who perform similar, typical tasks in a business.

Example: A system user fills the position of Account_Manager. This is a business role. The role is created as a database object and privileges are allocated to the role. In turn the role is allocated to all employees that work as account managers, and all account managers thereby inherit the privileges needed to perform their duties.

This figure shows privileges being allocated to roles, and the roles being allocated to two types of system users – Account_Mgr and Inventory_Mgr.



10.3.1 Facts About Roles

- You may also grant a role to another role (except to itself).
- A role can include both system and object privileges. Roles have system and object privileges granted to them just the same way that these privileges are granted to system users.
- You can require a password to enable a role.
- A role name must be unique.
- Roles are not owned by anyone - are not in anyone's schema.
- If a role has its privileges modified, then the privileges of the system users granted the role are also modified.
- There are no cascading revokes with roles.
- Using roles reduces how many Grants are stored in a database data dictionary.
- There is a limited set of privileges that can't be granted to a role, but most privileges can be granted to roles.

10.3.2 Role Benefits

- Easier privilege management: Use roles to simplify privilege management. Rather than granting the same set of privileges to several users, you can grant the privileges to a role, and then grant that role to each user.
- Dynamic privilege management: If the privileges associated with a role are modified, all the users who are granted the role acquire the modified privileges automatically and immediately.
- Selective availability of privileges: Roles can be enabled and disabled to turn privileges on and off temporarily. Enabling a role can also be used to verify that a user has been granted that role.
- Can be granted through the operating system: Operating system commands or utilities can be used to assign roles to users in the database.

10.3.3 Creating Roles

Sample commands to create roles are shown here. You must have the CREATE ROLE system privilege.

```
SQL> CREATE ROLE Account_Mgr;
SQL> CREATE ROLE Inventory_Mgr IDENTIFIED BY <password>;
```

The IDENTIFIED BY clause specifies how the user must be authorized before the role can be enabled for use by a specific user to which it has been granted. If this clause is not specified, or NOT IDENTIFIED is specified, then no authorization is required when the role is enabled.

10.3.4 Altering Roles

Use the ALTER ROLE command as is shown in these examples.

```
SQL> ALTER ROLE Account_Mgr IDENTIFIED BY <password>;
SQL> ALTER ROLE Inventory_Mgr NOT IDENTIFIED;
```

10.3.5 Granting Roles

Use the GRANT command to grant a role to a system user or to another role, as is shown in these examples.

```
SQL> GRANT Account_Mgr TO USER1;
SQL> GRANT Inventory_Mgr TO Account_Mgr, USER3;
SQL> GRANT Inventory_Mgr TO USER2 WITH ADMIN OPTION;
SQL> GRANT Access_MyBank_Acct TO PUBLIC;
```

10.3.6 Authentication

- Database Level
- OS Level
- Network Level

10.3.6.1 Database Level

Database authentication involves the use of a standard user account and password. Oracle performs the authentication.

- System users can change their password at any time.
- Passwords are stored in an encrypted format.
- Each password must be made up of single-byte characters, even if database uses a multi-byte character set.

Advantages:

- User accounts and all authentication are controlled by the database. There is no reliance on anything outside of the database.
- Oracle provides strong password management features to enhance security when using database authentication.
- It is easier to administer when there are small user communities.

Oracle recommends using password management that includes password aging/expiration, account locking, password history, and password complexity verification.

10.3.6.2 OS Level

Central authentication can be accomplished through use of Oracle Advanced Security software for directory service. Global users termed Enterprise Users are authenticated by SSL (secure socket layers) and the user accounts are managed outside of the database. Global Roles are defined in a database and known only to that database and authorization for the roles is done through the directory service.

The roles can be used to provide access privileges. Enterprise Roles can be created to provide access across multiple databases. They can consist of one or more global roles and are essentially containers for global roles.

Creating a Global User Example:

- ```
SQL> CREATE USER scott
 IDENTIFIED GLOBALLY AS 'CN=scott, OU=division1, O=oracle, C=US';

```
- Scott is authenticated by SSL and authorized by the enterprise directory service.
  - The AS clause provides a string identifier (distinguished name – DN) to the enterprise directory.
  - Disadvantage: Scott must have a user account created in every database to be accessed as well as in the directory service.

### Creating a Schema-Independent User Example:

Schema independent user accounts allow more than one enterprise user to access a shared database schema. These users are:

- Authenticated by SSL or passwords.
- Not created in the database with a CREATE USER statement.
- Privileges are managed in a directory.
- Most users don't need their own schemas – this approach separates users from databases.

```
SQL> CREATE USER inventory_schema IDENTIFIED GLOBALLY AS '';
```

In the directory create multiple enterprise users and a mapping object to tell the database how to map users DNs to the shared schema.

### 10.3.6.3 Network Level

External Authentication requires the creation of user accounts that are maintained by Oracle. Passwords are administered by an external service such as the **operating system** or a network service. This option is generally useful when a user logs on directly to the machine where the Oracle server is running.

- A database password is not used for this type of login.
- In order for the operating system to authenticate users, a DBA sets the init.ora parameter OS\_AUTHENT\_PREFIX to some set value. The default value is OPS\$ in order to provide for backward compatibility to earlier versions of Oracle.
- This prefix is used at the operating system level when the user's account username.
- We can also use a NULL string (a set of empty double quotes: "") for the prefix so that the Oracle username exactly matches the Operating System user name. This eliminates the need for any prefix.

```
#init.ora parameter
OS_AUTHENT_PREFIX=OPS$
#create user command
CREATE USER ops$scott IDENTIFIED EXTERNALLY DEFAULT TABLESPACE users
 TEMPORARY TABLESPACE temp QUOTA UNLIMITED ON users QUOTA UNLIMITED ON
temp;
```

When scott attempts to connect to the database, Oracle will check to see if there is a database user named OPS\$scott and allow or deny the user access as appropriate. Thus, to use SQLPlus to log on to the system, the UNIX user scott enters the following command from the operating system:

```
$ sqlplus /
```

All references in commands that refer to a user that is authenticated by the operating system must include the defined prefix OPS\$. Oracle allows operating-system authentication only for secure connections – this is the default. This precludes use of Oracle Net or a shared server configuration and prevents a remote user from impersonating another operating system user over a network.

The REMOTE\_OS\_AUTHENT parameter can be set to force acceptance of a client operating system user name from a nonsecure connection.

- This is NOT a good security practice.
- Setting REMOTE\_OS\_AUTHENT = FALSE creates a more secure configuration based on server-based authentication of clients.
- Changes in the parameter take effect the next time the instance starts and the database is mounted.

## 11. User Authentication

### 11.1 Overview

A basic security requirement is that we must know our users. We must identify them before we can determine their privileges and access rights, and so that we can audit their actions upon the data.

Users can be authenticated in a number of different ways before they are allowed to create a database session. In database authentication, we can define users such that the database performs both identification and authentication of users.

In external authentication, we can define users such that authentication is performed by the operating system or network service. Alternatively, we can define users such that they are authenticated by the Secure Sockets Layer (SSL).

For enterprise users, an enterprise directory can be used to authorize their access to the database through enterprise roles. Finally, we can specify users who are allowed to connect through a middle-tier server. The middle-tier server authenticates and assumes the identity of the user and is allowed to enable specific roles for the user. This is called proxy authentication.

### 11.2 Authentication by the Operating System

Some operating systems permit Oracle to use information they maintain to authenticate users. This has the following benefits:

Once authenticated by the operating system, users can connect to Oracle more conveniently, without specifying a user name or password. For example, an operating-system-authenticated user can invoke SQL\*Plus and skip the user name and password prompts by entering the following:

```
SQLPLUS /
```

With control over user authentication centralized in the operating system, Oracle need not store or manage user passwords, though it still maintains user names in the database.

Audit trails in the database and operating system can use the same user names.

When an operating system is used to authenticate database users, managing distributed database environments and database links requires special care.

### 11.3 Authentication by the Network

Authentication over a network is handled by the SSL protocol or by third-party services as described in the following subsections:

Authentication Using SSL

Authentication Using Third-Party Services

- Kerberos Authentication
- PKI-Based Authentication
- Authentication with RADIUS
- Directory-Based Services

#### 11.3.1 Authentication Using SSL

The Secure Socket Layer (SSL) protocol is an application layer protocol. It can be used for user authentication to a database, and it is independent of global user management in Oracle Internet Directory. That is, users can use SSL to authenticate to the database even without a directory server in place.

## 11.3.2 Authentication Using Third-Party Services

Authentication over a network makes use of third-party network authentication services. Prominent examples include Kerberos, Public Key Infrastructure (PKI), the Remote Authentication Dial-In User Service (RADIUS), and directory-based services, as described in the following subsections.

If network authentication services are available to us, then Oracle can accept authentication from the network service. If we use a network authentication service, then some special considerations arise for network roles and database links.

Note: To use a network authentication service with Oracle, we need Oracle Enterprise Edition with the Oracle Advanced Security option.

### 11.3.2.1 Kerberos Authentication

Kerberos is a trusted third-party authentication system that relies on shared secrets. It presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security.

It does this through a Kerberos authentication server, or through Cybersafe Active Trust, a commercial Kerberos-based authentication server.

### 11.3.2.2 PKI-Based Authentication

Authentication systems based on PKI issue digital certificates to user clients, which use them to authenticate directly to servers in the enterprise without directly involving an authentication server. Oracle provides a PKI for using public keys and certificates, consisting of the following components:

- Authentication and secure session key management using SSL.
- Oracle Call Interface (OCI) and PL/SQL functions.
- These are used to sign user-specified data using a private key and certificate. The verification of the signature on data is done by using a trusted certificate.
- Trusted certificates
- These are used to identify third-party entities that are trusted as signers of user certificates when an identity is being validated. When the user certificate is being validated, the signer is checked by using trust points or a trusted certificate chain of certificate authorities stored in the validating system. If there are several levels of trusted certificates in this chain, then a trusted certificate at a lower level is simply trusted without needing to have all its higher-level certificates recertified.
- Oracle wallets  
These are data structures that contain the private key of a user, a user certificate, and the set of trust points of a user (trusted certificate authorities).
- Oracle Wallet Manager

This is a standalone Java application used to manage and edit the security credentials in Oracle wallets. It performs the following operations:

- Protects user keys
- Manages X.509 version 3 certificates on Oracle clients and servers
- Generates a public-private key pair and creates a certificate request for submission to a certificate authority
- Installs a certificate for the entity
- Configures trusted certificates for the entity
- Creates wallets
- Opens a wallet to enable access to PKI-based services
- X.509 version 3 certificates obtained from (and signed by) a trusted entity, a certificate authority. Because the certificate authority is trusted, these certificates certify that the requesting entity's information is correct and that the public key on the certificate belongs to the identified entity. The certificate is loaded into an Oracle wallet to enable future authentication.

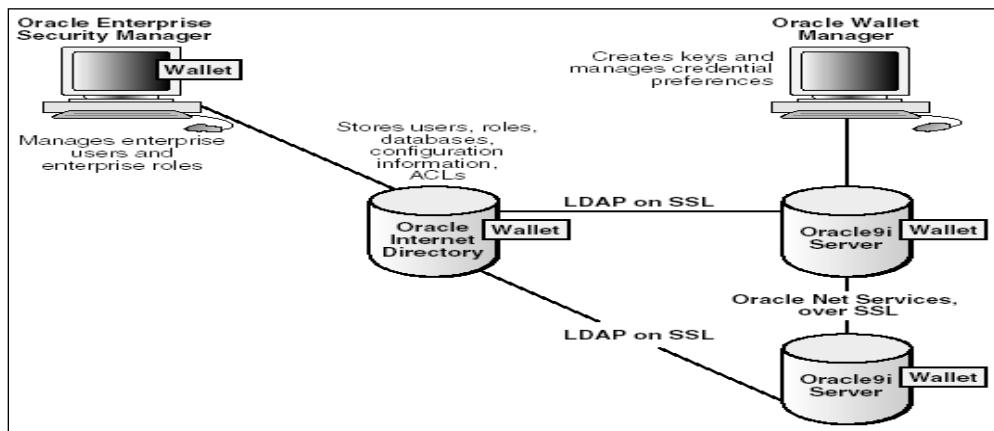


Figure 1.1 - Oracle Public Key Infrastructure

### 11.3.2.3 Authentication with RADIUS

Oracle supports remote authentication of users through the Remote Authentication Dial-In User Service (RADIUS), a standard lightweight protocol used for user authentication, authorization, and accounting.

### 11.3.2.4 Directory-Based Services

Using a central directory can make authentication and its administration extremely efficient. Directory-based services include the following:

- Oracle Internet Directory, which uses the Lightweight Directory Access Protocol (LDAP), enables information about users (called enterprise users) to be stored and managed centrally. Although database users must be created (with passwords) in each database that they need to access, enterprise user information is accessible centrally in the Oracle Internet Directory. We can also integrate this directory with Active Directory and iPlanet.
- Oracle Internet Directory lets us manage the security attributes and privileges for users, including users authenticated by X.509 certificates. Oracle Internet Directory also enforces attribute-level access control. This feature enables read, write, or update privileges on specific attributes to be restricted to specific users, such as an enterprise security administrator. Directory queries and responses can use SSL encryption for enhanced protection during authentication and other interactions.
- Oracle Enterprise Security Manager, which provides centralized privilege management to make administration easier and increase security levels. Oracle Enterprise Security Manager lets us store and retrieve roles from Oracle Internet Directory.

## 11.4 Authentication by Oracle Database

Oracle Database can authenticate users attempting to connect to a database, by using information stored in that database itself. To set up Oracle Database to use database authentication, we must create each user with an associated password. The user must provide this user name and password when attempting to establish a connection. This process prevents unauthorized use of the database, because the connection will be denied if the user provides an incorrect password. Oracle Database stores user passwords in the data dictionary in an encrypted format to prevent unauthorized alteration. Users can change their passwords at any time.

To identify the authentication protocols that are allowed by a client or a database, a DBA can explicitly set the SQLNET.ALLOWED\_LOGON\_VERSION parameter in the server sqlnet.ora file. Then each connection attempt is tested, and if the client or server does not meet the minimum version specified by its partner, authentication fails with an ORA-28040 error. The parameter can take the values 10, 9, or 8, which is the default. These values represent database server versions. Oracle recommends the value 8.

Database authentication includes the following features:

- Password Encryption While Connecting: This protection is always in force, by default.
- Account Locking
- Password Lifetime and Expiration
- Password History
- Password Complexity Verification

### **11.4.1 Password Encryption While Connecting**

Passwords are always automatically and transparently encrypted during network (client/server and server/server) connections, using AES (Advanced Encryption Standard) before sending them across the network.

### **11.4.2 Account Locking**

Oracle can lock a user's account after a specified number of consecutive failed login attempts. We can configure the account to unlock automatically after a specified time interval or to require database administrator intervention to be unlocked. Use the CREATE PROFILE statement to establish how many failed login attempts a user can attempt before the account locks, and how long it remains locked before it unlocks automatically.

The database administrator can also lock accounts manually, so that they cannot unlock automatically but must be unlocked explicitly by the database administrator.

### **11.4.3 Password Lifetime and Expiration**

The database administrator can specify a lifetime for passwords, after which they expire and must be changed before account login is again permitted. A grace period can be established, during which each attempt to login to the database account receives a warning message to change the password. If it is not changed by the end of that period, then the account is locked. No further logins to that account are allowed without assistance by the database administrator.

The database administrator can also set the password state to expired, causing the user account status to change to expired. The user or the database administrator must then change the password before the user can log in to the database.

**Note:** Failed login attempts will be limited by default in future Oracle Database releases.

### **11.4.4 Password History**

The password history option checks each newly specified password to ensure that a password is not reused for a specified amount of time or for a specified number of password changes. The database administrator can configure the rules for password reuse with CREATE PROFILE statements.

### **11.4.5 Password Complexity Verification**

Complexity verification checks that each password is complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

The sample Oracle password complexity verification routine (the PL/SQL script UTLPWDMG.SQL, which sets the default profile parameters) checks that each password meets the following requirements:

- Be a minimum of four characters in length
- Not equal the userid
- Include at least one alphabet character, one numeric character, and one punctuation mark
- Not match any word on an internal list of simple words like welcome, account, database, user, and so on
- Differ from the previous password by at least three characters

## 11.5 Single sign-on

Intranet users are commonly required to use a separate password to authenticate themselves to each server they need to access in the course of their work. Multiple passwords, however, present several problems. Users have difficulty keeping track of different passwords, tend to choose poor ones, and tend to record them in obvious places.

Administrators must keep track of a separate password database on each server and must address potential security problems arising from the fact that passwords are routinely and frequently sent over the network.

Single sign-on (SSO) does away with these problems. It enables a user to log in to different servers using a single password to obtain authenticated access to all servers she is authorized to access. It eliminates the need for multiple passwords. In addition, it simplifies management of user accounts and passwords for system administrators.

We can implement SSO in different ways, as described in the following sections:

Server-Based Single sign-on

Middle Tier Single Sign-On

### 11.5.1 Server-Based Single sign-on

A centralized directory server can be used to store user, administration, and security information. This enables the administrator to modify information in only one location, namely, the directory. This centralization lowers the cost of administration and makes the enterprise more secure.

A directory server can be used to provide centralization of user account, user role, and password information. A database server authenticates a user with the information stored in the directory. Once authenticated, a user can access the databases configured to use enterprise user security.

### 11.5.2 Middle Tier Single Sign-On

Oracle9i AS Single Sign-On server is part of the Oracle9i infrastructure and provides Web-based single sign-on and integration with legacy applications. With single sign-on, users need maintain only a single strong user name/password account for accessing all Web applications throughout the enterprise.

Applications integrated with Oracle9i AS Single Sign-On securely delegate the user authentication process. Among other things, this enforces password rules, account lockouts based on repeated login failures, and auditing. The Single Sign-On server also has the ability to authenticate a user by means of an external repository such as LDAP or a database user repository, or by local authentication.

The first time that a user tries to access an application using Single Sign-On, the server:

Authenticates the user by means of user name and password

Passes the client's identity to the Single Sign-On enabled applications

Marks the client being authenticated with an encrypted login cookie

In subsequent user logins, the login cookie provides the Single Sign-On Server with the user's identity and indicates that authentication has already been performed.

## 12. Schema Management

### 12.1 DDL Wait Option (New in 11g)

A DBA is trying to alter the table called SALES to add a column, TAX\_CODE. he issues the following SQL statement:

```
SQL> alter table sales add (tax_code varchar2 (10));
```

But instead of getting something like "Table altered", he gets:

```
Alter table sales add (tax_code varchar2 (10))
*
ERROR at line 1:
ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired
```

The error message says it all: the table is being used right now, probably by a transaction, so getting an exclusive lock on the table may be next to impossible. Of course, the rows of the table are not locked forever. When sessions perform commit the locks on those rows are released, but before that unlock period gets very far, other sessions may update some other rows of the table—and thus the slice of time to get the exclusive lock on the table vanishes.

In a typical business environment, the window for locking the table exclusively does open periodically, but the DBA may not be able to perform the alter command exactly at that time. Of course, the DBA can just keep on typing the same command over and over again until he gets an exclusive lock—or goes nuts, whichever comes first.

In Oracle Database 11g, there is a better option: the DDL Wait option.

```
SQL> alter session set ddl_lock_timeout = 10;
```

```
Session altered.
```

Now, when a DDL statement in the session does not get the exclusive lock, it will not error out. Instead, it will wait for 10 seconds. In that 10 seconds, it continually re-tries the DDL operation until it's successful or the time expires, whichever comes first. When he issues:

```
SQL> alter table sales add (tax_code varchar2 (10));
```

The statement hangs and does not error out. So, instead of DBA trying repeatedly to get the elusive fraction of time when the exclusive lock is available, she outsources repeated trials to Oracle Database 11g, somewhat like a telephone programmed to re-try a busy number.

This behavior is default so that they don't need to issue the ALTER SESSION statement every time. If you issue

```
ALTER SYSTEM SET DDL_LOCK_TIMEOUT = 10
```

The sessions automatically wait for that time period during DDL operations. Just like any other ALTER SYSTEM statement, this can be overridden by an ALTER SESSION statement.

### 12.2 Invisible Indexes (New in 11g)

An invisible index is an alternative to making an index unusable or even to drop it. An invisible index is maintained for any DML operation but is not used by the optimizer unless you explicitly specify the index with a hint. Applications often have to be modified without being able to bring the complete application offline.

Create invisible indexes temporarily for specialized non-standard operations, such as online application upgrades, without affecting the behavior of any existing application. Furthermore, invisible indexes can be used to test the removal of an index without dropping it right away, thus enabling a grace period for testing in production environments.

Oracle 11g introduces a new feature for indexes that is useful in several different situations. An invisible index is an index that is maintained by the database but ignored by the optimizer unless

explicitly specified. Using an invisible index is an alternative to dropping or making an index unusable. This feature can also prove useful when certain modules of an application require a specific index without affecting the rest of the application.

One possible use of this feature is to test the removal of an index before dropping it. Prior to 11g, this was typically achieved by making an index unusable during a set period of time. During this observation period, the DBA would monitor the database performance to make the determination of whether to drop the index. If performance was negatively affected, the index would need to be rebuilt before it could be used again.

Beginning with Oracle 11g, the DBA has the option of making the index invisible, as opposed to unusable, during this observation period. If performance degradation is observed, the index can be made visible again without rebuilding the index. This can minimize the period of performance degradation while also preventing an expensive operation to rebuild or recreate an index.

## 12.3 Virtual Columns (New in 11g)

Oracle 11g has introduced a new feature that allows you to create a "virtual column", an empty column that contains a function upon other table columns (the function itself is stored in the data dictionary). For example, assume that we have a table column named gross\_pay which is defined as a function of the (hours\_worked \* hourly\_pay\_rate) columns.

In this example, you cannot attempt to insert anything into the virtual gross\_pay column or you will get the new error:

**ORA-54013: INSERT operation disallowed on virtual columns**

Traditionally this sort of "process logic" would be stored inside the application code and computed on an as-needed basis. However, there are some benefits on placing data transformation rules inside the database itself, where it can be managed and controlled without touching the application code. However, past attempts by Oracle to place process logic inside the database have not been met with widespread acceptance. Possible benefits of virtual columns include:

- Automatic re-computation of derived columns for ad-hoc query tools
- Reduction in redundant disk space for columns that must be derived from other columns (e.g. a MONTH column that is derived from another DATE column).
- Easier interval partitioning
- Features and limitations of 11g virtual columns

Virtual computations based on other virtual columns - The virtual columns may not reference other virtual column values:

```
SQL> create table t(x number, x1 as (x+1), 2 as (x1+1));
create table t(x number, x1 as (x+1), x2 as (x1+1))
*
ERROR at line 1:
ORA-54012: virtual column is referenced in a column expression
```

Also, it's important to note that 11g virtual columns only work within the specified table, and you cannot reference columns within other tables.

## 12.4 Virtual columns (New in 11g)

However, the concept of virtual columns has the nice side effect of assisting in streamlining partitioning. For example, assume that we have a table that is partitioned by year-month (i.e. 2007-07). With 11g virtual columns, we can simply compute the partition key virtually, using a DATE column.

## 12.5 Read-Only Tables (New in 11g)

Oracle Database 11g introduces new ALTER TABLE syntax. For example:

```
ALTER TABLE <name> READ ONLY
And
ALTER TABLE <name> READ WRITE
```

The operating system sets the precedent to make a file read-only even for its owner. Earlier, a table could be made read-only (by granting only SELECT on it) to users other than the owner of the table. With this feature, the owner too can be prevented from doing unintended DML to a table.

While a table is in read-only mode only DMLs are disallowed; you can perform all DDL operations (create indexes, maintain partitions, and so on). So, a very useful application of this feature is table maintenance. You can make the table read only, perform the necessary DDL, and then make it read/write again. To see the status of the table, look for the read\_only column in the data dictionary view dba\_tables.

```
SQL> select read_only from user_tables where table_name = 'TRANS';
```

```
REA

NO
```

## 12.6 Invisible Columns (New in12c)

Invisible columns in Oracle Database 12c provide the ability to add a column to any table such that the newly added column will not be visible in a SELECT \* query and will not be considered for insertion in an INSERT statement that does not explicitly list that column. Invisible columns enable you to add a column to a table and not affect poorly coded applications that use SELECT \* or perform inserts without listing the columns.

## 12.7 In-Database Archiving (New in12c)

In-Database Archiving enables you to *archive* rows within a table by marking them as *inactive*. These *inactive* rows are in the database and can be optimized using compression, but are not visible to an application. The data in these rows is available for compliance purposes if needed by setting a session parameter.

With In-Database Archiving you can store more data for a longer period of time within a single database, without compromising application performance. Archived data can be compressed to help improve backup performance, and updates to archived data can be deferred during application upgrades to improve the performance of upgrades.

To manage In-Database Archiving for a table, you must enable ROW ARCHIVAL for the table and manipulate the ORA\_ARCHIVE\_STATE hidden column of the table. Optionally, you specify either ACTIVE or ALL for the ROW ARCHIVAL VISIBILITY session parameter.

Instead of deleting data, some applications have a concept of "mark for delete", so the data remains present in the table, but is not visible to the application. This is usually achieved by doing the following.

Add an extra column to the relevant tables that holds a flag to indicate the data is deleted.

Add an extra predicate to every statement that checks the deleted status, like "WHERE deleted = 'N'", to exclude the deleted rows from the SQL. The predicate can be hard coded into the SQL, or applied dynamically using a security policy, like in Virtual Private Database. In-Database Archiving is a feature added to Oracle Database 12c to allow this type of "mark for delete" functionality out-of-the-box, with fewer changes to the existing application code.

You can actually set ORA\_ARCHIVE\_STATE column to any string value other than '0' to archive the data, but the DBMS\_ILM package uses the following constants.

**ARCHIVE\_STATE\_ACTIVE='0'**

**ARCHIVE\_STATE\_ARCHIVED='1'**

## 12.8 Oracle Data Masking (New in12c)

Data masking (also known as data scrambling and data anonymization) is the process of replacing sensitive information copied from production databases to test non-production databases with realistic, but scrubbed, data based on masking rules. Data masking is ideal for virtually any situation when confidential or regulated data needs to be shared with non-production users. These users may include internal users such as application developers, or external business partners such as offshore testing companies, suppliers and customers. These non-production users need to access some of the original data, but do not need to see every column of every table, especially when the information is protected by government regulations.

Data masking enables organizations to generate realistic and fully functional data with similar characteristics as the original data to replace sensitive or confidential information. This contrasts with encryption or Virtual Private Database, which simply hides data and the original data, can be retrieved with the appropriate access or key. With data masking, the original sensitive data cannot be retrieved or accessed.

### 12.8.1 Data Redaction using DBMS\_REDACT

Oracle 10g gave us to ability to perform column masking to prevent sensitive data from being displayed by applications. In Oracle 12c, and back-ported to 10.2.0.4, the data redaction feature uses the `DBMS_REDACT` package to define redaction policies that give a greater level of control and protection over sensitive data.

Oracle Data Masking helps organizations comply with data privacy and protection mandates that restrict the use and sharing of private, sensitive or personally identifiable information (PII). With Oracle Data Masking, sensitive information such as credit card or social security numbers can be replaced with realistic values, allowing production data to be safely used for development, testing, or sharing with out-sourcing partners or off-shore teams for other non-production purposes.

## 12.9 Identity Columns (New in12c)

Creating new table rows often requires assigning a key value. In the past, it has been common to use an Oracle **SEQUENCE** to generate key values using an Insert trigger.

In old version of oracle database if you want to create automatic generated number you have to create sequence and use attribute nextval.

But with oracle database 12c this concept is changed new features add when you create table called generated as identity.

The new Oracle 12c now allows defining a table with the sequence. Nextval directly in the in-line column definition

```
SQL> conn test/test
SQL> create table test (test_id number generated as identity,
 test_name varchar2 (20));
```

Table created.

```
SQL> select sequence_name from user_sequences;
SEQUENCE_NAME

ISEQ$$_19934
```

## Points to Note:

A sequence is created with a system-generated name in the format **ISEQ\$\$<objectID>** where objectID is the object id of the table.

The sequence is owned by the table owner, not the SYS user.

By default the **GENERATED AS IDENTITY** clause implicitly includes the **ALWAYS** keyword i.e **GENERATED ALWAYS AS IDENTITY**. When the **ALWAYS** keyword is specified it is not possible to explicitly include values for the identity column in **INSERT OR UPDATE** statements

We can check which tables have identity columns in the **DBA\_TABLES**

When the identity clause is specified, a couple of extra bits are set in the **PROPERTY** column of the **TAB\$** table

### 12.10.1 Restrictions

There are a few restrictions on the use of identity columns.

A table can only contain one identity column

The column must be a numeric data type

If an identity clause is specified for a column, then a default clause cannot be specified for that column

**NOT NULL** and **NOT DEFERRABLE** constraints are created automatically for the identity column if they do not already exist. Conflicting constraints will raise an error.

**CREATE TABLE AS SELECT (CTAS)** statements do not inherit the identity property of a column in the source table.

## 13. Security Enhancements

There are several new security-related features in Oracle Database 11g, but for the purpose of the certification test, you must focus on the following new security related enhancements.

- Secure password support
- Configuring fine-grained access to network services
- Encrypting a tablespace

### 13.1 Secure Password Support

Oracle Database 11g provides several new ways to make database passwords more secure. Among these are the following new password-related features:

- Case-sensitive passwords make databases more secure. I discuss this feature in the following sections.
- You can include multibyte characters in a password without enclosing them in quotation marks.
- All passwords entered by users are passed through the string hash algorithm (SHA-1, which uses a 160-bit key) and compared with the stored credential for that user.
- Passwords always use salt, which is a random unique value added to the passwords to ensure a unique output credential.

#### 13.1.1 Configuring Case-Sensitive Passwords

In Oracle Database 11g, for the first time, database passwords are case-sensitive by default. That is, when you create or modify a user account, the passwords are automatically case sensitive. You can control case sensitivity in the database by setting the new initialization parameter `sec_case_sensitive_logon`. Because, by default, the database now enforces password case sensitivity, the default value of this parameter is set to TRUE.

Although Oracle recommends that you adhere to the new default of case sensitive passwords, there may be times when you have to disable case sensitivity in order to be compatible with some applications that, say, use hard-coded, case insensitive passwords. In such a case, you may reinstate the old-fashioned case insensitivity if you want, by changing the value for this parameter to FALSE.

```
$ alter system set sec_case_sensitive_logon = false scope=pfile;
```

#### 13.1.2 Case Sensitivity and Upgrading

When you upgrade from Oracle Database 11g or an older release of the database to Oracle Database 11g, the passwords remain case insensitive. You must change the passwords for the users in order to make the passwords case sensitive. Use the following query on the `DBA_USERS` view to find out which of your users have case-sensitive passwords, as shown here:

```
SQL> select username, password, password_versions from dba_users;
```

| User-Name   | PASSWORD | PASSWORD_VER |
|-------------|----------|--------------|
| DMTS        |          | 10G 11G 12C  |
| PP          |          | 10G 11G 12C  |
| ORACLE_OCM  |          | 10G 11G 12C  |
| RAMESH      |          | 10G 11G 12C  |
| XS\$NULL    |          | 11G          |
| RAM         |          | 10G 11G 12C  |
| OPS\$RAMESH |          | 10G 11G 12C  |
| P1          |          | 10G 11G 12C  |
| GSMCATUSER  |          | 10G 11G 12C  |
| P2          |          | 10G 11G 12C  |
| DIP         |          | 10G 11G 12C  |
| GSMUSER     |          | 10G 11G 12C  |
| AUDSYS      |          | 10G 11G 12C  |

In the preceding query, the new Oracle Database 12c column `PASSWORD VERSIONS` shows the database release in which that password was originally created or changed. In this case, it shows that all passwords were either created in Oracle Database 12c (or earlier releases) and changed in Oracle Database 12c, or were created in Oracle Database 12c. When you upgrade from the Oracle Database 12c release to the Oracle Database 12c release, all passwords remain case insensitive. You must make the passwords case sensitive by using the `alter user`

`<username> identified by <new_password>` command. If you create a new Oracle Database 12c database, on the other hand, the user accounts you create will have case-sensitive passwords by default.

Note that unlike in the previous releases, the `PASSWORD` column is blank. In the older releases, Oracle showed you the encrypted passwords. In Oracle Database 12c, you can't see the encrypted passwords by querying the `DBA_USERS` view. The encrypted passwords, of course, are still stored—in the `USER$` view. In Oracle Database 12c, user passwords are stored as a user credential after first passing them through a hash algorithm.

Whenever you log in, the database hashes the password you enter and compares it with the stored credential. In Oracle Database 12c, when a user tries to connect with a wrong password, the database will delay subsequent login attempts after the third failed attempt. The database will gradually increase the delay between consecutive attempts, up to a maximum of about ten seconds.

### 13.1.3 Case Sensitivity and Password Files

You are familiar with password files, which you use to specify passwords for users with the `SYSDBA` and `SYSOPER` privileges. In Oracle Database 11g, there is a new optional parameter you may specify when creating a new password file.

The parameter, named `ignore case`, determines whether the passwords in the password file are case sensitive or not. By default, the value of the `ignore case` parameter is set to `no (n)`, meaning that all passwords inside a password file will be automatically case sensitive. Here's an example that shows how you specify the `ignore case` parameter:

```
$ orapwd file=orapw$ORACLE_SID entries=30 ignorecase=y
Enter password for SYS:
```

In the preceding example, the value of the `ignorecase` parameter is set to `y`, meaning the database will ignore the case in which you enter the password when logging into the database. When you import users from an older database release, the passwords of any users with the `SYSDBA` or `SYSOPER` privilege will be imported into your current password file.

These passwords will be case insensitive by default and Oracle recommends that you have the users change their passwords. If you enable case sensitivity (setting the `sec_case_sensitive_logon` parameter to `TRUE`), when these users change their passwords they automatically become case sensitive.

By the by, in addition to the new `ignorecase` parameter, the `orapwd` command has other modifications in this release, as shown here:

```
$ orapwd
Usage: orapwd file=<fname> password=<password> entries=<users>
force=<y/n> ignorecase=<y/n> nosysdba=<y/n>
where
 file - name of password file (required),
 password - password for SYS (optional),
 entries - maximum number of distinct DBA (required),
 force - whether to overwrite existing file (optional),
 ignorecase - passwords are case-insensitive (optional),
 nosysdba - whether to shut out the SYSDBA logon (optional Database Vault
 only).
There must be no spaces around the equal-to (=) character.
```

Oracle Database 11g comes with a new version of the Oracle PL/SQL script utlpwdmg.sql, which provides you a simple password verification function. You can customize this function.

Note the following differences in the usage of the orapwd command:

- The password parameter is optional now, whereas it was required before.
- The ignorecase parameter is new, as explained earlier.
- The nosysdba parameter is also new, but is relevant only if you've installed Oracle Database Vault.

### 13.1.4 New Password Management Function

Oracle provides a script named utlpwdmg.sql (stored in the \$ORACLE\_HOME/ dbms/admin directory) to let you implement several password management features such as the setting of the default password resource limits. The script contains code for creating a password verification function named verify\_function\_11g, for checking password complexity.

The function checks only for minimal password complexity and you can customize it to satisfy more complex password checks. Oracle offers both the old verify\_function creation code and the code to create an updated Oracle Database 11g version of the function (verify\_function\_11g). The new version of the function includes the following additional password protection features:

- Ensures that the password is at least eight characters long. In the previous release, the minimum length of the password was only four characters.
- Checks if the password is the same as the username reversed.
- Checks if the password is the same or similar to the server name.

The following alter profile statement in the utlpwdmg.sql script will first create the new 11g version of the verify\_function and then alter the DEFAULT profile.

```
Alter profile default limit
 password_life_time 180
 password_grace_time 7
 password_reuse_time_unlimited
 password_reuse_max_unlimited
 failed_login_attempts 10
 password_lock_time 1
 password_verify_function verify_function_11g;
```

As you are aware from earlier releases, the database assigns the DEFAULT profile to all new users in the database who haven't been assigned a specific profile. It's the default profile inherited by all users in the database. Note the last part of the SQL statement (password\_verify\_function verify\_function\_11g).

This means that if you create the password verify function in your database as recommended by Oracle, any time a user (including the DBA) attempts to create a new password or to change an existing password, the database will execute the verify\_function\_11g function to ensure that the new password meets all the requirements specified by that function.

### 13.1.5 New Security-Related Initialization Parameters

You've learned about the new parameter sec\_case\_sensitive\_logon, which allows you to control the case sensitivity of user passwords, thus reducing your vulnerability to brute force attacks. In addition, there are also these new parameters that affect security:

- sec\_protocol\_error\_further\_action Specifies what action the database must take when it receives bad packets from a client, the presumption being that the client is acting with a malicious intent. The possible actions you can specify are: continue, drop the connection, or delay the acceptance of requests from the client.
- sec\_protocol\_error\_trace\_action Specifies a monitoring action such as none, trace, log, or alert.
- sec\_max\_failed\_login\_attempts Drops a connection after a specified number of failed login attempts. This policy remains enabled even if you don't enable a password profile.
- ldap\_directory\_sysauth Specifies whether the database uses strong authentication for database administrators. You must set the value of this parameter to yes if you want to implement strong authentication such as Kerberos tickets or certificates over a Secure Socket Layer (SSL). You disable strong authentication when you specify the value no for this parameter.

## 13.2 Encrypting Tablespaces

Oracle has been gradually improving its encryption capabilities over the years. In Oracle 8i, Oracle introduced the DBMS\_OBFUSCATION\_TOOLKIT, and the Oracle 8.1 release introduced the DBMS\_CRYPTO package to facilitate encryption. Both the toolkit and the DBMS\_CRYPTO package required that the application manage the encryption keys and call the APIs to perform necessary encryption/decryption operations.

In Oracle Database 11g, Oracle introduced the new Transparent Data Encryption (TDE) feature, which let you easily encrypt a column's data in a table. The encryption is called transparent because the Oracle database takes care of all the encryption and decryption details, with no need for you to manage any tables or triggers to decrypt data.

Now, in Oracle Database 11g, you can encrypt an entire tablespace by simply using a pair of special clauses during tablespace creation. The tablespace creation statement for an encrypted tablespace has the following

```
syntax:
 create tablespace <tbsp_name>
 encryption
 default storage (encrypt)
```

The encryption clause in line 2 doesn't actually encrypt the tablespace. You provide the encryption properties by setting values for the keyword encryption. You may additionally specify the using clause along with the encryption clause (encryption using . . .) to specify the name of the encryption algorithm you want to use, such as 3DES168, AES128, AES192, and AES250.

If you want to use the default algorithm of AES128, you can omit the using clause altogether. It is the encrypt keyword passed to the storage clause in line 3 that encrypts the tablespace. In the following sections, let's review how to encrypt a tablespace. But before I actually encrypt a tablespace, let me show you how to create an Oracle wallet, because you'll need the wallet when you encrypt a tablespace.

### 13.2.1 Creating the Oracle Wallet

An Oracle Wallet is a container to store authentication and signing credentials. The tablespace encryption feature uses the wallet to protect the master key used in the encryption. There are two kinds of Oracle wallets—encryption wallets and auto-open wallets.

You must manually open an encryption wallet after database startup, whereas the auto-open wallet automatically opens upon database startup. The encryption wallet is commonly recommended for tablespace encryption, unless you're dealing with unattended Data Guard environments, in which case the automatic opening of the wallet comes in handy.

In order to use Oracle Wallet, you must create the wallet itself and then add a master key to it. You can create a wallet in a couple of ways. You can create the Oracle Wallet by:

- Using the mkstore command from the operating system command line
- Invoking the Oracle Wallet Manager either through a GUI interface or by issuing the command owm at the command line
- Executing the alter system statement from SQL\*Plus

Here is the syntax to create a wallet from the OS:

```
$ mkstore -wrl $ORACLE_BASE/admin/$ORACLE_SID/wallet -create
Enter password:
Enter password again:
```

However, the simplest way to create the wallet is to simply use the following command in SQL\*Plus:

```
SQL> alter system set encryption key identified by "password"
```

This command both creates the wallet if it doesn't already exist and adds a master key to it. Oracle stores the encryption keys outside the database, in a file called an Oracle Wallet. By default, this file is named ewallet.p12 under both Windows and UNIX/Linux-based systems. The location where Oracle stores this file is operating system-specific. However,

you can specify a different location by using the parameter `encryption_wallet_location` in the `sqlnet.ora` file.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE=
 (METHOD=file)
 (METHOD_DATA=
 (DIRECTORY=/apps/oracle/general/wallet)))
```

You must have the `alter system` privilege as well as a password for an Oracle Wallet. If you don't have an Oracle Wallet, you must create one. You can create a new Oracle Wallet using the Oracle Wallet Manager (OWM) or by using special SQL statements. In the following example, we show you how to create and open an Oracle Wallet using a SQL statement.

Before you create the Oracle Wallet, you must first create a directory named `wallet` under the directory `$ORACLE_BASE/admin/$ORACLE_SID`. If you don't do this, you'll get the error ORA-28368: cannot auto-create wallet. After you create the directory named `wallet`, issue the following statement from SQL\*Plus:

```
SQL> alter system set encryption key identified by "sammyy11";
System altered.
```

The `alter system` statement you issued in the previous example works in the following way:

- If you already have an Oracle Wallet, it opens that wallet and creates (or re-creates) the master encryption key.
- If you don't have an Oracle Wallet already, it creates a new wallet, opens the wallet, and creates a new master encryption key.

Now that you've successfully created the Oracle Wallet and ensured it is open, you're ready to encrypt tablespaces using the new tablespace encryption feature.

### 13.2.2 Creating an Encrypted Tablespace

Once you create the Oracle Wallet, creating an encrypted tablespace is a breeze. The following is an example showing how to create a simple encrypted tablespace that uses the default DES128 encryption. Because you don't have to specify the default encryption level, you don't specify the `using` clause for the `encryption` clause in line 3.

```
SQL> create tablespace encrypt1
 2 datafile 'c:\orcl11\app\oracle\oradata\eleven\
 3 encrypt_01.dbf' size 100m
 4 encryption
 5* default storage (encrypt);
Tablespace created.
```

The `storage` parameter `encrypt` ensures that the tablespace is encrypted. The `encryption` clause determines the tablespace encryption properties. In this example, I use the `encryption` clause by itself, without specifying a particular encryption algorithm for the tablespace.

The database will use the default AES128 encryption algorithm to encrypt the tablespace. You can also specify the optional `using <algorithm>` clause along with the `encryption` clause, as shown in the following example, to specify the exact encryption algorithm you want.

```
SQL> create tablespace encrypt1
 2 datafile 'c:\orcl11\app\oracle\oradata\eleven\
 3 encrypt_01.dbf' size 100m
 4 encryption using '3des168'
 5* default storage (encrypt);
Tablespace created.
```

The previous example shows how to specify a particular encryption algorithm, 3DES168, instead of the default AES128 algorithm. The new column `ENCRYPTED` in the `DBA_TABLESPACES` view lets you check the encryption status of a tablespace:

```
SQL> select tablespace_name, encrypted from dba_tablespaces;
TABLESPACE_NAME ENC

SYSTEM NO
SYSAUX NO
UNDOTBS NO
TEMP NO
USERDATA NO
```

Oracle encrypts the data in the tablespace upon writing it and decrypts it upon reading the data. There is no additional memory requirement because the tablespace encryption and decryption aren't performed in memory, but there is an encryption overhead on I/O.

The encrypted data will remain encrypted in both the undo segments as well as the redo logs, in addition to being encrypted in temporary tablespaces during typical operations such as sort and join operations that make use of a temporary tablespace.

If you want to change the key for an encrypted tablespace, the only method in the present release is to create a new tablespace and move all the objects in the encrypted tablespace to the new tablespace. You can then encrypt the new tablespace.

### 13.2.3 Restrictions on Tablespace Encryption

When you encrypt a column(s) for a table, there are limitations on certain queries. By encrypting the entire tablespace, some of these restrictions are removed.

For example, in Oracle Database 11g, if the column is part of a foreign key or used in another Database Constraint, it cannot be encrypted. By encrypting the entire tablespace instead of just a table or tables, this restriction is lifted. Note the following restrictions on tablespace encryption. You

- Can transport an encrypted tablespace only if the two operating system platforms have the same endianness and the same wallet.
- Can't change the key for an encrypted tablespace.
- Can't encrypt temporary and undo tablespaces.
- Can't encrypt bfiles and external tables.

## 14. Storage Hierarchy in Oracle Database

A database is made up of one or more tablespaces. A **tablespace** is a logical storage container in Oracle that comes at the top of the storage hierarchy and is made up of one or more data files. These files might be cooked files in a file system, raw partitions, ASM-managed database files, or files on a clustered file system.

### 14.1 Blocks

A **block** is the smallest unit of space allocation in Oracle. Blocks are where your rows of data, or index entries, or temporary sort results will be stored. A block is what Oracle generally reads from and writes to disk. Blocks in Oracle are generally one of four common sizes: 2KB, 4KB, 8KB, or 16KB (although 32KB is also permissible in some cases; there are restrictions in place as to the maximum size by operating system).

The Operating System Block size is the minimum unit of operation (read/write) by the OS and is a property of the OS file system. While creating an Oracle database we have to choose the “**Data Base Block Size**” as a multiple of the Operating System Block size. The minimum unit of operation (read/write) by the Oracle database would be this “Oracle block”, and not the OS block.

**Note:** The default block size for a database does not have to be a power of two. Powers of two are just a convention commonly used. You can, in fact, create a database with a 5KB, 7KB, or  $n$ KB block size, where  $n$  is between 2KB and 32KB. I do not advise making use of this fact in real time; though stick with 2KB, 4KB, 8KB, or 16KB as your block size. Once set, the “Data Base Block Size” cannot be changed during the life of the database. The relationship between segments, extents, and blocks is shown in Figure 1-1.

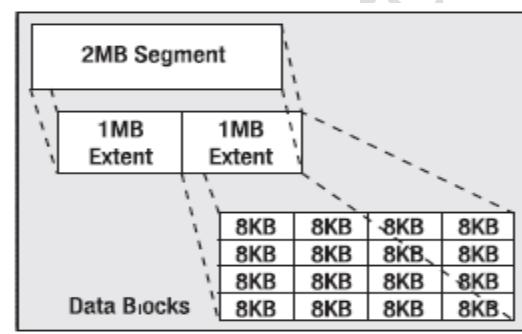


Figure 1-1. Segments, extents, and blocks

A segment is made up of one or more extents, and an extent is a contiguous allocation of blocks. Starting with Oracle9i Release 1, a database may have up to six different block sizes in it.

**Note:** This feature of multiple block sizes was introduced for the purpose of making transportable tablespaces usable in more cases. The ability to transport a tablespace allows a DBA to move or copy the already formatted data files from one database and attach them to another. For example, to immediately copy all of the tables and indexes from an Online Transaction Processing (OLTP) database to a Data Warehouse (DW).

However, in many cases, the OLTP database might be using a small block size, such as 2KB or 4KB, whereas the DW would be using a much larger one (8KB or 16KB). Without support for multiple block sizes in a single database, you would not be able to transport this information. Tablespaces with multiple block sizes should be used to facilitate transporting tablespaces and are not generally used for anything else.

There will be the database default block size, which is the size that was specified in the initialization file during the CREATE DATABASE command. The SYSTEM tablespace will have this default block size always, but you can then create other tablespaces with nondefault block sizes of 2KB, 4KB, 8KB, 16KB and, depending on the operating system, 32KB. The total number of block sizes is six if and only if you specified a nonstandard block size (not a power of two) during database creation. Hence, for all practical purposes, a database will have at most five block sizes: the default size and then four other nondefault sizes.

Any given tablespace will have a **consistent block size**, meaning that every block in that tablespace will be the same size. A multi-segment object, such as a table with a LOB column, may have each segment in a tablespace with a different block size, but any given segment (which is contained in a tablespace) will consist of blocks of exactly the same size. All blocks, regardless of their size, have the same general format, which looks something like Figure 1-2.

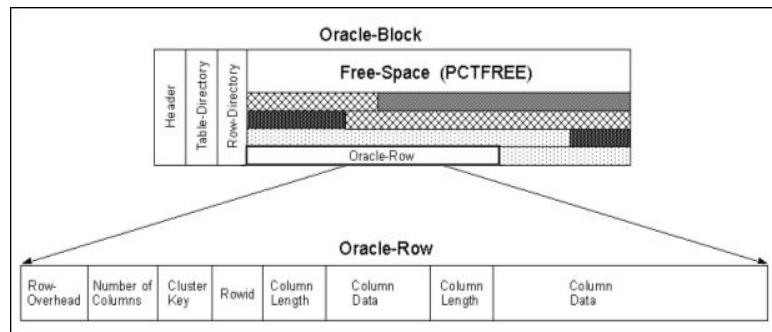


Figure 1-2. The structure of a block

### Header

Header contains the general information about the data i.e. block address, and type of segments (table, index etc). It also contains the information about table and the actual row (address) which holds the data. The next two block components, table directory and row directory, and are found on the most common types of database blocks, those of HEAP organized tables, but suffice it to say that most tables are of this type.

### Table-Directory

The **table directory**, if present, contains information about the tables that store rows in this block (data from more than one table may be stored on the same block).

### Row-Directory

The **row directory** contains information describing the rows that are to be found on the block. This is an array of pointers to where the rows are to be found in the data portion of the block. These three pieces of the block are collectively known as the **block overhead**, which is space used on the block that is not available for your data, but rather is used by Oracle to manage the block itself. The remaining two pieces of the block are straightforward: there will possibly be **free** space on a block, and then there will generally be **used** space that is currently storing data.

### Free Space

Space allocated for future update/insert operations. Generally affected by values of **PCTFREE** and **PCTUSED** parameters.

### Data

Actual row data.

### FREELIST, PCTFREE and PCTUSED

While creating / altering any table/index, Oracle used two storage parameters for space control.

- **PCTFREE** - The percentage of space reserved for future update of existing data.
- **PCTUSED** - The percentage of minimum space used for insertion of new row data.  
This value determines when the block gets back into the FREELISTS structure.
- **FREELIST** - Structure where Oracle maintains a list of all free available blocks.

Oracle will first search for a free block in the FREELIST and then the data is inserted into that block. The availability of the block in the FREELIST is decided by the PCTFREE value. Initially an empty block will be listed in the FREELIST structure, and it will continue to remain there until the free space reaches the PCTFREE value. When the free space reaches the PCTFREE value the block is removed from the FREELIST, and it is re-listed in the FREELIST table when the volume of data in the block comes below the PCTUSED value. Oracle uses FREELIST to increase the performance. So for every insert operation, Oracle needs to search for the free blocks only from the FREELIST structure instead of searching all blocks.

## 14.2 Extents

An **extent** is a logically contiguous allocation of space in a file (files themselves, in general, are not contiguous on disk; otherwise, we would never need a disk defragmentation tool). Also, with disk technologies such as Redundant Array of Independent Disks (RAID), we might find a single file is not only not contiguous on a single disk, but also spans many physical disks. Each extent in turn is made up of OBs which are contiguous meaning that if we have lot of free space in the Tablespace's datafile, still if it is not contiguous enough to fit our extent, the operation fails.

At the time of Segment creation, we define how many extents and also what should be the size of each extent. For e.g. if we have only 1 extent for our Table-Segment "EMP", which is only 20KB, may be it gets filled up after inserting some records. Then Oracle automatically allocates another extent for this Segment which is known as "Dynamic Space Allocation" Oracle strongly doesn't recommend this "DSA", as more and more extents we have for this Segment, we'll be spending longer times for selecting data from this Table-Segment, since it has so many extents which may not be contiguous and even spread across the.

This is the reason for time we spend on big-reports. To answer this problem, DBA should have clear understanding of at least High-Activity Segments, so that he can plan the Extents for that Segment properly. For an object to grow beyond its initial extent, it will request another extent be allocated to it. This second extent will not necessarily be located right next to the first extent on disk it may very well not even be allocated in same file as the first extent. The second extent may be located very far away from the first extent, but the space within an extent is always logically contiguous in a file. Extents vary in size from one Oracle data block to 2GB.

## 14.3 Segments

Segments are the major organizational structure within a tablespace. **Segments** are simply our database objects that consume storage - objects such as tables, indexes, rollback segments, and so on. When we create a table, we create a table segment. When we create a partitioned table, we create a segment per partition. When we create an index, we create an index segment, and so on. Every object that consumes storage is ultimately stored in a single segment. There are rollback segments, temporary segments, cluster segments, index segments, and so on.

**Note:** It might be confusing to read "Every object that consumes storage is ultimately stored in a single segment." We will find many CREATE statements that create multi segment objects. The confusion lies in the fact that a single CREATE statement may ultimately create objects that consist of zero, one, or **more** segments.

For example, CREATE TABLE T (x int primary key, y clob) will create four segments: one for TABLE T, one for index that will be created in support of primary key, and two for CLOB (one segment for CLOB is LOB index and other segment is LOB data itself). On the other hand, CREATE TABLE T (x int, y date) cluster MY\_CLUSTER, will create no segments. Oracle objects which consume (demand) space (bytes) are known as segments. These segments are:

- Table Segments
- Index Segments
- Rollback Segments
- Undo Segments
- Temporary Segments

Segments are created in Tablespace which are made up of Extents. Each segment should at least have 1 extent. There is no upper limitation for extents if we define as "unlimited-maxextents". These extents are created in that particular TS's "datafile" (where ever we have created Segment). As we already understood, one Segment can have many extents and these extents need not be next-to-next (contiguous). But each Extent is made up of contiguous OBs.

Having more than 20 extents for a Segment performance comes down, since our Report may need to use all these extents to collect data. So DBA's primary responsibility is making sure no Segments are consuming more than 20 extents. So at creation of Segment we need to provide above "Storage Parameters" detail to Oracle and depending on space availability, Oracle may either create this Segment or may fail because insufficient space.

If we don't define these Storage Parameters, Oracle will enforce these values from Tablespace definition. So at the time of Tablespace, DBA suppose to give the "Default Storage Parameters" (DSP this is for the future segments to be created in this Tablespace), which will be applied by Oracle on those Segment, which come without any values.

If DBA doesn't apply any DSP at the Tablespace level, then Oracle automatically offers its own DSP at the Tablespace Level, which in turn gets applied to Segment. For E.g. Oracle's DSP are: initial->10k next->10k pctincrease->50 minextents->2 maxextents->depends on the Oracle Block Size. If we are not cautious about these things, which are "core" things in Oracle, certainly we'll end up having performance problems as we go on inserting data in these Segments.

## 14.4 Tablespaces

A tablespace is a container which holds segments. Each and every segment belongs to exactly one tablespace. A tablespace may have many segments within it. All extents of a given segment will be found in tablespace associated with that segment.

Segments never cross tablespace boundaries. A tablespace itself has one or more data files associated with it. An extent for any given segment in a tablespace will contain entirely within one data file. However, a segment may have extents from many different data files. Graphically, a tablespace might look like Figure 4-3.

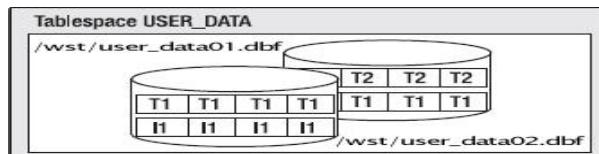


Figure 4-3. A tablespace containing two data files, three segments, and four extents

Figure 4-3 shows a tablespace named USER\_DATA. It consists of two data files, user\_data01 and user\_data02. It has three segments allocated it: T1, T2, and I1 (probably two tables and an index). The tablespace has four extents allocated in it, and each extent is depicted as a logically contiguous set of database blocks.

Segment T1 consists of two extents, one extent in each file. Segments T2 and I1 each have one extent depicted. If we need more space in this tablespace, we could either resize the data files already allocated to the tablespace or we could add a third data file to it. Tablespaces are a logical storage container in Oracle.

As developers, we will create segments in tablespaces. We will never get down to the raw "file level" - we do not specify that we want our extents to be allocated in a specific file (we can, but we do not in general). Rather, we create objects in tablespaces, and Oracle takes care of the rest.

If at some point in the future, the DBA decides to move our data files around on disk to more evenly distribute I/O, which is OK with us. It will not affect our processing at all. Since one Tablespace can be made up of different "", one Segment can have extents in multiple, which belong to the same Tablespace.

## 14.5 Storage Hierarchy Summary

In summary, the hierarchy of storage in Oracle is as follows:

1. A block is the smallest unit of allocation in the database. A block is the smallest unit of I/O used by a DB.
2. An extent is a logically contiguous set of blocks on disk. An extent is in a single tablespace and, furthermore, is always in a single file within that tablespace.
3. A segment (TABLE, INDEX, and so on) is made up of one or more extents. A segment exists in a tablespace, but may have data in many data files within that tablespace.
4. A tablespace is made up of one or more data files. These files might be cooked files in a file system, raw partitions, ASM managed database files, or a file on a clustered file system. A tablespace contains segments.
5. A database is made up of one or more tablespaces

## 14.6 Storage Parameters

| Storage Parameters | Description                                                                                                                                                                             |                        |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>INITIAL</b>     | The size of the very 1st extent for this segment                                                                                                                                        |                        |
| <b>NEXT</b>        | In case if all our previous extent got filled up, this defines the size of the next extent to be allocated by Oracle                                                                    |                        |
| <b>PCTINCREASE</b> | We can demand increased Ext size compared to previous Extent. This would minimize no. of Extents that Segment needs since as it grows the size of the Extent is getting bigger & bigger |                        |
| <b>MINEXTENTS</b>  | Min no. of Extents for the Segment at the time of Segment Creation                                                                                                                      |                        |
| <b>MAXEXTENTS</b>  | Max. no. of extents we can have for this segment. Oracle has certain default values, which depend on the OBS. E.g.,                                                                     |                        |
|                    | OBS                                                                                                                                                                                     | MAXEXTENTS             |
|                    | 2k                                                                                                                                                                                      | 121                    |
|                    | 4k                                                                                                                                                                                      | 249                    |
|                    | 8k                                                                                                                                                                                      | 508 approx. and so on. |

## 15. UNDO Management (From 9i)

### 15.1 Overview

Every Oracle database must have a method of maintaining information that is used to rollback, or undo the changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Oracle refers to these records collectively as UNDO.

Historically, Oracle has used rollback segment to store undo (past image). Space management for these rollback segments has proven to be quite complex. Oracle from 9i uses a new method called UNDO TABLESPACE.

This eliminates the complexities of managing rollback segment space and enables DBAs to exert control over how long undo is retained before being overwritten. But of course, you cannot use both the methods in the same database.

The undo space management mode for a database is set by specifying the initialization parameter UNDO\_MANAGEMENT = AUTO. An undo tablespace must be available into which, Oracle will store undo records. The default undo tablespace is created at database creation, or an undo tablespace can be explicitly created as

```
SQL> create undo tablespace undots
 datafile '/disk1/oradata/ORCL/undots01.dbf' SIZE 10M
 REUSE AUTOEXTEND ON;
```

You can switch from one undo tablespace to another since the UNDO\_TABLESPACE parameter is a dynamic parameter, which can be changed using 'ALTER SYSTEM'.

The following statement effectively switches to a new undo tablespace:

```
SQL> ALTER SYSTEM SET UNDO_TABLESPACE = UNDOTS
```

The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a **PENDING OFFLINE** mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

You can also specify the amount of undo information to retain in the undo tablespace to maintain read consistency.

Required use of UNDO clause with the CREATE command

- Used to store undo segments.
- Cannot contain any other object.
- Extents are locally managed.
- Can only use DATAFILE and EXTENT MANAGEMENT clauses.
- More than one UNDO tablespace can exist, but only one can be active at a time.

### 15.2 Undo Retention

If Undo Segment data is to be retained a long time, then the Undo tablespace will need larger datafiles.

- The retention period is set with the UNDO\_RETENTION parameter that defines the period in seconds.
- You can set this parameter in the initialization file or you can dynamically alter it with the ALTER SYSTEM command:  

```
SQL> ALTER SYSTEM SET UNDO_RETENTION = 43200;
```
- The above command will retain Undo Segment data for 720 minutes (12 hours) – the default value is 900 seconds (15 minutes).
- If the tablespace is too small to store Undo Segment data for 720 minutes, then the data is not retained – instead space is recovered by the Oracle Server to be allocated to new active transactions.

Oracle 11g automatically tunes undo retention by collecting database use statistics.

- Specifying UNDO\_RETENTION sets a low threshold so that undo data is retained at a minimum for the threshold value specified, providing there is sufficient Undo tablespace capacity.
- The RETENTION GUARANTEE clause of the CREATE UNDO TABLESPACE statement can guarantee retention of Undo data to support DML operations, but may cause database failure if the Undo tablespace is not large enough – unexpired Undo data segments are not overwritten.
- The TUNED\_UNDORETENTION column of the V\$UNDOSTAT dynamic performance view can be queries to determine the amount of time Undo data is retained for an Oracle database.
- Query the RETENTION column of the DBA\_TABLESPACES view to determine the setting for the Undo tablespace – possible values are GUARANTEE, NOGUARANTEE, and NOT APPLY (for tablespaces other than Undo).

## 15.3 Sizing and Monitoring an Undo Tablespace

Three types of Undo data exists in an Undo tablespace:

- Active (unexpired) – these segments are needed for read consistency even after a transaction commits.
- Expired – these segments store undo data that has been committed and all queries for the data are complete and the undo retention period has been reached.
- Unused – these segments have space that has never been used.

The minimum size for an Undo tablespace is enough space to hold before-image versions of all active transactions that have not been committed or rolled back.

When space is inadequate to support changes to uncommitted transactions for rollback operations, the error message ORA-30036: Unable to extend segment by space\_qtr in undo tablespace tablespace\_name is displayed, and the DBA must increase the size of the Undo tablespace.

Initial Size – enable automatic extension (use the AUTOEXTEND ON clause with the CREATE TABLESPACE or ALTER TABLESPACE commands) for Undo tablespace datafiles so they automatically increase in size as more Undo space is needed.

- After the system stabilizes, Oracle recommends setting the Undo tablespace maximum size to about 10% more than the current size.
- The Undo Advisor software available in Enterprise Manager can be used to calculate the amount of Undo retention disk space a database needs.

## 16. Oracle Managed Files

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

| Tablespaces | Redo log files | Control files |
|-------------|----------------|---------------|
|-------------|----------------|---------------|

Through initialization parameters, we specify the file system directory to be used for a particular type of file. Oracle then ensures that a unique file, an Oracle-managed file, is created and deleted when no longer needed. With Oracle-managed files, bigfile tablespaces make datafiles completely transparent for users. In other words, we can perform operations on tablespaces, rather than the underlying datafile.

### 16.1 Benefits of Using Oracle-Managed Files

- Make the administration of the database easier.
- Reduce corruption caused by administrators specifying the wrong file.
- Reduce wasted disk space consumed by obsolete files.
- Simplify creation of test and development databases.
- Make development of portable third-party tools easier.
- Automatic cleanup of the filesystem when database objects are dropped.
- Standardized naming of database files.
- Increased portability since file specifications is not needed.
- Simplified creation of test systems on differing operating systems.
- No unused files wasting disk space.

### 16.2 File Location

The location of database files is defined using the DB\_CREATE\_FILE\_DEST parameter. If it is defined on its own all files are placed in the same location. If the DB\_CREATE\_ONLINE\_LOG\_DEST\_n parameter is defined alternate locations and levels of multiplexing can be defined for Logfiles. These parameters are dynamic and can be changed using the ALTER SYSTEM statement:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/disk1/oradata/ORCL/WST1';
```

Files typically have a default size of 100M and are named using the following formats where %u is a unique 8 digit code, %g is the logfile group number, and %t is the tablespace name:

| File Type           | Format        |
|---------------------|---------------|
| Controlfiles        | Ora_%u.ctl    |
| Redo Log Files      | Ora_%g_%u.log |
| Datafiles           | Ora_%t_%u.dbf |
| Temporary Datafiles | Ora_%t_%u.tmp |

## 16.3 Managing Different Files in OMF

### 16.3.1 Managing Controlfiles Using OMF

During DB creation the controlfile names are not specified. Instead, a controlfile is created for each DB\_CREATE\_ONLINE\_LOG\_DEST\_n specified in the init.ora file. Once the DB creation is complete the CONTROL\_FILES parameter can be set in init.ora file using the generated names shown in V\$CONTROLFILE view.

### 16.3.2 Managing Redo Log Files Using OMF

When using OMF for redo logs the DB\_CREATE\_ONLINE\_LOG\_DEST\_n parameters in the init.ora file decide on the locations and numbers of logfile members. For example:

- DB\_CREATE\_ONLINE\_LOG\_DEST\_1 = /disk1/oradata/ORCL/WST1
- DB\_CREATE\_ONLINE\_LOG\_DEST\_2 = /disk2/oradata/ORCL/WST1

The above parameters mean two members will be created for the logfile group in the specified locations when the ALTER DATABASE ADD LOGFILE; statement is issued. Oracle will name the file and increment the group number if they are not specified. The ALTER DATABASE DROP LOGFILE GROUP 3; statement will remove the group and its members from the database and delete the files at operating system level.

### 16.3.3 Managing Datafiles and Tempfiles

Include this initialization parameter in our initialization parameter file to specify the default location for creating the Oracle Managed Datafiles and Tempfiles.

- DB\_CREATE\_FILE\_DEST='/disk1/oradata/ORCL'

This parameter is dynamic and can also be set with the ALTER SYSTEM or ALTER SESSION command. Once this parameter is set, we need not specify file names when creating TBS, undo tablespaces, and Temporary tablespaces.

#### Examples:

```
SQL> CREATE TABLESPACE user_data;
SQL> CREATE UNDO TABLESPACE undotbs_ts;
SQL> CREATE TEMPORARY TABLESPACE temp_ts;
```

In all the above examples Oracle creates a file with a unique name with a size of 100M with auto extend on. We can overwrite the default file size by explicitly using the clause SIZE followed by the size we want.

```
SQL> CREATE TABLESPACE user_data2 DATAFILE SIZE 50M;
```

We can add a datafile to the tablespace using the ALTER TABLESPACE command.

```
SQL> ALTER TABLESPACE user_data2 ADD DATAFILE SIZE 50M;
```

Similarly in case of Temporary tablespace we say:

```
SQL> ALTER TABLESPACE temp_ts ADD TEMPFILE SIZE 50M;
```

When a tablespace is dropped all its datafiles or tempfiles will be automatically dropped.

### 16.3.4 Managing Tablespaces Using OMF

As shown previously the DB\_CREATE\_FILE\_DEST parameter in the init.ora file specifies the location of the datafiles for OMF tablespaces. Since the file location is specified and Oracle will name the file, new tablespaces can be created using the following statement:

```
SQL> CREATE TABLESPACE wst_data;
```

The resultant datafiles will have a default size of 100M and AUTOEXTEND UNLIMITED. For a specific size file use:

```
SQL> CREATE TABLESPACE wst_data DATAFILE SIZE 150M;
```

**To add a datafile to a tablespace use:**

```
SQL> ALTER TABLESPACE wst_data ADD DATAFILE;
```

If a tablespace is dropped, Oracle will remove the OS files also. For tablespaces not using the OMF feature this cleanup can be performed by issuing the statement:

```
SQL> DROP TABLESPACE wst_data INCLUDING CONTENTS AND DATAFILES;
```

### 16.3.5 Default Temporary Tablespace

In previous releases, if we forgot to assign a temporary tablespace to a user the SYSTEM tablespace was used. This can cause contention and is considered bad practice. To prevent this 9i gives us the ability to assign a default temporary tablespace. If a temporary tablespace is not explicitly assigned the user is assigned to this tablespace. A default temporary tablespace can be created during database creation or assigned afterwards:

```
SQL> CREATE DATABASE TSH1
 ...
 DEFAULT TEMPORARY TABLESPACE dts1
 TEMPFILE '/disk1/oradata/ORCL/dts_1.f' SIZE 20M
 EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
-- or
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE dts2;
```

A default temporary tablespace cannot be taken offline until a new default temporary tablespace is brought online.

## 16.4 File Location for Controlfiles and Logfiles

Include this initialization parameter in our initialization parameter file to specify the default location for creating the Oracle Managed logfile and Controlfiles. Here 'n' ranges from 1 through 5.

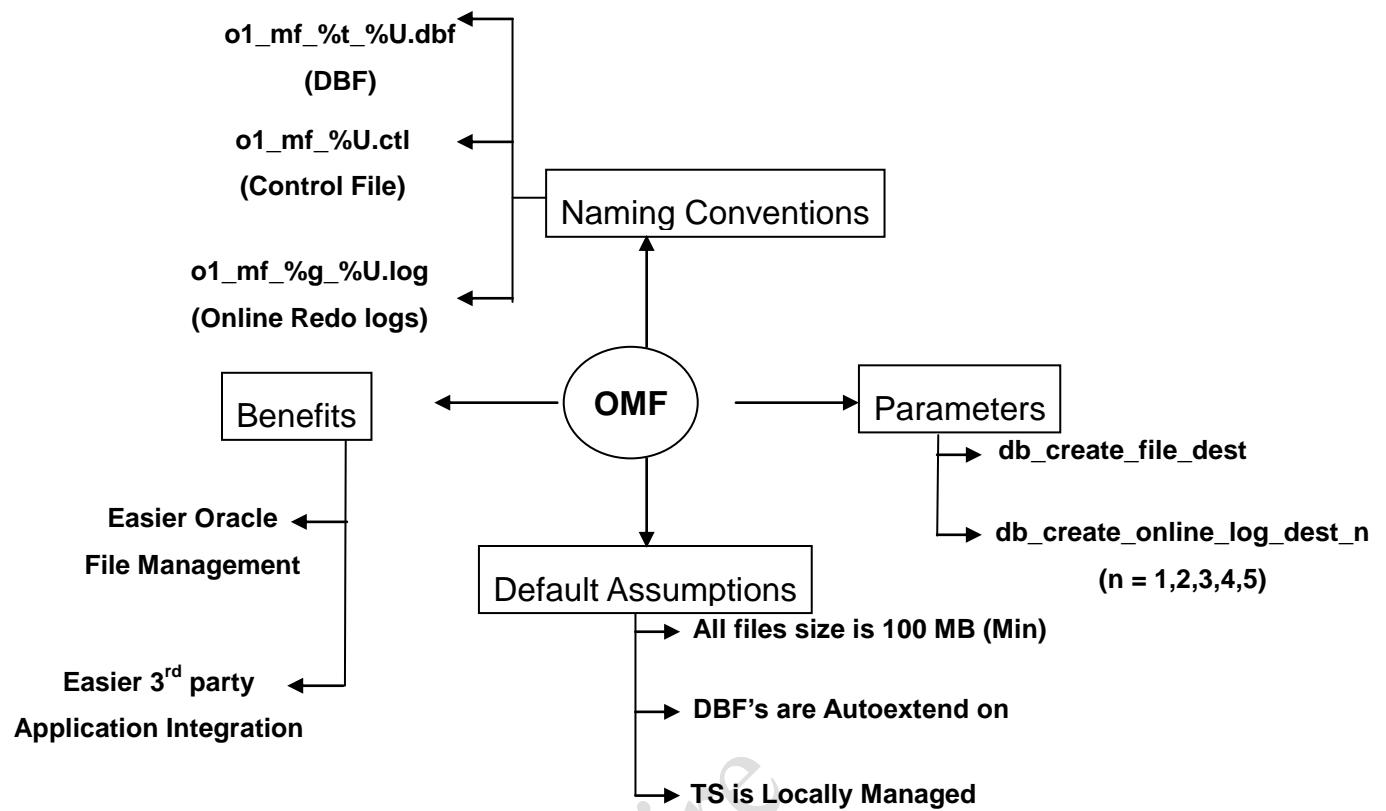
```
DB_CREATE_ONLINE_LOG_DEST_1='/disk1/oradata/ORCL'
DB_CREATE_ONLINE_LOG_DEST_2='/disk1/oradata/ORCL'
```

**Examples:**

```
SQL> ALTER DATABASE ADD LOGFILE group 3;
```

This command creates the logfiles in all the destinations specified by DB\_CREATE\_ONLINE\_LOG\_DEST\_n.

```
SQL> ALTER DATABASE ADD LOGFILE MEMBER TO group 4;
```



## 17. Server Parameter File (SPFile)

### 17.1 Overview

SPFILE stands for Server Parameter File. Oracle requires an initialization file to define the attributes and characteristics of the starting instance and the connecting sessions they're of. We are aware of the Initialization parameter file (init.ora file or PFILE). This file holds the setup parameters that define the attribute of the instance that is being started.

Administrators can control, tune and optimize an instance by setting and modifying the initialization parameters in this file. Some parameters can be dynamically modified to affect the present instance, while others require the instance to be brought down so that changes can take effect.

This remains the same when using PFILE or SPFILE. A simple search on the net will reveal a lot of information regarding PFILE and SPFILE. Parameter files are by default situated in \$ORACLE\_HOME/dbs (on UNIX) or %ORACLE\_HOME%\database (on Windows) directory.

### 17.2 Benefits and Features of SPFILE

SPFILE is an extension of initialization parameter storage mechanism, which allows some additional advantages as compared to simple text based init.ora file. This feature aims at advancing towards Oracle's self-tuning capabilities. As the name states, the file is present on the server side and is not required on the client to start the instance.

- It is a binary file and the use of editors to modify it is not supported. The only way of changing parameter values is by using the ALTER SYSTEM SET/RESET command.
- If using the PFILE option, parameter changes done with the ALTER SYSTEM command need to be manually changed in the init.ora file to reflect the changes on future startups. With SPFILE, the ALTER SYSTEM command can update the binary file as well. This allows the changes to be persistent across startups.
- Since it is binary, it is prone to becoming corrupt if things go wrong. As a backup strategy, create copies of the SPFILE and convert it to a text-based PFILE for safekeeping.
- The ISSYS\_MODIFIABLE column in V\$PARAMETER tells us whether the parameters are static or dynamic. Static parameters require instance to be restarted while dynamic parameters can take effect immediately upon being changed.

```
SQL> SELECT DISTINCT issys_modifiable FROM v$parameter;
ISSYS_MODIFIABLE

DEFERRED
FALSE
IMMEDIATE
```

If the ISSYS\_MODIFIABLE value is set to FALSE for a parameter, it means that the parameter cannot change its value in the lifetime of the instance; the DB needs to be restarted for changes to take effect. A parameter set to IMMEDIATE value means that it is dynamic and can be set to change the present active instance as well as future DB restarts. A parameter set to DEFERRED is also dynamic, but changes only affect subsequent sessions, currently active sessions will not be affected and retain the old parameter value.

- PFILE is required from the client from where the instance is being started. SPFILE is not required to be from the client from where the instance is started. This comes in handy when we are trying to start DB remotely.
- If you have a standby database, the primary database parameter changes are not propagated to the standby. This needs to be done manually and SPFILE does not aid in this.
- If no SPFILE exists and one needs to be created, Oracle requires that the instance should be started at least once with the PFILE and then the SPFILE created. SPFILE is the default mode in new releases.
- In multi-instance Real Application cluster system, a single copy of the SPFILE can be used for all instances. The SPFILE maintains different format styles to support both the common values for all instances as well as specific values for individual instances.
- The contents of SPFILE can be obtained from V\$SPPARAMETER view. The parameters having ISSPECIFIED column set to TRUE are the ones present in the binary file.

- We only need to be connected as SYSDBA/SYSOPER to create SPFILE or PFILE. Database need not be started. This option is useful in case the SPFILE has been corrupted and you need to rebuild it from a PFILE.

## 17.3 Which SPFILE on startup

If you have multiple SPFILEs, it will be confusing to identify which file is linked to which instance. By default, the SPFILE name is spfile<sid>.ora. This information can be identified by looking at the below logs/views:

1. The alert log can be looked at. If log shows SPFILE parameter in "System parameters with non-default value" section, then it is clear that instance was started with PFILE internally calling SPFILE using the parameter.
2. Check the SPFILE parameter value in V\$PARAMETER view.

**ALTER SYSTEM command for SPFILE:** ALTER SYSTEM command can modify SPFILE. This command allows us to set and reset the parameter values.

```
SQL> ALTER SYSTEM SET <parameter>=<value>
SCOPE=<memory/spfile/both> COMMENT=<'comments'> DEFERRED SID=<sid,*>
```

The SCOPE clause of command decides how the changes will take effect. This option can have the following values:

- **MEMORY** - changes are active for the current instance only and are lost on restart.
- **SPFILE** - changes are stored in the SPFILE and are activated on the next startup, the presently active instance is not affected. Static parameters need this option to change their values.
- **BOTH** - changes are effective immediately for the present instance and are stored in SPFILE for future startups. This is the default.
- **COMMENT** - is optional and can be specified to store remarks or audit information.
- **DEFERRED** - option is used to set parameter values for future connecting sessions. Currently active sessions are not affected and they retain the old parameter value. The option is required for parameters that have the ISSSYS\_MODIFIABLE column value in V\$PARAMETER set to 'DEFERRED'. It is optional if the ISSSYS\_MODIFIABLE value is set to 'IMMEDIATE'. For static parameters, this cannot be specified.

The SID clause is valid for Real Application Clusters only. Setting this to a specific SID value changes the parameter value for that particular instance only. Setting this to '\*' will affect all instances on the cluster--this is the default if the instance was started using the SPFILE. If the instance was started using PFILE then Oracle assumes the current instance as default.

## 17.4 On starting the Instance

When an instance is started, if SPFILE is present, it will be used; otherwise, Oracle searches for a default PFILE. If a PFILE is explicitly specified in the STARTUP command Oracle uses it.

```
SQL> STARTUP PFILE=initdb1.ora
```

In the above case, the PFILE can also have an SPFILE parameter set so that a different SPFILE can be used to start the instance rather than the default one. There is no option to specify an SPFILE manually while starting the database. This is identified and picked up by Oracle either by using the SPFILE parameter in PFILE or by searching the default paths.

## 17.5 Creating an SPFILE

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

```
SQL> CREATE SPFILE FROM PFILE; (or)
SQL> CREATE SPFILE=<file_name> FROM PFILE=<source File_name>;
```

On Linux, default spfile name will be spfile<SID>.ora and will be located in \$ORACLE\_HOME/dbs. DB must be restarted before the spfile takes effect. Once instance is started with spfile, we can use ALTER SYSTEM command to change init.ora parameters and these parameters will be persistent across startup and shutdown.

```
SQL> ALTER SYSTEM SET <init.ora Parameter> = <value>SCOPE=[BOTH | SPFILE | MEMORY];
```

- BOTH - Change is applied in both the server parameter file and to the current instance
- SPFILE-Change is applied in the spfile only and will be affective from next STARTUP & remains persistent.
- MEMORY - Change is applied only to the current Instance.

**Example:** SQL> ALTER SYSTEM SET db\_cache\_size=50m;

## 18. Oracle Networking

### 18.1 Overview

Oracle Net Services provides enterprise wide connectivity solutions in distributed, heterogeneous computing environments. Additionally, it eases the complexities of network configuration and management, maximizes performance, and improves network security and diagnostic capabilities in the following areas:

- Connectivity
- Manageability
- Performance and Scalability
- Network Security
- Diagnosability

#### 18.1.1 Connectivity

Oracle Net, a component of Oracle Net Services, enables a network session from a client application to an Oracle database server. Once a network session is established, Oracle Net acts as the data courier for both the client application and the database server. It is responsible for establishing and maintaining the connection between the client application and database server, as well as exchanging messages between them. Oracle Net is able to perform these jobs because it is located on each computer in the network.

#### 18.1.2 Manageability

Oracle Net Services offers a number of manageability features that enable you to easily configure and manage networking components. These features are described in the following three areas:

**Location Transparency:** A client uses this service name to identify the database it needs to access. The information about the database service and its location in the network is transparent to the client because the information needed for a connection is stored in a repository. To achieve this goal, several naming methods are available today: Oracle Net Directory naming, Local naming (TNSNAMES.ORA), Host naming, and External naming.

**Centralized Configuration and Management:** To manage large networking environments, administrators have to be able to easily access a centralized repository to specify and modify the network configuration. For this reason, the Oracle Net Services configuration can be stored in an LDAP-compliant directory server such as Oracle Internet Directory.

**Quick Installation and Configuration:** Oracle Net Services installs quickly and easily. Networking components for the Oracle database server and clients are pre-configured for most environments. Information about an Oracle database service is resolved in one or more naming methods. As a result, clients and servers are ready to immediately connect when installed.

#### 18.1.3 Performance and Scalability

Oracle Net provides scalability features that enable user to maximize system resources and improve performance. Feature such as shared server, which offers both connection pooling and session multiplexing (using Oracle Connection Manager), can increases the scalability of applications and the number of clients that can be simultaneously connected to the database. The shared server architecture also enables existing applications to scale up without making any changes to the application itself.

Another major performance improvement is using high speed interconnect technologies, such as InfiniBand, which can significantly improve network performance.

#### 18.1.4 Network Security

Granting and denying access to a database is crucial for a secure network environment. Oracle Net Services enables database access control using features of firewall access control and protocol access control.

### 18.1.5 Diagnosability

Oracle Net Services provides detailed information about the source and context of problems as they arise. This information is generated and stored in log and trace files. The process of logging and tracing error information will help you to diagnose and resolve network problems. Trace Assistant, a diagnostic and performance analysis tool, is provided with Oracle Net Services.

Oracle Net Services enables a network session from a client application to an Oracle database. In the Oracle database system environment, the database application and the database are separated into two parts:

- A front-end or client computer, and
- A back-end or server computer - hence the term client/server architecture.

The client runs the Database application that accesses database information and the server will be running the Oracle Database.

So, when a Client/Application wants to speak to the Oracle Database, a service called a Listener should be running on the node running the Oracle Database. The listener configuration is stored in a configuration file named listener.ora located in \$ORACLE\_HOME/network/admin. In this file we will be configuring parameters like: the Listener Name, Protocol, Host name or I.P. address of node running the Oracle database, and the SID (Instance name) for which the listener is serving (accepting in-coming calls).

Similarly, we need to configure an alias or the service name at the client side. The alias configuration is stored in a configuration file named tnsnames.ora located in \$ORACLE\_HOME/network/admin. Again in this file we need to specify the Alias name, Protocol, Host Name or I.P. of the database server, and Service name.

## 18.2 Establishing Connectivity

The task here is to show a TCP/IP connection between a client computer (I.P. 192.168.0.12) and a Database server computer (I.P. 192.168.0.15). The following about the database server and client computers is assumed:

### Database Server Computer

- It is running on the same network as the client.
- An Oracle database is installed.
- TCP/IP protocol support is installed.
- A listener is configured.

### Client Computer

- It is running on the same network as the database server.
- Oracle Client is installed.
- TCP/IP protocol support is installed.

## 18.3 Configuring the Server with Listener:

```
$ cat listener.ora
LIST_ORCL =
(DESCRIPTION_LIST =
 (DESCRIPTION =
 (ADDRESS = (PROTOCOL = TCP) (HOST = 192.168.0.15) (PORT = 1521))
)
)
SID_LIST_LIST_ORCL =
(SID_LIST =
 (SID_DESC =
 (SID_NAME = ORCL)
 (ORACLE_HOME = /oraeng/app/oracle/product/15.1.0)
)
)
```

In the above listener configuration the listener name is LIST\_ORCL, the protocol it uses is TCP, the host on which it is running is 192.168.0.12, Port at which it listens is 1521 and the Instance name is ORCL.

The listener service can be started with the utility lsnrctl

```
$ lsnrctl start list_orcl
```

When an instance starts, a listener process establishes a communication pathway to Oracle. When a user process makes a connection request, the listener determines whether it should use a shared server dispatcher process or a dedicated server process and establishes an appropriate connection.

## 18.4 Configuring the Client with Alias:

```
$ cat tnsnames.ora
to_orcl =
 (DESCRIPTION =
 (ADDRESS_LIST =
 (ADDRESS = (PROTOCOL = TCP) (HOST = 192.168.0.12) (PORT = 1521))
)
 (CONNECT_DATA =
 (SERVICE_NAME = ORCL)
)
)
```

In the above configuration file, the alias name is to\_orcl, Protocol in use is TCP, I.P of the server it has to connect to is 192.168.0.12, the Port it should reach is 1521 and the Instance it should connect to is ORCL.

To check the connectivity from the client to the server we can use the utility tnsping. The TNSPING utility determines whether or not a service (for example, an Oracle database service) on an Oracle Net network can be successfully reached.

If you can connect successfully from a client to a server using the TNSPING utility, it displays an estimate of the round trip time (in milliseconds) it takes to reach the Oracle Net service.

If it fails, it displays a message describing the error that occurred. This enables you to see the network error that is occurring without the overhead of a database connection.

```
$ tnsping to_orcl
```

So, now let us assume user SCOTT wants to connect to the Database Server using this alias.

```
$ sqlplus scott/tiger@to_orcl
```

The listener and Oracle Database must be running in order for the connection to be successful.

On the Database you can check for this session in two ways:

- Query V\$SESSION

```
SQL> SELECT username, osuser, machine, terminal
 FROM V$SESSION;
Using command, ps x (check for LOCAL = NO)
```

## 19. Databases Links

Distributed Environment is a network of disparate systems that seamlessly communicate with each other. Each system in the distributed environment is called a node. The system to which a user is directly connected is called local system. Any additional systems accessed by this user are called remote systems. Distributed environment allows applications to access and exchange data from local and remote systems. All the data can be simultaneously accessed and modified. While a distributed environment enables increased access to a large amount of data across a network, it must also hide the location of the data and the complexity of accessing it across the network. In order for a company to operate successfully in a distributed environment, it must be able to do the following:

- Exchange data between Oracle databases
- Communicate between applications
- Exchange information with customers, partners, and suppliers
- Replicate data between databases
- Communicate with non-Oracle databases

Oracle's distributed database architecture also provides query, update, and transaction transparency. For example, standard SQL statements such as SELECT, INSERT, UPDATE, and DELETE work just as they do in a non-distributed database environment. Additionally, applications control transactions using the standard SQL statements COMMIT, SAVEPOINT, and ROLLBACK. Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because Oracle must coordinate the committing or undo of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

### 19.1 Location Transparency

Location transparency occurs when the physical location of data is transparent to the applications and users. For example, a view that joins table data from several databases provides location transparency because the user of the view does not need to know from where the data originates.

### 19.2 Database link

Oracle uses database links to enable users on one database to access objects in a remote database. A local user can access a link to a remote database without having to be a user on the remote database. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

Database Links are of two types:

1. Private database link: A database link created by one user for his or her exclusive use.
2. Public database link: A database link created by a DBA on a local database that is accessible to all users on that database.

Normal user by default will not have permission to create a database link. We as a DBA have to grant permission to the users for creating the DB link using command:

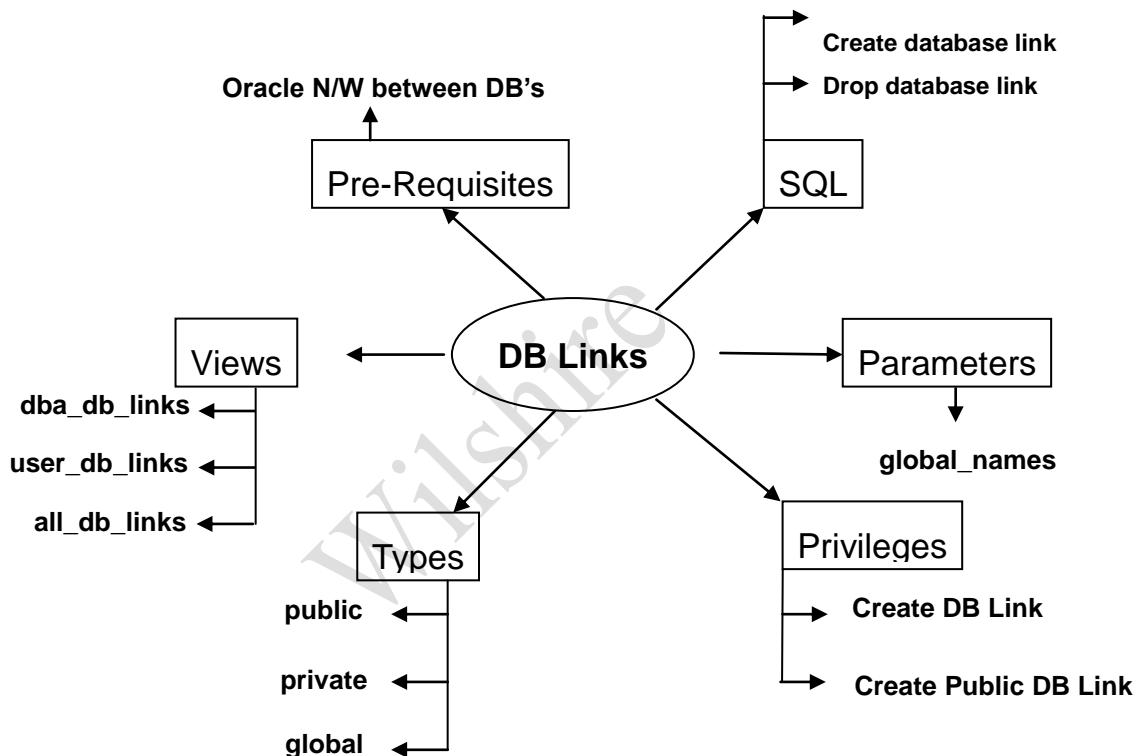
```
SQL> GRANT CREATE DATABASE LINK TO scott;
```

Consider a scenario, where the user scott is on database DB1 and wish to connect to user steve on database DB2. So, a listener needs to be configured and started on the server DB2 (remote) and an alias needs to be configured on the Server DB1 (local). So scott on A can give the command like:

```
SQL> CREATE DATABASE LINK db_lnk1
 CONNECT TO steve IDENTIFIED BY steve
 USING 'to_db2';
```

Where to\_db2 is alias connecting to database DB2. Now scott can access the steve's objects using database link. SQL> SELECT \* FROM emp@db\_lnk1;

Note: DDL operations are not supported using the Database Links.



## 20. Materialized Views

Materialized views are schema objects that can be used to summarize, compute, replicate, and distribute data. They are suitable in various computing environments such as data warehousing, decision support, and distributed or mobile computing. In data warehouses, materialized views are used to compute and store aggregated data such as sums and averages. In distributed environments, materialized views are used to replicate data at distributed sites and synchronize updates done at several sites with conflict resolution methods.

Materialized views are periodically updated, or refreshed, from their associated master tables through transactionally consistent batch updates.

Unlike a

normal ordinary view, which does not take up any storage space or contain any data, a materialized view contains the rows resulting from a query against one or more base tables or views. A materialized view can be stored in the same database as its base tables or in a different database.

Because materialized views do not require a dedicated connection, they are ideal for disconnected computing. For example, a company might choose to use updatable materialized views for the members of their sales force. A salesperson could enter orders into his or her laptop throughout the day, and then simply dial up the regional sales office at the end of the day to upload these changes and download any updates.

### 20.1 Materialized View Logs

A materialized view log is a schema object that records changes to a master table's data so that a materialized view defined on the master table can be refreshed incrementally.

Consider a scenario where the sales information is in a table sales\_master on the production box. We want to create a materialized view for this table in a local DSS box and we want the data to be refreshed every 12hrs going with FAST option.

Assume the listener is configured and started on the Production box and an alias is configured on the DSS pointing to the listener service. Also let us consider a database link db\_Lnk1 is created on the DSS box. We can create a materialized view as:

```
SQL> CREATE MATERIALIZED VIEW sales_mv
 REFRESH FAST
 STARTWITH SYSDATE
 NEXT SYSDATE+(1/12)
 AS SELECT * FROM sales_master@db_lnk1;
```

Note: If you are going with the Fast refresh option, you need to create a materialized view log on the master table i.e., on the production site and also you need to start some CJQ (Coordinator Job Queue) Processes on the client site i.e., on the DSS site.

The other refresh options that are available are: COMPLETE, FORCE, NEVER.

Create a materialized view log as:

```
SQL> CREATE MATERIALIZED VIEW LOG ON sales_master;
```

Start the CJQ Processes as:

```
SQL> ALTER SYSTEM SET job_queue_processes=5;
```

Or include the parameter in the init<SID>.ora as:

```
JOB_QUEUE_PROCESSES = 5
```

So, the CJQ processes fetch the data from the materialized view logs and refresh them in the materialized view at the specified refresh intervals.

## 20.2 Refresh Clause

The refresh option specifies:

1. The refresh method used by Oracle to refresh data in materialized view
2. Whether the view is primary key based or row-id based
3. The time and interval at which the view is to be refreshed

### 20.2.1 Fast Clause

FAST refreshes use materialized view logs to send rows that have changed from master tables to the materialized view. We should create a materialized view log for the master tables if we specify the REFRESH FAST clause.

```
SQL> CREATE MATERIALIZED VIEW LOG ON emp;
```

Materialized view log created.

Materialized views are not eligible for fast refresh if the defined subquery contains an analytic function.

### 20.2.2 Complete Clause

COMPLETE refresh re-creates the entire materialized view. If you request a complete refresh, Oracle performs a complete refresh even if a fast refresh is possible.

### 20.2.3 Force Clause

When we specify a FORCE clause, Oracle will perform a fast refresh if one is possible or a complete refresh otherwise. If we do not specify a refresh method (FAST, COMPLETE, or FORCE), FORCE is the default.

#### Primary Key and RowID Clause

WITH PRIMARY KEY is used to create a primary key materialized view i.e. the materialized view is based on the primary key of the master table instead of ROWID (for ROWID clause). PRIMARY KEY is the default option. To use the PRIMARY KEY clause you should have defined PRIMARY KEY on the master table or else you should use ROWID based materialized views.

Primary key materialized views allow materialized view master tables to be reorganized without affecting the eligibility of the materialized view for fast refresh.

Rowid materialized views should have a single master table and cannot contain any of the following:

- Distinct or aggregate functions
- GROUP BY Subqueries , Joins & Set operations

## 20.3 Refresh Modes

- ON COMMIT – refreshes occur whenever a commit is performed on one of the view's underlying detail table(s)
  - **Available only with single table aggregate or join based views**
  - **Keeps view data transactionally accurate**
  - **Need to check alert log for view creation errors**
- ON DEMAND – refreshes are initiated manually using one of the procedures in the DBMS\_MVIEW package
  - **Can be used with all types of materialized views**
  - **Manual Refresh Procedures**

- DBMS\_MVIEW.REFRESH(<mv\_name>, <refresh\_option>)
- DBMS\_MVIEW.REFRESH\_ALL\_MVIEWS()
- START WITH [NEXT] <date> – refreshes start at a specified date/time and continue at regular intervals

## 20.4 Out-of-Place refresh for Mviews (New in 12c)

The database maintains data in materialized views by refreshing them after changes to the base tables. The refresh method can be incremental or a complete refresh.

There are two incremental refresh methods, known as

Log-based refresh and

Partition change tracking (PCT) refresh.

The incremental refresh is commonly called **FAST** refresh as it usually performs faster than the complete refresh.

A complete refresh occurs when the materialized view is initially created when it is defined as **BUILD IMMEDIATE**, unless the materialized view references a prebuilt table or is defined as **BUILD DEFERRED**.

Users can perform a complete refresh at any time after the materialized view is created. The complete refresh involves executing the query that defines the materialized view. This process can be slow, especially if the database must read and process huge amounts of data.

An incremental refresh eliminates the need to rebuild materialized views from scratch. Thus, processing only the changes can result in a very fast refresh time. Materialized views can be refreshed either on demand or at regular time intervals. Alternatively, materialized views in the same database as their base tables can be refreshed whenever a transaction commits its changes to the base tables.

For materialized views that use the log-based fast refresh method, a materialized view log and/or a direct loader log keep a record of changes to the base tables. A materialized view log is a schema object that records changes to a base table so that a materialized view defined on the base table can be refreshed incrementally. Each materialized view log is associated with a single base table. The materialized view log resides in the same database and schema as its base table.

The PCT refresh method can be used if the modified base tables are partitioned and the modified base table partitions can be used to identify the affected partitions or portions of data in the materialized view. When there have been some partition maintenance operations on the base tables, this is the only incremental refresh method that can be used. The PCT refresh removes all data in the affected materialized view partitions or affected portions of data and recomputes them from scratch.

### 20.4.1 Out-of-Place Materialized Views

For each of these refresh methods, you have two options for how the refresh is performed, namely **in-place refresh** and **out-of-place refresh**.

The in-place refresh executes the refresh statements directly on the materialized view.

The out-of-place refresh creates one or more outside tables and executes the refresh statements on the outside tables and then switches the materialized view or affected materialized view partitions with the outside tables.

Both in-place refresh and out-of-place refresh achieve good performance in certain refresh scenarios. However, the out-of-place refresh enables high materialized view availability during refresh, especially when refresh statements take a long time to finish.

Also adopting the out-of-place mechanism, a new refresh method called synchronous refresh is introduced in Oracle Database 12c, Release 1.

This refresh approach enables you to keep a set of tables and the materialized views defined on them to be always in sync.

In this refresh method, the user does not directly modify the contents of the base tables but must use the APIs provided by the synchronous refresh package that will apply these changes to the base tables and materialized views at the same time to ensure their consistency.

The synchronous refresh method is well-suited for data warehouses, where the loading of incremental data is tightly controlled and occurs at periodic intervals.

When creating a materialized view, you have the option of specifying whether the refresh occurs **ON DEMAND** or **ON COMMIT**. In the case of **ON COMMIT**, the materialized view is changed every time a transaction commits, thus ensuring that the materialized view always contains the latest data.

Alternatively, you can control the time when refresh of the materialized views occurs by specifying **ON DEMAND**. In the case of **ON DEMAND** materialized views, the refresh can be performed with refresh methods provided in either the **DBMS\_SYNC\_REFRESH** or the **DBMS\_MVIEW** packages.

Performing a refresh operation requires temporary space to rebuild the indexes and can require additional space for performing the refresh operation itself.

There are three types of out-of-place refresh:

- **out-of-place fast refresh**
  - This offers better availability than in-place fast refresh. It also offers better performance when changes affect a large part of the materialized view.
- **out-of-place PCT refresh**
  - This offers better availability than in-place PCT refresh. There are two different approaches for partitioned and non-partitioned materialized views. If truncation and direct load are not feasible, you should use out-of-place refresh when the changes are relatively large. If truncation and direct load are feasible, in-place refresh is preferable in terms of performance. In terms of availability, out-of-place refresh is always preferable.
- **out-of-place complete refresh**
  - This offers better availability than in-place complete refresh.

## 21. Distributed Transactions

### 21.1 Overview

In recent years, the availability of databases and of computers networks has given rise to a new field, which is Distributed Database. A distributed Database is integrated database, which is built on top of a computer rather than on a single computer.

The data, which constitute the database, are stored at the different sites of the computer network, and the application programs, which are run by the computers, access data at different sites. Databases may involve different database management systems, running on different architectures that distribute the execution of transactions.

In a distributed database environment, a given transaction updates data in multiple physical databases protected by two-phase commit to ensure all nodes or none. Oracle ensures the integrity of data in a distributed transaction using the two-phase commit mechanism

A Distributed Database Management System (DDBMS) is defined as the software that handles the management of the DDB (Distributed Database) and makes the operation of such a system appears to the user as if it were a centralized database.

Distributed Database is a collection of data, which belongs logically to the same system but are spread over the sites of a computer network.

1. Data is stored at several sites, each managed by DBMS that can run independently
2. The data from different sites are tied together using the computer network.
3. On each site there are two types of data
4. Data required for local site
5. Data required for other sites.

Following two properties are satisfied by distributed database:

- Distributed Data Independence: Users should not have to know where data is located (extends Physical and Logical Data Independence principles)
- Distributed Transaction Atomicity: Users should be able to write Xacts accessing multiple sites just like local Xacts.

### 21.2 Advantages of RDBMS

- Data is located near the greatest demand site, access is faster, processing is faster due to several sites spreading out the work load, new sites can be added quickly and easily, communication is improved, operating costs are reduced, it is user friendly, there is less danger of a single-point failure, and it has process independence.
- Reliable and flexible interconnection of existing database, and the future incremental growth. Distributed database is more suitable solution when several databases already exist in an organization.
- Performing global application can be easily performed with distributed database. If an organization grows by adding new relatively independent organizational units, then the distributed database approach support a smooth incremental growth.
- Data can physically reside nearest to where it is most often accessed, thus providing users with local control of data that they interact with. This result in local autonomy of the data allowing users to enforce locally the policies regarding access to their data.
- Parallel architecture to improve reliability and availability of the data in a scalable system. In a distributed system, with some careful tact, it is possible to access some or possibly all of the data in a failure mode if there is sufficient data replication.

## 21.3 Disadvantages of RDBMS

- Managing and controlling is complex, there is less security because data is at so many different sites.
- Distributed Database provides more flexible accesses that increase the chance of security violations since the database can be accessed throughout every site within the network.
- If there are multiple copies of the same data, then this duplicated data introduces additional complexity in ensuring that all copies are updated for each update. The notion of concurrency control and recoverability consume much of the research efforts in the area of distributed database theory. Increasing in reliability and performance is the goal and not the status quo.

## 21.4 The Two-Phase Commit Mechanism

A two-phase commit mechanism guarantees that all database servers participating in a distributed transaction either all commit or all undo the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

The Oracle two-phase commit mechanism is completely transparent to users who issue distributed transactions. In fact, users need not even know the transaction is distributed. A COMMIT statement denoting the end of a transaction automatically triggers the two-phase commit mechanism to commit the transaction. No coding or complex statement syntax is required to include distributed transactions within the body of a database application.

## 21.5 Recoverer Process (RECO)

The recoverer process is a background process used with the distributed database configuration that automatically resolves failures involving distributed transactions. The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When the RECO process reestablishes a connection between involved database servers, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved in-doubt transactions.

If the RECO process fails to connect with a remote server, RECO automatically tries to connect again after a timed interval. However, RECO waits an increasing amount of time (growing exponentially) before it attempts another connection. The RECO process is present only if the instance permits distributed transactions. The number of concurrent distributed transactions is not limited.

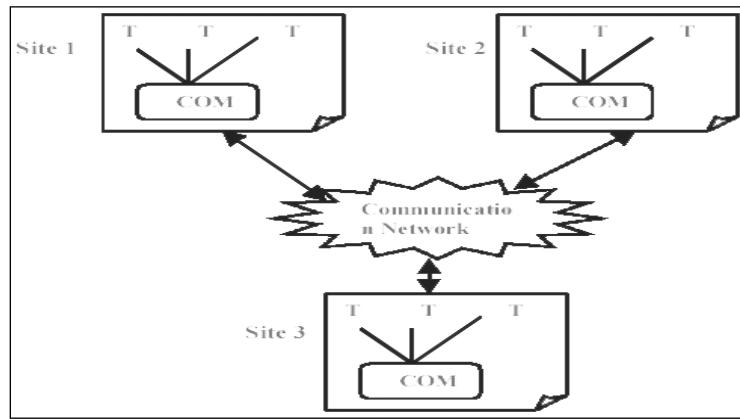
### Example

Consider the bank having three branches connected by computer network containing teller terminals of the branch and the account database of the branch. All the branches can process the database.

Each branch that has local database constitutes one site of the distributed database. The applications that are required from the terminals of a branch need only to access the database of that branch. Local applications are applications processed by the computer of the branch from where they are issued. A global application involves the generation of a number of sub-transactions, each of which may be executed at different sites.

For instance, suppose the customer wants to transfer the money from one account of one branch to an account of another branch. This application requires updating the database at two different branches. One has to perform two local updates at two different branches.

Thus if  $A$  is the data item with two copies  $A_1$  and  $A_2$  exists at sites  $S_1$  and  $S_2$ , then the operation of updating the value of  $A$  involves generating two sub-transactions  $T_{S1}$  and  $T_{S2}$ , each of which will perform the required operation from sites  $S_1$  and  $S_2$ . The branches are located at different geographical areas, and then the banking system is called distributed database on a geographical dispersed network.



The branches are situated in one building and are connected with a high bandwidth local network. The teller terminals of the branches are connected to their respective computer by telephone lines. Each processor and its database constitute a site of the local computer network. Here the distributed database is implemented on a local network instead of a geographical network and is called distributed database on a local network.

Let us now consider the same example as above. The data of the different branches are distributed on three backend computers, which perform the database management functions. The application programs are executed by different computer, which request database access services from the back ends when necessary.

This type of system is not distributed database because the data is physically distributed over different processors but their distribution is not relevant from the application viewpoint. Here the existence of local application on computers can't be executed.

From above examples, one can summarized and obtain the following working definition:

A distributed database is a collection of data, which are distributed over different computers of a computer network. Each site of the network has processing capacity and can perform local applications. Each site also participates in the execution of at least one global application.

- Can have many sites, each with their own DBMS.
- Local data stored at each site.
- Sites connected by communication network.
- To a user (application program) all sites together appear as one big DB.

## 22. Two Phase Commit Mechanism

### 22.1 Overview

Unlike a transaction on a local database, a distributed transaction involves altering data on multiple databases. Consequently, distributed transaction processing is more complicated, because the database must coordinate the committing or rolling back of the changes in a transaction as a self-contained unit. In other words, the entire transaction commits, or the entire transaction rolls back.

The database ensures the integrity of data in a distributed transaction using the two-phase commit mechanism. In the prepare phase, the initiating node in the transaction asks the other participating nodes to promise to commit or roll back the transaction. During the commit phase, the initiating node asks all participating nodes to commit the transaction. If this outcome is not possible, then all nodes are asked to roll back.

All participating nodes in a distributed transaction should perform the same action: they should either all commit or all perform a rollback of the transaction. The database automatically controls and monitors the commit or rollback of a distributed transaction and maintains the integrity of the global database (the collection of databases participating in the transaction) using the two-phase commit mechanism. This mechanism is completely transparent, requiring no programming on the part of the user or application developer.

### 22.2 Phases

The commit mechanism has the following distinct phases, which the database performs automatically whenever a user commits a distributed transaction:

| Phase         | Description                                                                                                                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Prepare phase | The initiating node, called the global coordinator, asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back. |
| Commit phase  | If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction.                                      |
| Forget phase  | The global coordinator forgets about the transaction.                                                                                                                                                                                                    |

#### 22.2.1 Prepare Phase

The first phase in committing a distributed transaction is the prepare phase. In this phase, the database does not actually commit or roll back the transaction. Instead, all nodes referenced in a distributed transaction (except the commit point site, described in the "Commit Point Site") are told to prepare to commit. By preparing, a node:

- Records information in the redo logs so that it can subsequently either commit or roll back the transaction, regardless of intervening failures
- Places a distributed lock on modified tables, which prevents reads

When a node responds to the global coordinator that it is prepared to commit, the prepared node promises to either commit or roll back the transaction later, but does not make a unilateral decision on whether to commit or roll back the transaction. The promise means that if an instance failure occurs at this point, the node can use the redo records in the online log to recover the database back to the prepare phase.

### 22.2.1.1 Types of Responses in the Prepare Phase

When a node is told to prepare, it can respond in the following ways:

| Response  | Meaning                                                                                                                   |
|-----------|---------------------------------------------------------------------------------------------------------------------------|
| Prepared  | Data on the node has been modified by a statement in the distributed transaction, and the node has successfully prepared. |
| Read-only | No data on the node has been, or can be, modified (only queried), so no preparation is necessary.                         |
| Abort     | The node cannot successfully prepare.                                                                                     |

#### Prepared Response

When a node has successfully prepared, it issues a prepared message. The message indicates that the node has records of the changes in the online log, so it is prepared either to commit or perform a rollback. The message also guarantees that locks held for the transaction can survive a failure.

#### Read-Only Response

When a node is asked to prepare, and the SQL statements affecting the database do not change any data on the node, the node responds with a read-only message. The message indicates that the node will not participate in the commit phase.

There are three cases in which all or part of a distributed transaction is read-only:

| Case                                          | Conditions                                                                                                                                                                   | Consequence                                                                                                                                                                                                                                                                                      |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Partially read-only                           | Any of the following occurs:<br>Only queries are issued at one or more nodes.<br>No data is changed.<br>Changes rolled back due to triggers firing or constraint violations. | The read-only nodes recognize their status when asked to prepare. They give their local coordinators a read-only response. Thus, the commit phase completes faster because the database eliminates read-only nodes from subsequent processing.                                                   |
| Completely read-only with prepare phase       | All of following occur:<br>No data changes.<br>Transaction is <i>not</i> started with SET TRANSACTION READ ONLY statement.                                                   | All nodes recognize that they are read-only during prepare phase, so no commit phase is required. The global coordinator, not knowing whether all nodes are read-only, must still perform the prepare phase.                                                                                     |
| Completely read-only without two-phase commit | All of following occur:<br>No data changes.<br>Transaction <i>is</i> started with SET TRANSACTION READ ONLY statement.                                                       | Only queries are allowed in the transaction, so global coordinator does not have to perform two-phase commit. Changes by other transactions do not degrade global transaction-level read consistency because of global SCN coordination among nodes. The transaction does not use undo segments. |

Note that if a distributed transaction is set to read-only, then it does not use undo segments. If many users connect to the database and their transactions are not set to READ ONLY, then they allocate undo space even if they are only performing queries.

## Abort Response

When a node cannot successfully prepare, it performs the following actions:

1. Releases resources currently held by the transaction and rolls back the local portion of the transaction.
2. Responds to the node that referenced it in the distributed transaction with an abort message.

These actions then propagate to the other nodes involved in the distributed transaction so that they can roll back the transaction and guarantee the integrity of the data in the global database. This response enforces the primary rule of a distributed transaction: all nodes involved in the transaction either all commit or all roll back the transaction at the same logical time.

### 22.2.1.2 Steps in the Prepare Phase

To complete the prepare phase, each node excluding the commit point site performs the following steps:

1. The node requests that its descendants, that is, the nodes subsequently referenced, prepare to commit.
2. The node checks to see whether the transaction changes data on itself or its descendants. If there is no change to the data, then the node skips the remaining steps and returns a read-only response (see "Read-Only Response").
3. The node allocates the resources it needs to commit the transaction if data is changed.
4. The node saves redo records corresponding to changes made by the transaction to its redo log.
5. The node guarantees that locks held for the transaction are able to survive a failure.
6. The node responds to the initiating node with a prepared response or, if its attempt or the attempt of one of its descendants to prepare was unsuccessful, with an abort response.

These actions guarantee that the node can subsequently commit or roll back the transaction on the node. The prepared nodes then wait until a COMMIT or ROLLBACK request is received from the global coordinator.

After the nodes are prepared, the distributed transaction is said to be **in-doubt**. It retains in-doubt status until all changes are either committed or rolled back.

## 22.2.2 Commit Phase

The second phase in committing a distributed transaction is the commit phase. Before this phase occurs, all nodes other than the commit point site referenced in the distributed transaction have guaranteed that they are prepared, that is, they have the necessary resources to commit the transaction.

### 22.2.2.1 Steps in the Commit Phase

The commit phase consists of the following steps:

1. The global coordinator instructs the commit point site to commit.
2. The commit point site commits.
3. The commit point site informs the global coordinator that it has committed.
4. The global and local coordinators send a message to all nodes instructing them to commit the transaction.
5. At each node, the database commits the local portion of the distributed transaction and releases locks.

6. At each node, the database records an additional redo entry in the local redo log, indicating that the transaction has committed.
7. The participating nodes notify the global coordinator that they have committed.

When the commit phase is complete, the data on all nodes of the distributed system is consistent.

### **Guaranteeing Global Database Consistency**

Each committed transaction has an associated system change number (SCN) to uniquely identify the changes made by the SQL statements within that transaction. The SCN functions as an internal timestamp that uniquely identifies a committed version of the database.

In a distributed system, the SCNs of communicating nodes are coordinated when all of the following actions occur:

- A connection occurs using the path described by one or more database links
- A distributed SQL statement executes
- A distributed transaction commits

Among other benefits, the coordination of SCNs among the nodes of a distributed system ensures global read-consistency at both the statement and transaction level. If necessary, global time-based recovery can also be completed.

During the prepare phase, the database determines the highest SCN at all nodes involved in the transaction. The transaction then commits with the high SCN at the commit point site. The commit SCN is then sent to all prepared nodes with the commit decision.

### **22.2.3 Forget Phase**

After the participating nodes notify the commit point site that they have committed, the commit point site can forget about the transaction. The following steps occur:

1. After receiving notice from the global coordinator that all nodes have committed, the commit point site erases status information about this transaction.
2. The commit point site informs the global coordinator that it has erased the status information.
3. The global coordinator erases its own information about the transaction.

## 23. Control File Management

### 215.1 Overview

The Control file is a small binary file necessary for the database to start and operate successfully. Each control file is associated with only one Oracle database. Before a database is opened, the control file is read to determine if the database is in a valid state to use. Oracle server updates the control file continuously during database is in use, so it must be available for writing whenever the database is open.

Only the Oracle server can modify the information in the control file; no database administrator or end user can edit the control file. If for some reason the control file is not accessible, the database does not function properly. If all copies of a database's control files are lost, the database must be recovered before it can be opened.

A Control file contains information such as:

- The database name
- The timestamp of database creation
- The names and locations of associated datafiles and redo log files
- Tablespace information
- Datafile offline status
- The log history
- Archived log information
- Backup set and backup piece information
- Backup datafile and redo log information
- Datafile copy information
- The current log sequence number
- Checkpoint information

Each time that a datafile or a redo log file is added to, renamed in, or dropped from the database, the control file is updated to reflect this physical structure change. These changes are recorded so that:

Oracle can identify the datafiles and redo log files to open during database startup

Oracle can identify files that are required or available in case database recovery is necessary

Therefore, if you make a change to the physical structure of your database (using ALTER DATABASE statements), then you should immediately make a backup of your control file.

### 211.2 Multiplexing the Control File

Oracle enables multiple, identical control files to be open concurrently and written for the same database. By storing multiple control files for a single database on different disks, you can safeguard against a single point of failure with respect to control files.

If a single disk that contained a control file crashes, then the current instance fails when Oracle attempts to access the damaged control file. However, when other copies of the current control file are available on different disks, an instance can be restarted without the need for database recovery.

If all control files of a database are permanently lost during operation, then the instance is aborted and media recovery is required.

### 215.3 Recreating the Controlfile

The size of the control file is influenced by the following keywords in the CREATE DATABASE or CREATE CONTROLFILE commands:

- MAXLOGFILES MAXLOGMEMBERS MAXLOGHISTORY MAXDATAFILES MAXINSTANCES

So, if ever we have to change the values of the above parameters we have to recreate the controlfile. Also when we want to change the name of the database

## 24. Redolog Files Management

### 24.1 Overview

Redo log files are crucial to the Oracle database. These are the transaction logs for the database. They are generally used only for recovery purposes, but they can be used for the following as well:

- Instance recovery after a system crash
- Media recovery after a data file restore from backup
- Standby database processing
- Input into Streams, a redo log mining process for information sharing (a fancy way of saying replication)

Redo log files provide the means to redo transactions in the event of a database failure. Every transaction is written synchronously to the redo log files in order to provide a recovery mechanism in case of media failure.

This includes transactions that have not yet been committed, undo segment information, and schema and object management statements. Redo log files are used in a situation such as an instance failure to recover committed data that has not been written to the data files. The redo log files are used only for recovery.

### 24.2 Characteristics of Redo Log Files:

- Redo log files are organized into groups.
- An Oracle database requires at least two groups.
- Each redo log within a group is called a member.
- All members of a group of logfiles contain the same information.
- The LGWR BG process concurrently writes the same information to all online redo log files in a group.
- Redo Logs are used in cyclic fashion.
- When a Redo Log file is full, LGWR will write to the next log group by performing a Log-Switch.

### 24.3 Redo Log File Organization – Multiplexing

Initially Redo Log Files are created when a database is created, preferably in groups to provide for multiplexing. Additional groups of files can be added as the need arises.

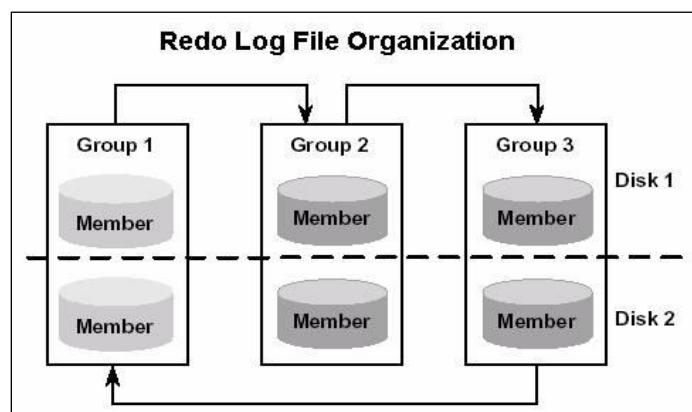


Figure 1-1. Redo Log File Organization

- Each Redo Log Group has identical Redo Log Files (however, each Group does not have to have the same # of Redo Log Files).
- If you have Redo Log Files in Groups, you must have at least two Groups. The Oracle Server needs a minimum of two online Redo Log Groups for normal database operation.
- The LGWR concurrently writes identical information to each Redo Log File in a Group.
- Thus, if Disk 1 crashes as shown in the figure above, none of the Redo Log Files are truly lost because there are duplicates.

Redo Log Files in a Group are called Members.

- Each Group Member has identical log sequence numbers and is the same size – they cannot be different sizes.
- The log sequence number is assigned by the Oracle Server as it writes to a log group and the current log sequence number is stored in the control files and in the header information of all Datafiles – this enables synchronization between Datafiles and Redo Log Files.
- If the group has more members, you need more disk drives in order for the use of multiplexed Redo Log Files to be effective.

Each database instance has its own set of Redo Log Files (Groups).

- Each group (whether or not the files are multiplexed) is called an instance redo thread.
- Typically one database instances access an Oracle database so the instance has one redo thread.
- When using Oracle Real application Clusters, more than one instance can access a single database so each instance will have a redo thread.

A Redo Log File stores Redo Log Records (or entries as they may be called).

- Each record consists of "vectors" that store information about:
- Changes made to a database block.
- Undo block data.
- Transaction table of undo segments.
- These enable the protection of rollback information as well as the ability to roll forward for recovery.
- Each time a Redo Log Record is written from the Redo Log Buffer to a Redo Log File, a System Change Number (SCN) is assigned to the committed transaction.

### 24.3.1 Adding Online Redo Log Groups

In some cases you might need to create additional log file groups. To create a new group of online redo log files, use the following SQL command:

```
SQL> ALTER DATABASE ADD logfile group 3
 ('/disk1/oradata/ORCL/redolog3a.log',
 '/disk2/oradata/ORCL/redolog3b.log') SIZE 4M;
```

We specify the name and location of the members with the file specification. The value of the GROUP parameter can be selected for each redo log file group. If you omit this parameter, the Oracle server generates its value automatically.

### 24.3.2 Dropping a Redo Log Group

To increase or decrease the size of online redo log groups, add new online redo log groups (with the new size) and then drop the old ones. An entire online redo log group can be dropped with the following command:

```
SQL> ALTER DATABASE ADD logfile group 3;
```

NOTE: An active or current group cannot be dropped.

To obtain the information of the Redo log groups query V\$LOG

```
SQL> SELECT group#, members, status, bytes, sequence# FROM V$LOG;
```

### 24.3.3 Adding Redo Log Members

You can add new members to existing redo log groups using the following Command:

```
SQL> ALTER DATABASE ADD LOGFILE MEMBER
 '/disk3/oradata/ORCL/redolog3C.log' TO GROUP 3;
```

### 24.3.4. Dropping a Redo Log Member

You may want to drop an online redo log member because it is invalid using command:

```
SQL> ALTER DATABASE DROP LOGFILE MEMBER
 '/disk3/oradata/ORCL/redolog3C.log' TO GROUP 3;
```

- Note:** 1. If the group is current, you must force a log file switch before you can drop the Member.  
 2. Also If the member you want to drop is the last valid member of the group, you cannot drop that member.

To obtain the information of the Redo log groups query V\$LOGFILE

```
SQL> SELECT group#, member, status FROM V$LOGFILE;
```

## 24.4 Redo Log File Usage

Redo Log Files are used in a circular fashion.

- One log file is written in sequential fashion until it is filled, and then the second redo log begins to fill. This is known as a Log Switch.
- When the last redo log is written, the database begins overwriting the first redo log again.

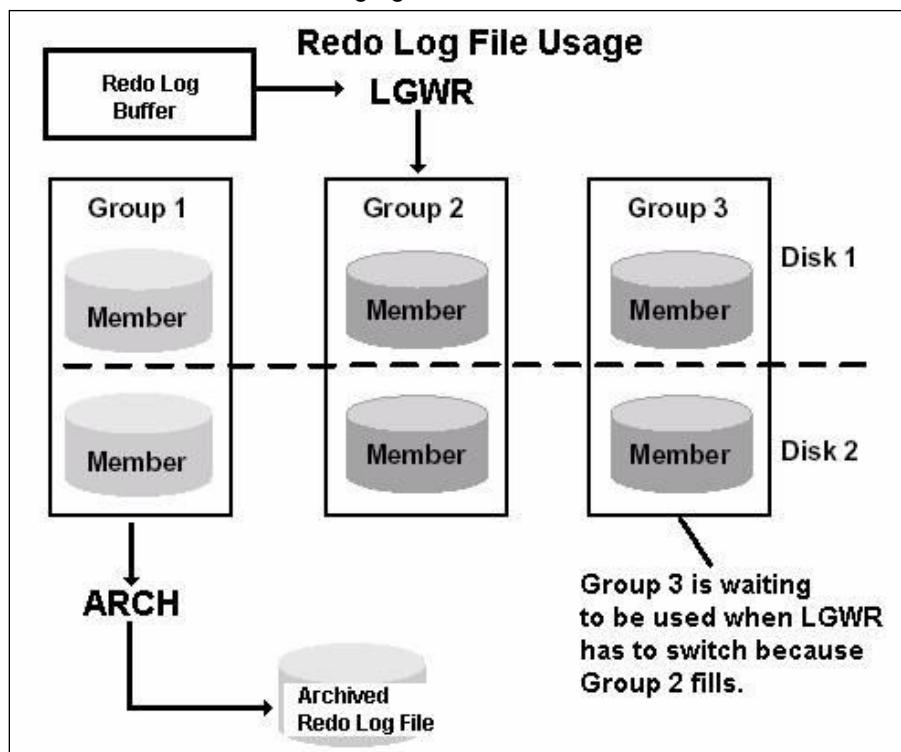


Figure 1-2. Redo Log File Usage

- The Redo Log file to which LGWR is actively writing is called the current log file.
- Log files required for instance recovery are categorized as active log files.
- Log files no longer needed for instance recoveries are categorized as inactive log files.
- Active log files cannot be overwritten by LGWR until ARC0 has archived the data when archiving is enabled.

## 25. Archivelog Files Management

The Oracle database can run in one of two modes:

- (a) NOARCHIVELOG (Default)      (b) ARCHIVELOG (Optional)

Difference between these two modes is simply what happens to a redo log file when Oracle goes to reuse it. "Will we keep a copy of that redo or should Oracle just overwrites it, losing it forever?" is an important question to answer. Unless you keep this file, you can't recover data from a backup to current point in time. Say you take a backup once a week on Saturday. Now, on Friday afternoon, after you have generated hundreds of redo logs over the week, your hard disk fails. If you haven't been running in ARCHIVELOG mode, the only choices you have now are as follows:

- Drop the tablespace(s) associated with the failed disk. Any tablespace that had a file on that disk must be dropped, including the contents of that tablespace. If the SYSTEM tablespace (Oracle's data dictionary) is affected, you cannot do this.
- Restore last Saturday's data and lose all of the work you did that week.

Neither option is very appealing. Both imply that you lose data. If you had been executing in ARCHIVELOG mode, on the other hand, you simply would have found another disk. You would have restored the affected files from Saturday's backup onto it. Lastly, you would have applied the archived redo logs and, ultimately, the online redo logs to them (in effect replaying the week's worth of transactions in fast-forward mode).

You lose nothing. The data is restored to the point in time of the failure. We believe that a system is not a production system unless it is in ARCHIVELOG mode. A database that is not in ARCHIVELOG mode will, some day, lose data. It is inevitable; you will lose data if your database is not in ARCHIVELOG mode. Only a test or development system should execute in NOARCHIVELOG mode.

Note: There are some cases in which a large DW could justify being in NOARCHIVELOG mode if it made judicious use of READ ONLY tablespaces and was willing to fully rebuild any READ WRITE tablespace that suffered a failure by reloading the data.

One of the important decisions that a database administrator has to make is whether the database is configured to operate in ARCHIVELOG mode or in NOARCHIVELOG mode.

### NOARCHIVELOG Mode

In NOARCHIVELOG mode, the online redo log files are overwritten each time an online redo log file is filled, and log switches occur. LGWR does not overwrite a redo log group until the checkpoint for that group is completed.

### ARCHIVELOG Mode

If the database is configured to run in ARCHIVELOG mode, active groups of filled online redo log files must be archived. This is done by the Oracle background process Archiver (ARCn). Because all changes made to the database are recorded in the online redo log files, the database administrator can use the physical backup and the archived online redo log files to recover the database without losing any committed data.

## 26. Exports & Imports

**Export** and **Import** are utilities that allow us to make exports and imports of **data objects** (such as tables). Exp/Imp allows transferring data across DB's that reside on different hardware platforms and/or on different Oracle versions.

If the data is exported on a system with a different Oracle version then on that on which it is imported, imp must be the newer version. That means, if something needs to be exported from 11g into 9i, it must be exported with 9i's exp.

Imp doesn't re-create an already existing table. It either errors out or ignores the errors. In order to use exp and imp, the `catexp.sql` script must be run. `catexp.sql` basically creates the `exp_full_database` and `imp_full_database` roles. It is found under `$ORACLE_HOME/rdbms/admin`:

```
SQL> @?/rdbms/admin/catexp.sql
catexp.sql is called by catalog.sql.
```

### 26.1 Import export modes

Exp/Imp can be used in four modes:

- **Full Export:** The `EXP_FULL_DATABASE` and `IMP_FULL_DATABASE`, respectively, are needed to perform a full export. Use the `full` export parameter for a full export.
- **Tablespace:** Use the `tablespaces` export parameter for a tablespace export.
- **User:** This mode can be used to export and import all objects that belong to a user. Use the `owner` export parameter and the `from user` import parameter for a user (owner) export-import.
- **Table:** Specific tables (and partitions) can be exported/imported with table export mode. Use the `tables` export parameter for a table export.

### 26.2 Export (exp)

Objects owned by SYS cannot be exported.

#### Prerequisites

- One must have the `create session privilege` for being able to use exp. If objects of another user's schema need to be exported, the `EXP_FULL_DATABASE` role is required.

#### Parameters

- **Full:** Use this parameter to specify full export mode.
- **Tablespaces:** Use this parameter to specify tablespace export mode.
- **Owner:** Use this parameter to specify user export mode.
- **Tables:** Use this parameter to specify table export mode.
- **Direct:** Used for a direct path export.
- **feedback=n:** Prints a dot after each  $n^{\text{th}}$  exported row.
- **flashback\_scn:** The exported data is consistent with the specified SCN.
- **flashback\_time:** The exported data is consistent with a SCN that approx. matches that of the specified time.
- **Consistent**
- **object\_consistent**
- **Query:** Restricts the exported rows by means of a `where clause`. The query parameter can only be used for table export mode. For obvious reasons, it must be applicable to all exported tables.
- **Parfile:** Specifies a parfile.

### 26.3 NLS\_LANG settings

As exp and imp are client utilities they use the NLS\_LANG settings.

## 26.4 Import (imp)

If the parameter `touser` is used and (?) the export was made with `FULL=YES`, the users must already be created in the target database.

### Parameters

- **Show:** This parameter only shows the contents of an export file; it does not perform an import.
- **Fromuser:** This parameter is used when an import in 'user export/import mode is made.

Primarily, there are 2 backup methods in Oracle.

1. Logical (using Exports - Only on Logical objects)
2. Physical

Though every night we do Physical backups, it's very important that we must perform logical backup also, which in a way serves as standby backup in case of our Physical backup failed (when tried to restore). Apart from acting as Secondary backup, Export is very important in some cases. For e.g. On Tuesday Night, we perform Physical & Logical backups and on Wed Morning at 10AM, one of the Users dropped a Table accidentally and he wants that Table real bad. Without disturbing other users, if he accepts the Table from our LN-LB, we can simply "Imp" that Table. So "Exp" is very important (some times equally or even more imp than our PB).

## 26.5 New Feature Oracle 11g R2 export and import

In oracle database 11g R2, the `DEFERRED_SEGMENT_CREATION` parameter is set TRUE by default. This means that any table which you create will not have any segment until the first row of data is inserted into the table. Original export ignores tables without segments. Therefore, if we create any new tables and do not insert any data before performing on export, those tables will not be exported.

Example:

```
$ EXP FILE=TEST.DMP LOG=TEST.LOG OWNER=TEST
```

**Username:** username/password

## 26.6 Reasons why we perform Logical Backup:

1. Whenever we are migrating from One OS to another OS. (E.g. say our NT server has become slow, since no. of users increased etc. and we are planning to shift to Solaris. Then the only way we can get the database is by Exporting from NT and Importing into Solaris). Here we should observe that Oracle Logical Backup is, platform independent. Wherever we have Oracle, we can take this "Exported Dump-File", and we can Import the data. (Oracle supports 80 Platforms)
2. Whenever we do Oracle Version Upgrades. E.g. say we are upgrading our Oracle from 10g to 11g. In this case, we would like our Oracle to not only upgrade the Server-Software but-also upgrade the Database. (We cannot open a database created under 10g using 11g, since 11g Server requires its own set of BT, which are completely different from our 10g BT). So in this case, we would like our Oracle to do the Database upgradation also. But, if it fails, we end-up having a down-DB. To avoid such unwanted scenario, before even upgrading our Server we go-ahead and perform Oracle "Exp" on 10g Database.

Then upgrade the Server to 11g including DB-Upgrade. If it is successful well & good. Otherwise, we can create an empty Database (with similar TS structures) and then start importing our "DMP" file. So all logical structure will now be created in 11g Database.

3. Whenever Segments are having too-many extents (more than 20), we have to perform DB-REORG. Usually we do this REORG every 3-4 months. To do REORG, PB can't help us. Since by restoring PB, again we are back to same number of extents, since PB is

only copying the Datafiles. We are interested in Exports since what EXP does, it "calculates" how-many extents each segment has and how much space is required etc. So in its Exp-Definition, the Segment's properties are changed ("Initial") compared to our original Segment's properties. At the same time, by doing frequent REORGs, we can also be sure of the Data-Integrity (finding bad-blocks etc).

We do exports whenever we want to take objects from one database to another or from one user to another.

If the DB is too big, we can also perform "Incremental" exports on weekdays, and complete on Weekends.

Exports can be done at 3 Levels.

1. Table-Level
2. User-Level
3. DB-Level (We need DBA Privilege)

When we are doing at Table-Level, we can mention either one or More Tables according our need. Similarly, same rule applies to (B) also. But, we can't be choosy at different Levels with different options. For E.g. If we want Scott's Emp, Dept & Steve's Table\_A, Table\_B this can't be mentioned. When we select at User-Level, complete Schema will be exported. But this is allowed:

```
tables=scott.emp,scott.dept,steve.tab1,steve.tab2
```

1. To see help in Exp or Imp, simply say

```
$ exp help=y
```

This will show us all the different parameters with small descriptions and with their default values.

2. Export is basically: Table-Def + Data + Const-Def (Indexes) + Index-Def + Views-Def + Synonyms-Def + PL\*SQL-Proc + Packages + DB-Links + Sequences etc. So as we can see here, except the Table-Data, everything else is "DEF-Only". This is the reason why size of the Exp is lot less (probably 8-10 times) compared to DB-Size. And this DMP file (which came from our EXP) can further be "compressed" using UNIX "compress" utility (still 10times). So ultimately we can backup an 11gB database in about 120-Mb and make very portable.
3. When we do DB-Level Export, it contains def. for all TSs. Users and every thing. But the reasons why Exp file is so less in size are:
4. It is carrying DEF only. For e.g. On a Table we may have 8 Indexes. In this case more than Table, Indexes are occupying more space in the Database. But, at the time of Exp, we only worry about INDEX-DEF.
5. SYS Tables (Base-Tables) are not getting exported. The reason is, these Base tables are belonging to this Database. We don't have to take these to the Target-DB, since that Database has its own set of BT.
6. TEMP - Def is only exported but not the content.
7. REDOs & Controlfile are not getting exported. So all these are reasons why our Exp file is so small compared to our Physical Database though we exported FULL Database.
8. In Oracle, there is no utility to give us the Complete-Schema Definition. But we can some-how get this out using Exp & Imp. What we do:

```
$ exp scott/tiger file=scott.dmp log=scott.log
$ imp scott/tiger file=scott.dmp show=y log=scott_schema.sql
```

Once the second command is done, we should have a file "scott\_schema.sql". By editing this file, we can get the complete-schema for SCOTT.

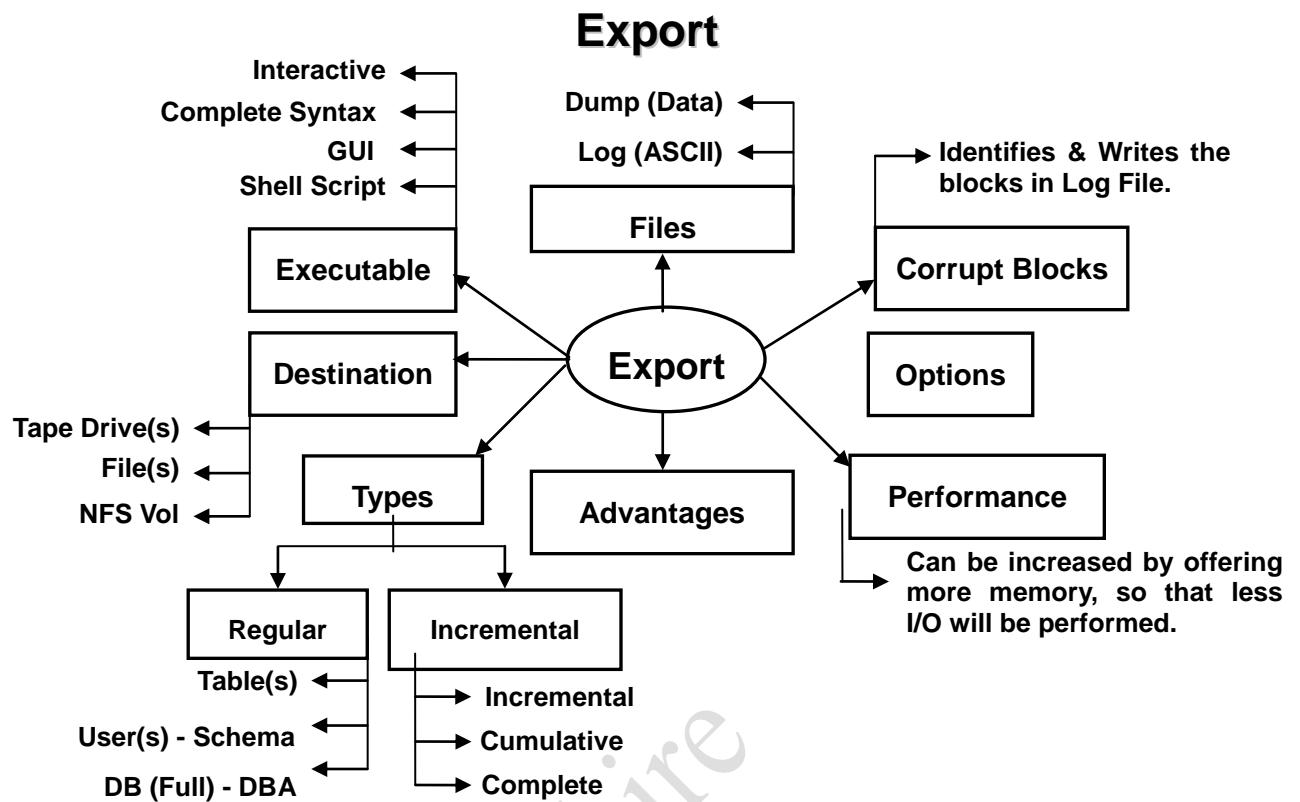
1. Exp must be done while Database is up & running (same with Import).
2. While Exp is going on users may access & modify the Database. But when this happens, Exp is not guaranteeing a time-image. This is the reason we do Exp in the night before doing the Physical backup.
3. Always mention the file names --> DMP-File & LOG-File names. Otherwise Oracle will use default file name for DMP, which is "expdat.dmp". By default Log file is not produced unless we mention it.
4. By def. Oracle only offers 256K of buffer size which is very less when exporting very big tables. So supply our own buffer-size as a parameter.
5. In case there are so many tables to be exported, instead of mentioning all their names at the command level, we can create a "parameter file" in which we can list all the tables etc.
6. If we want to export some tables from SCOTT and import those into STEVE then we can do it from a DBA account.  
\$ exp system/manager table=scott.emp file=scott\_emp.dmp \  
log=scott\_emp.log  
\$ imp system/manager fromuser=scott touser=steve file=scott\_emp.dmp

Exp can be done in 3 diff. ways.

1. Command Line
2. Interactive Method
3. GUI Based (May be on NT)

If exp and imp are used to export data from an Oracle database with a different version than the database in which is imported, then the following rules apply:

1. exp must be of the lower version
2. imp must match the target version.



## 27. Data Pump

The data pump comes new with Oracle 10g. Traditional export and import ('exp' and 'imp') have not gone away: they are still there in 10g. But they have been augmented (superseded, perhaps) by EXPDP and IMPDP. It provides high speed, parallel, bulk data and metadata movement of Oracle database contents across platforms and database versions. If a data pump job is started and fails for any reason before it has finished, it can be restarted at a later time. The commands to start the data pump are **expdp** and **impdp**, respectively.

### 27.1 Characteristics of Data Pump

- We can disconnect from a DP job, leaving it running in the background while we can get on with other work. Compare that to EXP or IMP, where our terminal/session was tied up until export/import job had completed.
- We can monitor a Data Pump job from within the database itself. There are data dictionary views to show us what DP jobs are running, what their names are, and how far along they've gotten. With traditional EXP/IMP, we could only tell the progress of a job by looking at the terminal/session it had taken over, and hoping that the screen displayed some activity.
- DP Jobs are Resumable. They keep tabs on where they've gotten to by populating a special table in the export/import data set, and thus if the job is abnormally terminated, it can resume from where it left off by reading the contents of this table.
- DP Jobs are Parallelisable. There is no way of making EXP/IMP parallelize its operations, except by starting lots of different export jobs, each with a different list of things to export, but DP definitely does, and easily.
- Data Pump has an application programming interface (API), which is a posh way of saying it works because there are a bunch of new packages and procedures in the database (DBMS\_DATAPUMP, for example) to control it, and that in turn means that it is possible to write our own code to interface with it. Hence, programmatic control of DP with our own customized interfaces becomes possible.
- Data Pump does its work entirely and only on the server itself. It can be invoked from a client, of course, but the processes will run, and the outputs will only ever be produced, on the server.
- We can ask an Export DP job to estimate how big the output file would be, without actually running it for real.
- Potentially one of the most useful features is that it is now possible to import data into one database directly from another one over a database link (that is, over the network). Doing that in 9i or earlier, we would have exported out of one and generated a dump file; then we would have moved the dump file between the two servers; then we would have run import on the other database. With Data Pump running in Network Mode, we simply extract the data directly over the database link: there are no intermediate files.

Every Data Pump job causes a table to be created in the schema of the user who is performing the job and with a name that is the same as the Data Pump job name (which can be system-generated if we don't supply one). This table is known generically as the 'Master Table' or MT for short (and in the Oracle documentation).

The Master Table is used to keep track of where a particular Data Pump job has gotten to, and is thus the mechanism, which makes Resumable Data Pump jobs possible. When the job finishes, the table is dropped. If the job aborts abnormally, the table persists and may need to be manually dropped (assuming we don't want to restart the job, of course).

If we already own a real table which happens to have the same name as the name we give to a Data Pump job, then that job will not even start, because Oracle won't be able to create the Master Table for the proposed job due to the potential duplicate table names.

Because Data Pump runs entirely on the server, it will output to a DIRECTORY object (which obviously needs to be created first). It can also output to a default directory object (which still needs to be created first, of course), specified with an environment variable DATA\_PUMP\_DIR.

```
SQL> CREATE DIRECTORY dpump AS '/disk1/oradata/ORCL/dump';
SQL> GRANT READ, WRITE ON DIRECTORY dpump TO Scott, steve;
$ cd /disk1/oradata/ORCL
$ mkdir dpump
```

We can obtain a list of all the parameters that can be specified for a Data Pump job by doing this:

```
$ expdp help=y
$ impdp help=y
```

**We can control how Export runs by entering expdp command followed by various parameters.**

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| ATTACH        | Attach an existing job                                                    |
| DIRECTORY     | Directory to be used for dump files and log files.                        |
| DUMPFILE      | List of dumpfile(s)                                                       |
| LOGFILE       | Log File name.                                                            |
| ESTIMATE_ONLY | Calculate job estimates without performing the Export.                    |
| FULL          | Export entire database.                                                   |
| INCLUDE       | Include specific object types (like Constraints, Triggers, Grants, etc.,) |
| JOB_NAME      | Name of the export job to create                                          |
| PARALLEL      | No. of active workers for current job.                                    |
| PARFILE       | Specify parameter file.                                                   |
| SCHEMAS       | List of Schemas to Export                                                 |
| TABLES        | List of Tables to Export                                                  |
| TABLESPACES   | List of Tablespaces to Export                                             |
| QUERY         | Clause to be used when a subset of records to be exported                 |

**We can control how import runs by entering 'impdp' command followed by various parameters:**

|               |                                                         |
|---------------|---------------------------------------------------------|
| ATTACH        | Attach an existing job                                  |
| CONTENT       | Specifies data to load (ALL, DATA_ONLY, METADATA_ONLY)  |
| DIRECTORY     | Directory to be used for dump files and log files.      |
| DUMPFILE      | List of dumpfile(s)                                     |
| LOGFILE       | Log File name.                                          |
| ESTIMATE_ONLY | Calculate job estimates without performing the Import.  |
| EXCLUDE       | Exclude Specific objects e.g., EXCLUDE=TABLE:EMP        |
| JOB_NAME      | Name of the export job to create                        |
| PARALLEL      | No. of active workers for current job.                  |
| PARFILE       | Specify parameter file.                                 |
| REMAP_SCHEMA  | Objects from one schema are loaded into another schema. |
| SCHEMAS       | List of Schemas to Import                               |
| SQLFILE       | Write all the SQL DDL to a file.                        |
| TABLES        | List of Tables to Import                                |
| TABLESPACES   | List of Tablespaces to Import                           |

**The Following commands can be used in interactive mode:**

|                 |                                                |
|-----------------|------------------------------------------------|
| ADD_FILE        | Add dump file to the dump file set.            |
| CONTINUE_CLIENT | Return to logging mode                         |
| EXIT_CLIENT     | Quit client session and leave the job running. |
| KILL_JOB        | Detach & delete job.                           |
| START_JOB       | Start/Resume current job.                      |
| STOP_JOB        | Stop the current job.                          |
| STATUS          | Display the current job status.                |

## 27.2 Performing a Full Database Export

```
$ expdp system/manager directory=dpump dumpfile=full.dmp
 logfile=full.log parallel=3 job_name=full_exp
```

A table will be created in the SYSTEM's schema with the job name i.e., FULL\_EXP which keeps track of the Datapump export job and this table will be dropped automatically once the job is completed successfully. Now if we wish to stop the above job:

Press Ctrl+C

```
Export > Stop_job
Export > exit
$ expdp system/manager ATTACH=full_exp
Export > START_JOB
```

## 27.3 Performing a Schema-Mode export

By the Schema owner:

```
$ expdp scott/tiger directory=dpump dumpfile=scott.dmp
 logfile=scott.log job_name=scott_job
```

By the DBA:

```
$ expdp system/manager directory=dpump dumpfile=scott.dmp
 logfile=scott.log job_name=scott_job Schemas=scott
```

## 27.4 Importing objects of one Schema into other

```
$ impdp system/manager directory=dpump dumpfile=scott.dmp
 logfile=steve.log job_name=steve_job remap_schema=steve
```

## 27.5 Performing a Table-Mode export

```
$ expdp scott/tiger directory=dpump dumpfile=tables.dmp
 logfile=tables.log job_name=tables_job tables=emp.dept
```

## 27.6 Estimating the Disk space needed

```
$ expdp system/manager directory=dpump estimate_only=Y logfile=full.log
```

The Estimate is printed in the logfile and displayed on the standard output.

## 27.7 Monitoring DataPump Operations

Oracle 11g provides two new views, `DBA_DATAPUMP_JOBS` and `DBA_DATAPUMP_SESSIONS` that allow the DBA to monitor the progress of all DataPump operations. Example shown below has two queries that return valuable information about ongoing DataPump jobs and sessions.

Querying status of currently active DataPump operations

```
SQL> SELECT owner_name,job_name,operation,job_mode,
 state,degree,attached_sessions
 FROM dba_datapump_jobs
```

Querying status of currently active DataPump Sessions

```
SQL> SELECT DPS.owner_name,DPS.job_name,S.osuser
 FROM dba_datapump_sessions DPS,v$session S
 WHERE S.saddr = DPS.saddr
```

## 27.8 Data Pump Legacy Mode

If you use original Export (exp) and Import (imp), then you may have scripts you have been using for many years. To ease the transition to the newer Data Pump Export and Import utilities, Data Pump provides a legacy mode which allows you to continue to use your existing scripts with Data Pump.

Data Pump enters legacy mode once it determines a parameter unique to original Export or Import is present, either on the command line or in a script. As Data Pump processes the parameter, the analogous Data Pump Export or Data Pump Import parameter is displayed. Oracle strongly recommends that you view the new syntax and make script changes as time permits.

Note: Data Pump Export and Import only handle dump files and log files in the Data Pump format. They never create or read dump files compatible with original Export or Import. If you have a dump file created with original Export, then you must use original Import to import the data into the database.

### 27.8.1 Management of File Locations in Data Pump Legacy Mode

Original Export and Import and Data Pump Export and Import differ on where dump files and log files can be written to and read from because the original version is client-based and Data Pump is server-based.

Original Export and Import use the FILE and LOG parameters to specify dump file and log file names, respectively. These file names always refer to files local to the client system and they may also contain a path specification.

Data Pump Export and Import use the DUMPFILE and LOGFILE parameters to specify dump file and log file names, respectively. These file names always refer to files local to the server system and cannot contain any path information. Instead, a directory object is used to indirectly specify path information. The path value defined by the directory object must be accessible to the server. The directory object is specified for a Data Pump job through the DIRECTORY parameter.

It is also possible to prepend a directory object to the file names passed to the DUMPFILE and LOGFILE parameters. For privileged users, Data Pump supports the use of a default directory object if one is not specified on the command line. This default directory object, DATA\_PUMP\_DIR, is set up at installation time.

If Data Pump legacy mode is enabled and the original Export FILE=filespec parameter and/or LOG=filespec parameter are present on the command line, then the following rules of precedence are used to determine a file's location:

Note: If the FILE parameter and LOG parameter are both present on the command line, then the rules of precedence are applied separately to each parameter.

Also, when a mix of original Export/Import and Data Pump Export/Import parameters are used, separate rules apply to them. For example, suppose you have the following command:

```
SQL> expdp system FILE=/user/disk/foo.dmp LOGFILE=foo.log
DIRECTORY=dpump_dir
```

The Data Pump legacy mode file management rules, as explained in this section, would apply to the FILE parameter. The normal (that is, non-legacy mode) Data Pump file management rules, as described in "Default Locations for Dump, Log, and SQL Files", would apply to the LOGFILE parameter.

1. If a path location is specified as part of the file specification, then Data Pump attempts to look for a directory object accessible to the schema executing the export job whose path location matches the path location of the file specification. If such a directory object cannot be found, then an error is returned. For example, assume that a server-based directory object named USER\_DUMP\_FILES has been defined with a path value of '/disk1/user1/dumpfiles/' and that read and write access to this directory object has been granted to the hr schema. The following command causes Data Pump to look for a server-based directory object whose path value contains '/disk1/user1/dumpfiles/' and to which the hr schema has been granted read and write access:
2. `expdp hr FILE=/disk1/user1/dumpfiles/hrdata.dmp`

In this case, Data Pump uses the directory object USER\_DUMP\_FILES. The path value, in this example '/disk1/user1/dumpfiles/', must refer to a path on the server system that is accessible to the Oracle Database.

If a path location is specified as part of the file specification, then any directory object provided using the DIRECTORY parameter is ignored. For example, if the following command is issued, then Data Pump does not use the DPUMP\_DIR directory object for the file parameter, but instead looks for a server-based directory object whose path value contains '/disk1/user1/dumpfiles/' and to which the hr schema has been granted read and write access:

```
expdp hr FILE=/disk1/user1/dumpfiles/hrdata.dmp DIRECTORY=dpump_dir
```

3. If no path location is specified as part of the file specification, then the directory object named by the DIRECTORY parameter is used. For example, if the following command is issued, then Data Pump applies the path location defined for the DPUMP\_DIR directory object to the hrdata.dmp file:
4. `expdp hr FILE=hrdata.dmp DIRECTORY=dpump_dir`
5. If no path location is specified as part of the file specification and no directory object is named by the DIRECTORY parameter, then Data Pump does the following, in the order shown:
  1. Data Pump looks for the existence of a directory object of the form DATA\_PUMP\_DIR\_schema\_name, where schema\_name is the schema that is executing the Data Pump job. For example, the following command would cause Data Pump to look for the existence of a server-based directory object named DATA\_PUMP\_DIR\_HR:
  2. `expdp hr FILE=hrdata.dmp`

The hr schema also must have been granted read and write access to this directory object. If such a directory object does not exist, then the process moves to step b.

3. Data Pump looks for the existence of the client-based environment variable DATA\_PUMP\_DIR. For instance, assume that a server-based directory object named DUMP\_FILES1 has been defined and the hr schema has been granted read and write access to it. Then on the client system, the environment variable DATA\_PUMP\_DIR can be set to point to DUMP\_FILES1 as follows:
4. `setenv DATA_PUMP_DIR DUMP_FILES1`
5. `expdp hr FILE=hrdata.dmp`

Data Pump then uses the server-based directory object DUMP\_FILES1 for the hrdata.dmp file.

If a client-based environment variable DATA\_PUMP\_DIR does not exist, then the process moves to step c.

6. If the schema that is executing the Data Pump job has DBA privileges, then the default Data Pump directory object, DATA\_PUMP\_DIR, is used. This default directory object is established at installation time. For example, the following command causes Data Pump to attempt to use the default DATA\_PUMP\_DIR directory object, assuming that system has DBA privileges:
7. `expdp system FILE=hrdata.dmp`

### 27.8.2 Multiple Schema object Exported

The restriction that in table mode all tables had to reside in the same schema has been removed we can now specify the tables in multiple schemas.

Examples

```
$expdp dumpfile=multisch.dmp logfile=multisch.log tables=js.emp,
test.dept
```

### 27.8.3 Execute Privilege for Directory Object

EXECUTE privilege is allowed for DIRECTORY objects in this release. The ORACLE\_LOADER access driver creates a process that runs a user-specified program. That program must live in a directory path specified by a directory object defined in the database. Only a user that has been given EXECUTE access to the directory object is allowed to run programs in it.

This feature allows the DBA to control who is allowed to run preprocessors as part of loading data with external tables. It also allows the DBA to restrict which programs those users can run. No existing users with access to the directory object are allowed to run any programs from that directory unless the DBA gives them EXECUTE access to that directory.

Examples

Create a directory to store the dump files.

```
SQL>CREATE DIRECTORY dump_dir1 AS '/usr/apps/datafiles';
SQL>GRANT EXECUTE ON DIRECTORY dump_dir1 to TEST,SCOTT;
$ expdp file=test.dmp log=test.log owner=test directory=dump_dir1
```

## 27.9 Data Pump Enhancements in 12c

A multitenant container database (CDB) is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs). A PDB is a portable set of schemas, schema objects, and non-schema objects that appear to an Oracle Net client as a non-CDB. A non-CDB is an Oracle database that is not a CDB.

You can use Data Pump to migrate all, or portions of, a database from a non-CDB into a PDB, between PDBs within the same or different CDBs, and from a PDB into a non-CDB. In general, using Data Pump with PDBs is identical to using Data Pump with a non-CDB.

**Note:** In Oracle Database 12c Data Pump does not support any CDB-wide operations.

Data pump issues the following warning if you are connected to the root or seed database of a CDB.

ORA-39357: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database.

If you want to specify a particular PDB for the export/import operation, then on the Data Pump command line, you can supply a connect identifier in the connect string when you start Data Pump.

The following requirements when using Data Pump to move data into a CDB:

To administer a multitenant environment, you must have the CDB\_DBA role.

Full database exports from Oracle Database 11.2.0.2 and earlier may be imported into Oracle Database 12c (CDB or non-CDB). However, Oracle recommends the source database first be upgraded to Oracle Database 11g release 2 (11.2.0.3 or later) so that information about registered options and components is included in the export.

When migrating Oracle Database 11g release 2 (11.2.0.3 or later) to a CDB (or to a non-CDB) using either full database export or full transportable database export, you must set the Data Pump Export parameter `VERSION=12` in order to generate a dump file that is ready for import into Oracle Database 12c. If you do not set `VERSION=12`, then the export file that is generated will not contain complete information about registered database options and components.

Network-based full transportable imports require use of the `FULL=YES`, `TRANSPORTABLE=ALWAYS`, and `TRANSPORT_DATAFILES=`*datafile\_name* parameters. When the source database is Oracle Database 11g release 11.2.0.3 or later, but earlier than Oracle Database 12c Release 1 (15.1), the `VERSION=12` parameter is also required.

File-based full transportable imports only require use of the `TRANSPORT_DATAFILES=`*datafile\_name* parameter. Data Pump Import infers the presence of the `TRANSPORTABLE=ALWAYS` and `FULL=YES` parameters.

The default Data Pump directory object, `DATA_PUMP_DIR`, does not work with PDBs. You must define an explicit directory object within the PDB that you are exporting or importing.

### 27.15.1 NOLOGGING Option (DISABLE\_ARCHIVE\_LOGGING)

The `TRANSFORM` parameter of `impdp` has been extended to include a `DISABLE_ARCHIVE_LOGGING` option. The default setting of 'N' has no effect on logging behaviour.

Using a value 'Y' reduces logging associated with tables and indexes during the import by setting their logging attribute to `NOLOGGING` before the data is imported and resetting it to `LOGGING` once the operation is complete.

`TRANSFORM=DISABLE_ARCHIVE_LOGGING=Y`

The effect can be limited to a specific type of object (Table or Index) by appending the object type.

`TRANSFORM=DISABLE_ARCHIVE_LOGGING=Y:TABLE`

`TRANSFORM=DISABLE_ARCHIVE_LOGGING=Y:INDEX`

### 27.11.2 LOGTIME Parameter

The `LOGTIME` parameter determines if timestamps should be included in the output messages from the `expdp` and `impdp` utilities.

`LOGTIME=[NONE | STATUS | LOGFILE | ALL]`

The allowable values are explained below.

**NONE:** The default value, which indicates that no timestamps should be included in the output, making the output look similar to that of previous versions.

**STATUS:** Timestamps are included in output to the console, but not in the associated log file.

**LOGFILE:** Timestamps are included in output to the log file, but not in the associated console messages.

**ALL:** Timestamps are included in output to the log file and console.

### 27.15.3 Export View as Table

The **VIEWS\_AS\_TABLES** parameter allows Data Pump to export the specified views as if they were tables. The table structure matches the view columns, with the data being the rows returned by the query supporting the views.

By default `expdp` creates a temporary table as a copy of the view, but with no data, to provide a source of the metadata for the export. Alternatively you can specify a table with the appropriate structure.

### 27.15.4 Change Table Compression at Import

The **TABLE\_COMPRESSION\_CLAUSE** clause of the **TRANSFORM** parameter allows the table compression characteristics of the tables in an import to be altered on the fly.

The allowable values for the **TABLE\_COMPRESSION\_CLAUSE** include the following.

**NONE**: The table compression clause is omitted, so the table takes on the compression characteristics of the tablespace.

**NOCOMPRESS**: Disables table compression.

**COMPRESS**: Enables basic table compression.

**ROW STORE COMPRESS BASIC**: Same as COMPRESS.

**ROW STORE COMPRESS BASIC**: Same as COMPRESS.

**ROW STORE COMPRESS ADVANCED**: Enables advanced compression, also known as OLTP compression.

**COLUMN STORE COMPRESS FOR QUERY**: Hybrid Columnar Compression (HCC) available in Exadata and ZFS storage appliances.

**COLUMN STORE COMPRESS FOR ARCHIVE**: Hybrid Columnar Compression (HCC) available in Exadata and ZFS storage appliances.

### 27.15.5 Transportable Database

The **TRANSPORTABLE** option can now be combined with the **FULL** option to transport a whole database. The transport database option is supported for database version higher than 11.2.0.2.

This method can also be used to upgrade the database. Below syntax is provided for reference.

## 28. Transportable Tablespaces

### 28.1 Overview

Transportable tablespaces were introduced in Oracle8i to allow whole tablespaces to be copied between databases in the time it takes to copy the datafiles. In Oracle8i one of the restrictions was that the block size of both databases must be the same. In Oracle9i the introduction of multiple block sizes has removed this restriction.

Transportable tablespaces are the fastest way for moving large volumes of data between two Oracle databases. You can transport tablespaces between different computer architectures and operating systems. We can use the Transportable Tablespaces feature to copy a set of tablespaces from one Oracle Database to another. The tablespaces being transported can be either dictionary managed or locally managed.

Starting with Oracle9i, the transported tablespaces are not required to be of the same block size as the target database standard block size. Moving data using transportable tablespaces is much faster than performing either an export/import or unload/load of the same data. This is because the datafiles containing all of the actual data are just copied to the destination location, and you use an export/import utility to transfer only the metadata of the tablespace objects to the new database.

### 28.2 Limitations on Transportable Tablespace Use

- The source and target database must use the same character set and national character set.
- We can't transport a tablespace to a target DB in which a tablespace with the same name already exists.
- Objects with underlying objects (such as materialized views) or contained objects (such as partitioned tables) are not transportable unless all of the underlying or contained objects are in the tablespace set.
- The Tablespace should not have any SYS owned objects.
- The Tablespace should not have any Online Rollback Segments.

### 28.3 Transporting Tablespaces Across Platforms

Starting with Oracle 10g DB, you can transport tablespaces across platforms (like from Windows to Linux). Many, but not all, platforms are supported for cross-platform tablespace transport. We can query the V\$TRANSPORTABLE\_PLATFORM view to see platforms that are supported, and to determine each platform's endian format (byte ordering). The following query displays the platforms that support cross-platform tablespace transport:

```
SQL> SELECT * FROM V$TRANSPORTABLE_PLATFORM;
Steps for Transporting Tablespace across platforms using Data Pump
```

Let us consider a case where we are required to Transport the tablespace USER\_DATA from a database on Linux to a database on Windows Platform. For a tablespace to be transportable it must be totally autonomous. This can be checked using the DBMS\_TTS.TRANSPORT\_SET\_CHECK procedure. The TS\_LIST parameter accepts a comma separated list of tablespace names and the INCL\_CONSTRAINTS parameter indicates if constraints should be included in the check:

Execute the below procedure for check the tablespace constraints

```
SQL> EXEC
DBMS_TTS.TRANSPORT_SET_CHECK(ts_list=>'USER_DATA',incl_constraints =>
TRUE);
The TRANSPORT_SET_VIOLATIONS view can be used to check for any
violations:
```

```
SQL> SELECT * FROM transport_setViolations;
Make the tablespace READ ONLY
SQL> ALTER TABLESPACE user_data READ ONLY;
Export the tablespace. From the OS prompt, issue:
$ EXPDP dumpfile=userdata_tts.dmp logfile=userdata_tts.log
 directory=dump_dir1 transport tablespaces=user_data
```

Use operating system utilities to copy the datafiles of USER\_DATA tablespace to the target server.

From the OS command prompt, you would issue the following command to import on Windows

```
C:\> IMPDP directory=dump_dir dumpfile=userdata_tts.dmp
 transport_datafiles='c:\oracle\oradata\ORCL\userdata01.dbf'
```

Make the new tablespaces read write in target database, SQL> ALTER TABLESPACE user\_data READ WRITE;

## 29. Tablespace Transport for a Single Partition

In earlier versions of Oracle Database, you gained the ability to transport a tablespace and later plug it into a different database or to the same one. The process involves the copying of datafiles so it is the fastest way to transfer data across databases. However, until now, you didn't have the ability to transport the tablespace of a single partition and then plug it back. In Oracle Database 11g, you can. Suppose you have a table called SALES5, with several partitions named CT, NY, etc.

```
SQL> select partition_name, tablespace_name
 2 from user_tab_partitions
 3 where table_name = 'SALES5';

PARTITION_NAME TABLESPACE_NAME
----- -----
CT TS1
NY TS2
```

Now, you can transport the partition CT using the command shown below:

```
$ expdp tables=scott.sales5:ct transportable=always
directory=data_pump_dir dumpfile=p_ct.dmp
```

```
Export: Release 15.1.0.4.0 - Beta on Sunday, 10 June, 2007 16:05:40 Copyright
(c) 2003, 2005, Oracle. All rights reserved. Username: / as sysdba
Connected to: Oracle Database 11g Enterprise Edition Release 15.1.0.4.0 - Beta
With the Partitioning, Oracle Label Security, OLAP, Data Mining
and Oracle Database Vault options
Starting "SYS"."SYS_EXPORT_TABLE_01": /***** AS SYSDBA
tables=scott.sales5:ct transportable=
 always directory=data_pump_dir dumpfile=p_ct.dmp
Processing object type TABLE_EXPORT/TABLE/PLUGTS_BLK
Processing object type TABLE_EXPORT/TABLE
Processing object type TABLE_EXPORT/TABLE/END_PLUGTS_BLK
Master table "SYS"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded

Dump file set for SYS.SYS_EXPORT_TABLE_01 is:
 /home/oracle/oracle/admin/PROBE2/dpdump/p_ct.dmp

Datafiles required for transportable tablespace TS1:
 /home/oracle/oradata/PROBE2/PROBE2/ts1_01.dbf
Job "SYS"."SYS_EXPORT_TABLE_01" successfully completed at 16:05:55
```

Now, you can take these two files p\_ct.dmp and ts1\_01.dmp to another system and try to plug into the database. For learning purposes, let's try to plug into the same database. First, you need to drop the table and then the tablespace ts1.

```
SQL> drop table scott.sales5;
Table dropped.

SQL> drop tablespace ts1 including contents;
Tablespace dropped.
```

Now, plug the tablespace into the database. But here's a little problem: the table sales5 no longer exists and you had initially exported only one partition (ct), not the entire table. So how can you import just one partition of a non-existent table?

In Oracle Database 11g, a new command line option in Data Pump Import called partition\_options makes that possible. If you specify the value `departition`, Data Pump will create a new table from the partitions exported. In a way this approach "breaks" partitions, so it's appropriately named `departition`. Let's see how it works.

```

$ impdp partition_options=departition dumpfile=p_ct.dmp
 transport_datafiles='/home/oracle/oradata/PROBE2/PROBE2
 /ts1_01.dbf'

Import: Release 15.1.0.4.0 - Beta on Sunday, 10 June, 2007 21:58:08
Copyright (c) 2003, 2005, Oracle. All rights reserved.
Username: / as sysdba
Connected to: Oracle Database 11g Enterprise Edition Release 15.1.0.4.0
- Beta
With the Partitioning, Oracle Label Security, OLAP, Data Mining
and Oracle Database Vault options
Master table "SYS"."SYS_IMPORT_TRANSPORTABLE_04" successfully
loaded/unloaded
Starting "SYS"."SYS_IMPORT_TRANSPORTABLE_04": /***** AS SYSDBA
partition_options=departition dumpfile=p_ct.dmp
transport_datafiles=/home/oracle/oradata/PROBE2/PROBE2/ts1_01.dbf
Processing object type TABLE_EXPORT/TABLE/PLUGTS_BLK
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/END_PLUGTS_BLK
Job "SYS"."SYS_IMPORT_TRANSPORTABLE_04" successfully completed at
21:58:23

```

This SQL creates a table called sales5\_ct, which is nothing but the ct partition of SALES5 table exported by transportable tablespace earlier. The table name, as you can see, is a combination of the original table and the partition names. You can confirm the presence of the segment by checking the DBA\_SEGMENTS view.

```

SQL> select segment_name
 2 from dba_segments
 3 where tablespace_name = 'TS1';

SEGMENT_NAME

SALES5_CT

```

You can use single-partition transportable tablespace feature to plug in a single partition of a table to a different database. After plugging it in, you may want to perform an exchange partition operation to put that as a partition on some table there.

## 30. Backups & Recovery Concepts

### 30.1 Basics of Backup and Recovery

A backup is a representative copy of data. This copy can include important parts of a database such as the control file, redo logs, and datafiles. A backup protects data from application error and acts as a safeguard against unexpected data loss, by providing a way to restore original data. Backups are divided into two types.

1. Physical Backups
2. Logical Backups

Physical backups are copies of physical database files. The phrase "backup and recovery" usually refers to the transfer of copied files from one location to another, along with the various operations performed on these files. In contrast, logical backups contain data that is exported using SQL commands and stored in a binary file.

Oracle records both committed and uncommitted changes in redo log buffers. Logical backups are used to supplement physical backups. Restoring a physical backup means reconstructing it and making it available to the Oracle server. To recover a restored backup, data is updated using redo records from the transaction log. The transaction log records changes made to the database after the backup was taken.

Oracle performs crash recovery and instance recovery automatically after an instance failure. In the case of media failure, a database administrator (DBA) must initiate a recovery operation. Recovering a backup involves two distinct operations: rolling the backup forward to a more recent time by applying redo data and rolling back all changes made in uncommitted transactions to their original state.

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

### 30.2 Backup and Recovery Operations

A backup is a snapshot of a datafile, tablespace, or database at a certain time. If periodic backups of the database have been made and data is lost, users can apply the stored redo information to their latest backup to make the database current again.

Oracle enables users to restore an older backup and apply only some redo data, thereby recovering the database to an earlier point in time. This type of recovery is called incomplete media recovery. If the backup was consistent, then users are not required to apply any redo data, at all.

A simple example of media recovery illustrates the concept. Suppose a user makes a backup of the database at noon. Starting at noon, one change to the database is made every minute. At 1 p.m. one of the disk drives fails, causing the loss of all data on that disk.

Fortunately, Oracle records all changes in the redo log. The user can then restore the noon backup onto a functional disk drive and use redo data to recover the database to 1 p.m., reconstructing the lost changes.

## 30.3 Elements of Backup and Recovery Strategy

Although backup and recovery operations can be intricate and vary from one business to another, the basic principles follow these four simple steps:

1. Multiplex the online redo logs
2. Run the database in ARCHIVELOG mode and archive redo logs to multiple locations
3. Maintain multiple concurrent backups of the control file
4. Take frequent backups of physical datafiles and store them in a safe place, making multiple copies if possible. As long as users have backups of DB and archive redo logs in safe storage, the original database can be recreated.

## 30.4 Key Data Structures for Backup and Recovery

Before users begin to think seriously about backup and recovery strategy, the physical data structures relevant for backup and recovery operations must be identified. This section discusses the following physical data structures:

- Datafiles
- Archived Redo Log Files
- Control Files
- Automatic Managed Undo
- Online Redo Log Files

### 30.4.1 Datafiles

Every Oracle DB has one or more physical datafiles that belong to logical structures called tablespaces. The datafile is divided into smaller units called data blocks. The data of logical database structures, such as tables and indexes, is physically located in the blocks of the datafiles allocated for a database. Datafiles hold the following characteristics:

User-defined characteristics allow datafiles to automatically extend when the database runs out of space.

One or more physical datafiles form a logical database storage unit called a tablespace.

The first block of every datafile is the header. The header includes important information such as file size, block size, tablespace, and creation timestamp. Whenever the database is opened, Oracle checks to see that the datafile header information matches the information stored in the control file. If it does not, then recovery is necessary.

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

### 30.4.2 Control Files

Every Oracle database has a control file containing the operating system filenames of all other files that constitute the database. This important file also contains consistency information that is used during recovery, such as the:

- Database name
- Timestamp of database creation
- Names of the database's datafiles and online and archived redo log files
- Checkpoint, a record indicating the point in the redo log where all database changes prior to this point have been saved in the datafiles
- Recovery Manager(RMAN) backup meta-data

Users can multiplex the control file, allowing Oracle to write multiple copies of control file to protect it against disaster. If the operating system supports disk mirroring, the control file can also be mirrored, allowing the O/S to write a copy of the control file to multiple disks. Every time a user

mounts an Oracle database, its control file is used to identify the datafiles and online redo log files that must be opened for database operation.

If the physical makeup of the database changes, such as a new datafile or redo log file is created, Oracle then modifies the database's control file to reflect the change. The control file should be backed up whenever the structure of the database changes. Structural changes can include adding, dropping, or altering datafiles or tablespaces and adding or dropping online redo logs.

### 30.4.3 Online Redo Log Files

Redo logs are absolutely crucial for recovery. For example, imagine that a power outage prevents Oracle from permanently writing modified data to the datafiles. In this situation, an old version of the data in the datafiles can be combined with the recent changes recorded in the online redo log to reconstruct what was lost. Every Oracle database contains a set of two or more online redo log files. Oracle assigns every redo log file a log sequence number to uniquely identify it. The set of redo log files for a database is collectively known as the database's redo log.

Oracle uses the redo log to record all changes made to the database. Oracle records every change in a redo record, an entry in the redo buffer describing what has changed. For example, assume a user updates a column value in a payroll table from 5 to 7. Oracle records the old value in undo and the new value in a redo record. Since the redo log stores every change to the database, the redo record for this transaction actually contains three parts:

- The change to the transaction table of the undo
- The change to the undo data block
- The change to the payroll table data block

If user then commits the update to payroll table - to make permanent changes executed by SQL statements - Oracle generates another redo record. In this way, system maintains a careful watch over everything that occurs in the DB.

#### 30.4.3.1 Circular Use of Redo Log Files

Log Writer (LGWR) writes redo log entries to disk. Redo log data is generated in the redo log buffer of the system global area. As transactions commit and the log buffer fills, LGWR writes redo log entries into an online redo log file. LGWR writes to online redo log files in a circular fashion, when it fills current online redo log file, called active file, LGWR writes to next available inactive redo log file. LGWR cycles through the online redo log files in the database, writing over old redo data. Filled redo log files are available for reuse depending on whether archiving is enabled:

If archiving is disabled, a filled online redo log is available once the changes recorded in the log have been saved to the datafiles.

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

### 30.4.4 Archived Redo Log Files

Archived log files are redo logs that Oracle has filled with redo entries, rendered inactive, and copied to one or more log archive destinations. Oracle can be run in either of two modes:

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

Running the database in ARCHIVELOG mode has the following benefits:

- The database can be completely recovered from both instance and media failure.
- The user can perform online backups, i.e., back up tablespaces while the DB is open and available for use.
- Archived redo logs can be transmitted and applied to the standby database
- Oracle supports multiplexed archive logs to avoid any possible single point of failure on the archive logs.

- User has more recovery options, such as the ability to perform tablespace-point-in-time recovery (TSPITR)

Running the database in NOARCHIVELOG mode has the following consequences:

- The user can only back up the database while it is completely closed after a clean shutdown.
- Typically, the only media recovery option is to restore the whole database, which causes the loss of all transactions issued since the last backup.

### 30.4.5 Automatic Managed Undo

Every Oracle DB must have a method of maintaining information that is used to roll back, or undo, changes to the DB. Such information consists of records of actions of transactions, primarily before they are committed. Oracle refers to these records collectively as undo. Historically, Oracle has used rollback segments to store undo.

Space management for these rollback segments has proven to be quite complex. Oracle now offers another method of storing undo that eliminates complexities of managing rollback segment space, and allows DBAs to exert control over how long undo is retained before being overwritten. This method uses an undo tablespace. Undo records are used to:

- Roll back transactions when a ROLLBACK statement is issued
- Recover the database
- Provide read consistency

When a rollback statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

## 30.5 Understanding Basic Backup

A backup strategy provides a safeguard against data loss. Answering the following questions can help database administrators develop a strong backup strategy:

### 30.5.1 What Types of Failures can occur?

Data loss can occur for various reasons. Here are some most common types of failures that can lead to data loss.

A statement failure is a logical failure in the handling statement in an Oracle program. For example, a user issues a statement that is not a valid SQL construction. When statement failure occurs, Oracle automatically undoes any effects of the statement and returns control to the user.

A process failure is a failure in a user process accessing Oracle, i.e., an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. If the user process fails while modifying the DB, Oracle background processes undo the effects of uncommitted transactions.

An instance failure is a problem that prevents an Oracle instance, i.e., the SGA and background processes, from continuing to function. Instance failure can result from a hardware problem such as a power outage, or a software problem such as an operating system crash. When an instance fails, Oracle does not write the data in the buffers of the SGA to the datafiles.

A user or application error is a user mistake that results in the loss of data. For example, a user can accidentally delete data from a payroll table. Such user errors can require the database or object to be recovered to a point in time before the error occurred. To allow recovery from user error and accommodate other unique recovery requirements, Oracle provides Flashback Technology.

A media failure is a physical problem that arises when Oracle tries to write or read a file that is

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

### 30.5.2 What Information should be Backed Up?

A database contains a wide variety of types of data. When developing backup strategy, DBAs must decide what information they want to copy. The basic backup types include:

- Online DB Backup
- Offline DB Backup
- Whole Database
- Tablespace
- Datafile
- Control File
- Archived Redo Log
- Configuration Files

In deciding what to back up, the basic principle is to prioritize data depending on its importance and degree to which it changes. Archive logs do not change, for example, but they are crucial for recovering the DB, so multiple copies should be maintained, if possible. Expense account tables, however, are constantly updated by users.

Therefore, this tablespace should be backed up frequently to prevent having to apply as much redo data during recovery. Backups can be combined in a variety of ways. For example, a DBA can decide to take weekly whole DB backups, to ensure a relatively current copy of original DB information, but take daily backups of the most accessed tablespaces. The DBA can also multiplex the all important control file and archived redo log as an additional safeguard.

#### 30.5.2.1 Online Database Backup

An online backup or also known as an open backup is a backup in which all read-write datafiles and control files have not been check pointed with respect to the same SCN. For example, one read-write datafile header may contain an SCN of 100 while other read-write datafile headers contain an SCN of 95 or 90.

Oracle cannot open the database until these entire header SCNs are consistent, that is, until all changes recorded in the online redo logs have been saved to the datafiles on disk. If the database must be up and running 24 hours a day, 7 days a week, then we have no choice but to perform online backups of a whole database which is in ARCHIVELOG mode.

#### 30.5.2.2 Offline Database Backup

In this backup all datafiles and control files are consistent to same point in time - consistent with respect to same SCN, for example. Only tablespaces in a consistent backup that are allowed to have older SCNs are read only and offline normal tablespaces, which are consistent with the other datafiles in the backup.

This type of backup allows the user to open the set of files created by the backup without applying redo logs, since the data is already consistent. The only way to perform this type of backup is to shut down the DB cleanly and make the backup while the DB is closed. A consistent whole DB backup is the only valid backup option for DB running in NOARCHIVELOG mode.

#### 30.5.2.3 Whole Database Backup

The most common type of backup, a whole database backup contains the control file along with all database files that belong to a database. If operating in ARCHIVELOG mode, the DBA also has the option of backing up different parts of the database over a period of time, thereby constructing a whole database backup piece by piece.

#### 30.5.2.4 Tablespace Backups

A tablespace backup is a subset of the database. Tablespace backups are only valid if the database is operating in ARCHIVELOG mode. The only time a tablespace backup is valid for a database running in NOARCHIVELOG mode is when that tablespace is read-only or offline-normal.

### 30.5.2.5 Datafile Backups

A datafile backup is a backup of a single datafile. Datafile backups, which are not as common as tablespace backups and are only valid if the database is run in ARCHIVELOG mode. The only time a datafile backup is valid for a DB running in NOARCHIVELOG mode is if that datafile is the only file in a tablespace. For example, the backup is a tablespace backup, but the tablespace only contains one file and is read-only or offline-normal.

### 30.5.2.6 Control File Backups

A control file backup is a backup of a DB's control file. If a DB is open, the user can create a valid backup by issuing the following SQL statement: ALTER DATABASE BACKUP CONTROLFILE to 'location'; or use RMAN.

### 30.5.2.7 Archived Redo Log Backups

Archived redo logs are the key to successful media recovery. Depending on the disk space available and the number of transactions executed on the database, we want to keep as many days of archive logs on disk and we want to back them up regularly to ensure a more complete recovery.

### 30.5.2.8 Configuration Files

Configuration files may consist of spfile or init.ora, password file, tnsnames.ora, and sqlnet.ora. Since these files do not change often, then they require a less frequent backup schedule. If we lost a configuration file it can be easily recreated manually. When restore time is a premium, it will be faster to restore a backup of the configuration file then manually creating a file with a specific format.

## 30.5.3 Which Backup Method should be used?

Oracle provides users a choice of several basic methods for making backups. The methods include:

- Oracle Data Pump - The utility makes logical backups by writing data from an Oracle database to operating system files in a proprietary format. This data can later be imported into a database.
- User Managed - The DB is backed up manually by executing commands specific to user's operating system.
- Recovery Manager (RMAN) - A component that establishes a connection with a server process and automates the movement of data for backup and recovery operations.
- Oracle Enterprise Manager - A GUI interface that invokes Recovery Manager.

### 30.5.3.1 Using the Data Pump for Supplemental Backup Protection

Physical backups can be supplemented by using the Data Pump utility to make logical backups of data. Logical backups store information about the schema objects created for a database. Data Pump writes data from a database into Oracle files in a proprietary format, which can then be imported into a database using the Import utility.

### 30.5.3.2 User Managed Backups

Operating system commands can be used such as UNIX dd or tar command to make backups. Backup operations can also be automated by writing scripts. User can make a backup of whole database at once or back up individual tablespaces, datafiles, control files, or archived logs.

A whole database backup can be supplemented with backups of individual tablespaces, datafiles, control files, and archived logs. O/S commands can also be used to perform these backups if the database is down or if the database is placed into hot backup mode to take an online backup.

### 30.5.3.3 Making Recovery Manager Backups

Recovery Manager (RMAN) is a powerful and versatile program that allows users to make an RMAN backup or image copy of their data. When the user specifies files or archived logs using the RMAN BACKUP command, By default RMAN creates a backup set as output. A backup set is a file or files in a proprietary-specific format that requires the use of the RMAN RESTORE command for recovery operations. In contrast, when the BACKUP AS COPY command is used to create an image copy of a file, it is in an instance-usable format - the user does not need to invoke RMAN to restore or recover it.

When a RMAN command is issued, such as backup or restore, RMAN establishes a connection to an Oracle server process. The server process then back up the specified datafile, control file, or archived log from the target DB. The recovery catalog is a central repository containing a variety of information useful for backup and recovery.

RMAN automatically establishes the names and locations of all the files needed to back up. RMAN also supports incremental backups, backups of only those blocks that have changed since a previous backup. In traditional backup methods, all the datablocks ever used in a datafile must be backed up.

#### Automatic Disk-Based Backup and Recovery

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, we 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 our database within that space. Files no longer needed are eligible for deletion when RMAN needs to reclaim space for new files.

If we do not use a flash recovery area, we must manually manage disk space for our backup-related files and balance the use of space among the different types of files. Oracle Corporation recommends that we enable a flash recovery area to simplify our backup management.

### 30.5.3.4 Oracle Enterprise Manager

Although Recovery Manager is commonly used as a command-line utility, the Backup Wizards in Oracle Enterprise Manager is the GUI interface that enables backup and recovery via a point-and-click method. Oracle Enterprise Manager (EM) supports Backup and Recovery features commonly used by users.

- Backup Configurations to customize and save commonly used configurations for repeated use
- Backup and Recovery wizards to walk the user through the steps of creating a backup script and submitting it as a scheduled job
- Backup Job Library to save commonly used Backup jobs that can be retrieved and applied to multiple targets
- Backup Management to view and perform maintenance on RMAN backups.

## 30.6 Understanding Basic Recovery Strategy

Basic Recovery involves two parts: Restoring a physical backup and then updating it with changes made to DB since the last backup. Most important aspect of recovery is making sure all data files are consistent with respect to same point in time. Oracle has integrity checks that prevent user from opening DB until all data files are consistent with one another. When preparing a recovery strategy, it is critical to understand answers to these questions:

- How does recovery work?
- What are the types of recovery?
- Which recovery method should be used?

### 30.6.1 How does Recovery Work?

In every type of recovery, Oracle sequentially applies redo data to data blocks. Oracle uses information in the control file and datafile headers to ascertain whether recovery is necessary. Recovery has two parts: rolling forward and rolling back. When Oracle rolls forward, it applies redo records to the corresponding data blocks.

Oracle systematically goes through the redo log to determine which changes it needs to apply to which blocks, and then changes the blocks. For example, if a user adds a row to a table, but the server crashes before it can save the change to disk, Oracle can use the redo record for this transaction to update the data block to reflect the new row. Once Oracle has completed the rolling forward stage, the Oracle database can be opened.

The rollback phase begins after the database is open. The rollback information is stored in transaction tables. Oracle searches through the table for uncommitted transactions, undoing any that it finds. For example, if the user never committed the SQL statement that added the row, then Oracle will discover this fact in a transaction table and undo the change.

### 30.6.2 What are the types of Recovery?

Three basic types of recovery:

- (a)Instance Recovery (b)Crash Recovery (c)Media Recovery

Oracle performs the first two types of recovery automatically at instance startup. Only media recovery requires the user to issue commands. An instance recovery, which is only possible in an Oracle Real Applications Cluster configuration, occurs in an open database when one instance discovers that another instance has crashed. A surviving instance automatically uses the redo log to recover the committed data in the DB buffers that was lost when the instance failed. Oracle also undoes any transactions that were in progress on the failed instance when it crashed, then clears any locks held by the crashed instance after recovery is complete.

A crash recovery occurs when either a single-instance DB crash or all instances of multi-instance database crash. In crash recovery, an instance must first open the database and then execute recovery operations. In general, the first instance to open the DB after a crash or SHUTDOWN ABORT automatically performs crash recovery. Unlike crash and instance recovery, a media recovery is executed on the user's command, usually in response to media failure. In media recovery, online or archived redo logs can be used to make a restored backup current or to update it to a specific point in time.

Media recovery can restore the whole DB, a tablespace or a datafile and recover them to a specified time. Whenever redo logs are used or a DB is recovered to some non-current time, media recovery is being performed. A restored backup can always be used to perform the recovery. The principal division in media recovery is between complete and incomplete recovery. Complete recovery involves using redo data combined with a backup of a DB, tablespace, or

datafile to update it to the most current point in time. It is called complete because Oracle applies all of the redo changes to the backup. Typically, media recovery is performed after a media failure damages datafiles or the control file.

### 30.6.2.1 Recovery Options

If the user does not completely recover the database to the most current time, Oracle must be instructed how far to recover. The user can perform:

- Tablespace point-in-time recovery (TSPITR), which enables users to recover one or more tablespaces to a point-in-time that is different from the rest of the database.
- Time-based recovery, also called point-in-time recovery (PITR), which recovers the data up to a specified point in time.
- Cancel-based recovery, which recovers until the CANCEL command is issued.
- Change-based recovery or log sequence recovery. If O/S commands are used, change-based recovery recovers up to a specified SCN in the redo record.
- Flashback from human error

If Recovery Manager is used, log sequence recovery recovers up to a specified log sequence number. When performing an incomplete recovery, the user must reset the online redo logs when opening the database. The new version of the reset database is called a new incarnation. Opening the database with the RESETLOGS option tells Oracle to discard some redo. In Oracle Database 11g, the control file has added new structures that provides the ability to recover through a RESETLOGS operation using backups from a previous incarnation.

### 30.6.2.2 Recovering From Human Errors

The Oracle Database 11g architecture leverages the unique technological advances in the area of database recovery due to human errors. Flashback Technology provides a set of new features to view and rewind data back and forth in time. The Flashback features offer the capability to query past versions of schema objects, query historical data perform change analysis or perform self-service repair to recover from logical corruptions while the database is online. With the Oracle Database 11g Flashback Technology, we can indeed undo the past!

### 30.6.3 Which Recovery method should be used?

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

#### 30.6.3.1 Recovering with Recovery Manager

The basic RMAN commands are RESTORE and RECOVER. RMAN can be used to restore datafiles from backup sets or image copies, either to their current location or to a new location. If any archived redo logs are required to complete the recovery operation, RMAN automatically restores and applies them. In a recovery catalog, RMAN keeps a record containing all the essential information concerning every backup ever taken.

If a recovery catalog is not used, RMAN uses the control file for necessary information. The RMAN RECOVER command can be used to perform complete media recovery and apply incremental backups, and to perform incomplete media recovery.

#### 30.6.3.2 Recovering with SQL\*Plus

Administrators can use the SQL\*Plus utility at the command line to restore and perform media recovery on our files. Users can recover a database, tablespace, or datafile. Before performing recovery, users need to:

- Determine which files to recover. Often the table V\$RECOVER\_FILE can be used.
- Restore backups of files permanently damaged by media failure. If the user does not have a backup, recovery can still be performed if the user has the necessary redo log files and the control file contains the name of the damaged file.
- If a file cannot be restored to its original location, then the user must relocate the restored file and inform the control file of the new location.
- Restore necessary archived redo log files.

## 31. Cold Backup (Physical)

### 31.1 Overview

Here we are concentrating on PB. The basic difference between COLD & HOT is in COLD --> The Instance must be down. So let's concentrate on COLD-PB. In this, we shutdown the Database and try to BKUP 3 types of files (CTLs, REDO-LOGS, DBFs, optionally INIT.ORA & Password File).

Usually these 3 files we backup to TAPE drive, and if time permits it's a good idea to also backup to another server, so that in case of the first server is dead meaning that it's not booting, at least we can have all our production users connect to the other server, which we usually call it as "Standby" Server. Basically there are 2 modes the Database can be put into, which are

1. No-Archivelog-Mode (NALM)
2. Archivelog-Mode (ALM)

Although the backup method is same whether the Database is running in either case, the recovery part differs. Now coming to "Recovery", there are 2 methods in recovery.

1. Simple Restore (SR)
2. Restore & Recovery (R + R)

Let us observe a scenario where on a Tuesday morning at 11AM, the Database crashed and we need to get this back. Here let's assume the Database is running in No-Archivelog-Mode. In this case all we can do is perform a "Simple Restore" from our Monday's Night Tape-Backup. But as you can observe, we are not going to get any updates to database, which happened during the Tuesday morning (8-11am). Basically this is not the industry standard option since we are losing lot of data which means our end-users need to RE-INPUT that data again. In some scenarios our BOSS may opt this incase,

1. The server is very slow already and by running the Database in "Archive-Log-Mode" it may even become slower. All transactions are basically can be reproduced because we are doing the data-entry from some physical-media (e.g. emp-hours are received in our FAX-Sheets). On the contrary, regardless our machine being slow or not, we have to turn-on "Archive-Logging", since the transactions performed by our end-users are basically cannot be re-produced.
2. Updating an Employee's Address while the EMP is on the phone. Credit-Card transactions (which are done electronically). Well, there are so many other examples we can give. But the point is in case if this kind of DB is put under "No-Archivelog-Mode", and if it gets crashed, we can only perform "Simple-Restore", meaning that we are going to lose lot of data which is done by our end-users on that day, since now we have a "Last-Night Database". Unfortunately our end-users cannot reproduce the data at any cost, since it's not recorded anywhere. In Simple-Restore, there is nothing much to talk about since all we are doing is, getting all these 3 types of files from our TAPE. But, in Restore & Recovery, there are some issues to observe.

### 31.2 Complete Recovery & Incomplete Recovery

In order to perform #1, we have to have

- (a) Present Control-File
- (b) Current Online-Redo-LogFile

If any of these files (or both) are missing, we can only perform ICR, since

- If Current Redo is missing, we are losing most recent transactions.
- If Present-Control-File is missing, although we are applying the current-online-redolog file, Oracle has no idea that this is the latest file, since that is known only to CF, which is missing (in this case, according to DBA, it's complete-recovery, but according to Oracle it's ICR).

## 32. Recovery Matrix

Every Oracle Database could be running in any of the two modes:

1. NOARCHIVELOG Mode
2. ARCHIVELOG Mode

Before we move further, let us understand two terms:

- **Restore:** Bring back the lost files from the most recent backup.
- **Recover:** Applying the archive logs on the restored files.

If the Database is operating in NOARCHIVELOG Mode, the only option is to restore the most recent full backup and redo all the work performed after taking the backup.

If the Database is operating in ARCHIVELOG Mode, several recovery options are possible, depending on the type of file being lost.

Again, Recovery could be **Complete** or **Incomplete**

Complete Recovery results due to loss of any Datafile and could be performed Online (with DB up) or Offline (with DB down).

- **Online Complete Recovery** can be performed, when we lose any Non-System Datafile.
- **Offline Complete Recovery** can be performed, when we lose any System/Undo Datafile.

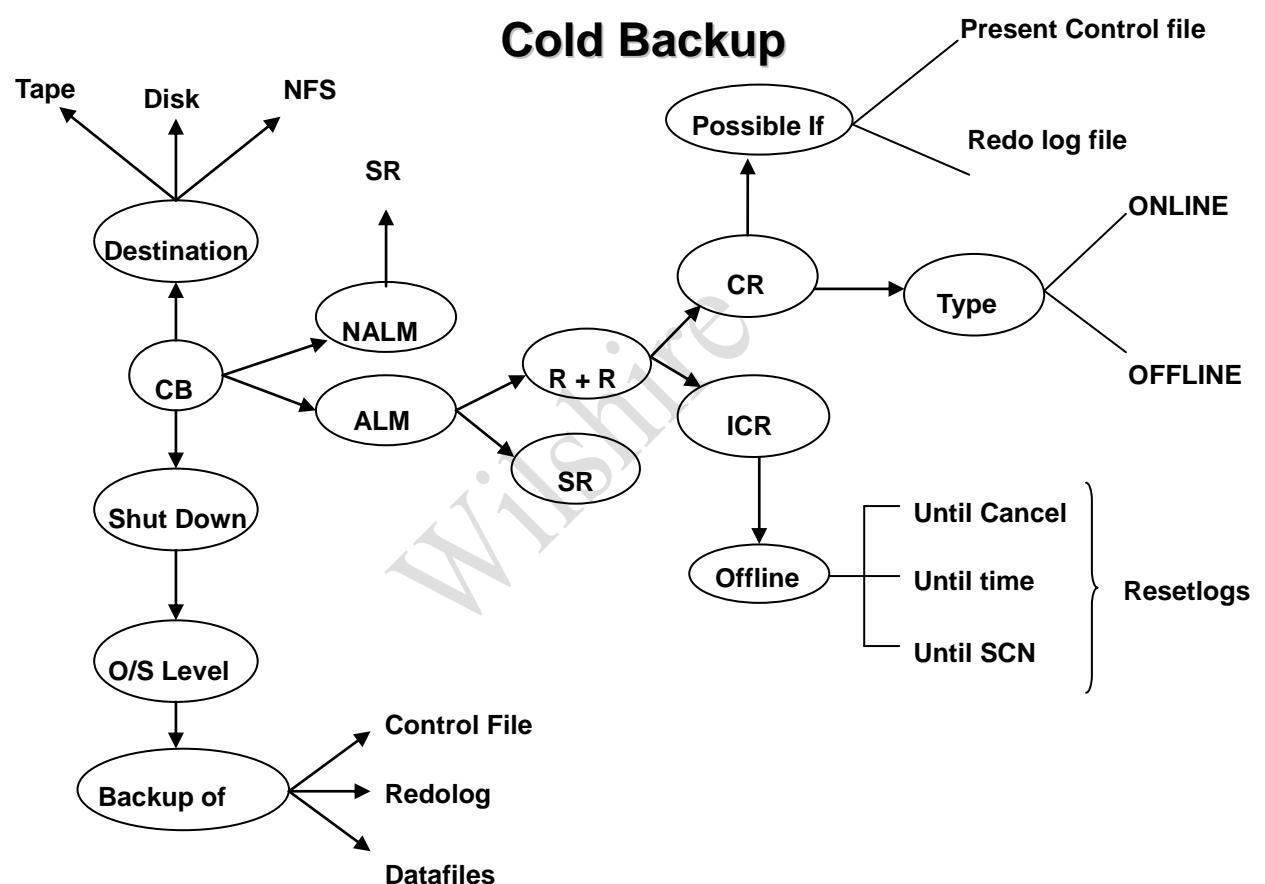
Incomplete Recovery results due to loss of current online log or Control file (either or both) and is always performed Offline (with DB down) and we need to open the Database with RESETLOGS option.

The table below shows how to recover from media failure depending on the type of file being lost.

| S.No | Data Files                       | Online Redo Logs | Archive Redo Logs | Control File | Online / Offline | Complete / Incomplete       | Conclusion                                                                   |
|------|----------------------------------|------------------|-------------------|--------------|------------------|-----------------------------|------------------------------------------------------------------------------|
| 1    | ✗                                | ✗                | ✗                 | ✗            | Offline          | Complete (as per last bkup) | Only Restore. Recovery is not possible because we lost the archive logs also |
| 2    | ✗                                | ✗                | ✓                 | ✗            | Offline          | Incomplete                  | Restore + Recover until the last available archive log file                  |
| 3    | ✗ (Non-System dbfs)              | ✓                | ✓                 | ✓            | Online           | Complete                    | Restore & Recover lost datafiles of Non-System Tablespaces                   |
| 4    | ✗ (Including System / Undo dbfs) | ✓                | ✓                 | ✓            | Offline          | Complete                    | Restore & Recover all the Datafiles including System/Undo Tablespaces        |
| 5    | ✓                                | ✗                | ✓                 | ✓            | Offline          | Incomplete                  | If all the members of a group are lost                                       |

|   |   |   |   |   |         |                          |                                                                                             |
|---|---|---|---|---|---------|--------------------------|---------------------------------------------------------------------------------------------|
| 6 | ✓ | ✓ | ✓ | ✗ | Offline | Could be Complete or ICR | Complete, if you recreate the ctl file. Incomplete, if you recover from backup control file |
| 7 | ✓ | ✗ | ✓ | ✗ | Offline | Incomplete               | As both online logs & control files are lost                                                |

**Note:** In all the above cases we assume that we have a valid Physical Backup (Cold or Hot) and all the archive logs that got generated after taking the backup.



## 33. Backup & Recovery Scenarios - 1

### 33.1 CASE 1: Full Database Recovery

We backup our database every night at 10PM. We have our Database running in Archivelog mode. On Tuesday at 11:00AM we lost everything. (How? we don't know).

Target: Try to get as much as possible.

Observation: The last Archivelog file got generated was about 10:58AM.

Action: Since we lost everything, we have to depend on yesterday's backup (Monday night's backup). Which means we will apply all "available" Archivelog files (generated on Tuesday), on top of Monday Night's Database.

Does Monday's Backup needs Recovery? Certainly not! But we are forcing it to under-go Recovery (after Restore). As far as Monday Night's Backup is concerned, that Controlfile says everything is fine (as far all SCN numbers are concerned).

But, since we want to apply Tuesday's Archivelog files on top of it we are going to treat the Monday's controlfile as "Backup-Controlfile", and the recovery is known as "In-Complete Recovery"

Complete Recovery can only be done in case of Present Controlfile (in this case Tuesday's 11 AM File)

3. Current On-Line-Redolog (-do-) file's availability. In case of either of the files is missing, we can only do "In-Complete Recovery". So, in the above case, we issued a command to perform

### At the OS Level, first RESTORE all 3 types of DB-Files. Then

```
SQL> STARTUP MOUNT
SQL> ALTER DATABASE RECOVER AUTOMATIC
 USING BACKUP CONTROLFILE UNTIL CANCEL;
SQL> RECOVER CANCEL;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

(The reason we are resetting the LOG files is because, the REDOs we have from Monday are at lower sequence number than our DBFs, since we have applied Tuesday's Archivelog files.). In the above scenario, after the recovery, we are fine except we lost latest changes made by users, which were recorded in the Current-On-Line-Redolog file, which was lost.

- Why we're treating Monday's controlfile as backup controlfile?
- Why we're saying its in-complete recovery?
- Why we have to reset-logs at the end of the recovery?

### 33.2 Case 2: Loss of a Non-SYSTEM datafile

On Thursday morning at 10:30 AM, we lost a UNIX File system, which resulted in loss of a Database file.

Tablespace: "USERS \_ DATA" (has 4 datafiles associated with it)

Datafile: "/disk1/oradata/ORCL/users\_data1.dbf"

Observation: Since we lost this file, users are getting an Oracle-Error message, saying that 'datafile' not found.

Target: Would like to fix this problem on-line, since there are other users who are working in other tablespaces do not want to stop working.

Type: For this scenario we can perform Offline or Online Complete Recovery. (On-line recovery is not possible in case if the data-file belongs to SYSTEM TS) So, if the file is lost which belongs to SYSTEM-TS, then we can only perform off-line CR.

Action: After fixing the UNIX FS, we have copied only the lost datafile from our Wednesday night's backup. This file is surely at an out-dated SCN, which needs to be "Recovered".

When a non-SYSTEM data file is lost, there are three manual methods by which the data file can be recovered.

### 33.2.1 Method 1: Database Recovery

Now, let's test the first method in recovering this database by using **RECOVER DATAFILE** command:

```
SQL> STARTUP MOUNT
SQL> RECOVER DATABASE;
SQL> ALTER DATABASE OPEN;
```

(To do the database recovery, which requires the database to be mounted but not open (offline recovery).

### 33.2.2 Method 2: Data File Recovery

This method involves taking the data file offline and opening the database before issuing the **RECOVER DATAFILE** command;

```
SQL> STARTUP MOUNT - No need if database already opened.
SQL> ALTER DATABASE DATAFILE '/disk1/oradata/ORCL/user_data1.dbf'
OFFLINE;
SQL> ALTER DATABASE OPEN; - No need if database is already open.
SQL> ALTER DATABASE RECOVER AUTOMATIC
 DATAFILE '/disk1/oradata/ORCL/user_data1.dbf';
SQL> ALTER DATABASE DATAFILE '/disk1/oradata/ORCL/user_data1.dbf' ONLINE;
```

### 33.2.3 Method 3: Tablespace Recovery

You can also perform a tablespace recovery. This method requires tablespace to be offline and the database is open:

```
SQL> STARTUP MOUNT
SQL> alter database datafile '/disk1/oradata/ORCL/user_data1.dbf'
offline;
SQL> ALTER DATABASE OPEN;
(Perform above statements only when your database is down)
SQL> ALTER TABLESPACE USER_DATA OFFLINE;
sql> RECOVER TABLESPACE USER_DATA;
sql> ALTER TABLESPACE USER_DATA ONLINE;
```

At this point using any one of these methods, the file is brought up to the right SCN number and can function normally. In this scenario, we haven't lost any data. We did use

1. Present Controlfile
2. Current Redo Log file, which results in "Complete Recovery".

Since we have Present-controlfile & Current-On-Line-Redolog files, we don't need to use "using backup controlfile" or "until cancel" options. And also we didn't need to "RESET" the Redo-Log files, as this is "Complete Recovery".

The only difference in ON-LINE-Recovery and OFF-LINE-Recovery is users are working in On-line and Instance is UP. But while recovering, we are mentioning the name of the file which we have restored from the previous night and which needs "recovery". Where as in OFF-LINE recovery, since the database was just mounted, we could able to do "recover automatic" and that keyword knows which datafile to fix (need not mention the file name as in On-line-Recovery). Both procedures would work fine. It's just your choice which one to pickup.

### 33.3 CASE 3: Loss of a System Data File

There are only a limited number of recovery options that you have if some of the crucial data files such as the SYSTEM data files are lost. A solid backup procedure is the only way out of such disasters.

We backup our database every night at 10PM. We have our Database running in Archivelog mode.

**PROBLEM:** On Tuesday at 11:00AM a disk crashed, losing the SYSTEM data file residing on the disk. As this happened at peak processing time, DBA had to keep the downtime to a minimum and open the database as soon as possible.

**SOLUTION:** The only solution here is to restore the SYSTEM data file from the previous night's backup and then perform the database complete recovery.

SQL> STARTUP MOUNT

```
SQL> RECOVER DATABASE;
SQL> ALTER DATABASE OPEN;
```

Note that if the disk crash has damaged several data files, then all the damaged data files need to be restored from the backup. The database needs to be mounted and the **recover database** command issued before the database can be opened.

### 33.4 CASE 4: Loss of an Object and Point in Time Recovery.

One of the managers dropped a table around 2PM on Tuesday and realized this stupid thing at 4PM.

**OBSERVATION:** People are working left & right on the system.

**CHOICE:** You ask the manager, if he provides the table as of yesterday could satisfy him. If he says yes all you have to do is:

```
$ imp system/manager fromuser=scott touser=scott tables=emp \ file=exp.dmp
```

(Using this method, you are basically extracting the table from your logical backup without touching any physical files or restore/recovery procedures). In case if he did some modifications to this table, which he says, can't lose, you have no choice but go for recovery. But there is something to understand here before proceeding any further.

The issue is, somehow if we can go back to 2PM, would that do or not? This is the question you should ask your end-users. If they say, its fine, your problem is simplified. But if they say that is not an accepted choice - which means they need to have the database as it is (the way it looked at 4PM). This would be a difficult case for the DBA, since he has to first get TABLE (do TIME-BASED-RECOVERY up to 2PM) and again go up to 4PM.

Here you should see if there is any other server available for you. In case it is available, your work is reduced to 25%, but the server must be of exactly of same type (OS, Oracle Version etc.). And it should have enough disk-space to do RESTORE of your last night's backup. (For e.g. we call the server as our Development Server where as the "TABLE DROP" occurred on our Production Box).

The most unfortunate thing is Oracle can't walk backwards. Please don't think that we can go back in time (which means, we can put 4PM database and go back to 2PM). This is not possible. Oracle offers "Roll Forward" mechanism only. (Don't confuse this word by comparing with "commit" & "rollback", which is only per transaction basis).

So, now we are on development server (which means we haven't touched our production server yet. End-Users are still working on the Prod). On the Development, "RESTORE" last-night's Production database. And also copy all the "ARCHIVELOG" files from "PRODUCTION" to "DEVELOPMENT" which were generated today. These files are going to help us in

performing "ROLL-FORWARD" until 2PM, which is known as "TIME-BASED RECOVERY".

```

SQL> STARTUP MOUNT;
SQL> ALTER DATABASE RECOVER AUTOMATIC
 USING BACKUP CONTROLFILE UNTIL TIME '2004-10-24-14:00:00';
SQL> ALTER DATABASE OPEN RESETLOGS;

```

Now you should be able to see the EMP Table under SCOTT. All you have to do is:

```

$ exp scott/tiger file=scott_emp.dmp tables=emp
$ imp scott/tiger@dev12prod file=scott_emp.dmp

```

After doing this, we should have the table EMP in our Production Server also. By doing this way, we really didn't touch our production End-Users and at the same time, we saved our Manager by giving him his LOST Table EMP back. But if there is no other server except our Prod, then perform the following steps:

1. Ask your users to exit and perform Backup. This backup we call it as a BACKUP\_4PM.
2. Now restore your LN backup.
3. Do a timed-based recovery as explained above.
4. Export the table out of the database onto a ".dmp" file.
5. Shutdown this database
6. Restore the original 4PM database performed in Step#1.
7. Import this table (from Step#4) into Scott's account.

So doing this way, there is extra effort (backing up the 4PM database, which will take some time.) But either method you take, basically by going through TIME-BASED recovery, lost objects can be brought-over.

### 33.5 CASE 5: Loss of an Un-archived Online Log File

**PROBLEM:** A power surge caused the database to crash and also caused a media failure, losing all the online log files. All the data files and the current control files are intact.

**OBSERVATION:** Although the data files are OK after the crash, these files cannot be used because crash recovery cannot be performed (since all online log files are lost). Forcing the database open in a situation like this can cause database inconsistency. If any of the un-archived log files are lost, crash recovery cannot be performed and, instead, media recovery needs to be performed.

**SOLUTION:** In this case, all the datafiles need to be restored from an online (or offline) full backup and rolled forward until the last available archived log file is applied. Since this is incomplete recovery.

```

$ cp /disk1/oradata/DEMO/BKUP/*.dbf /disk1/oradata/DEMO
SQL> startup mount
SQL> alter database recover automatic until cancel;
SQL> recover cancel;
SQL> alter database open resetlogs;

```

Note that all the datafiles need to be restored from the backup before applying recovery. The **recover database until cancel** command lets you apply the log files one at a time and you can cancel when the last archive log file is applied. After recovery, Oracle creates the online log files automatically when the database was opened with the RESETLOGS option. Multiplexing the log files is strongly recommended to protect the database from losing the online log files.

### 33.6 CASE 6: Recovery with a Backup Control File

**Description:** Accidentally controlfile is deleted and we didn't have any current copy of the controlfile. Now we had copied the controlfile from the backup and tried to start up the database. While opening the database, Oracle complained that an old control file was being used.

**Choice:** In this case, we have two options. Where all our datafiles and online log files are safe, we can create a new control file using the **create controlfile** command, perform recovery

if required, and start up the database. Alternatively, we can use the backup control file.

**SOLUTION:** If you use a backup control file, you need to perform media recovery. Also, Oracle will force you to use the using **backup controlfile** option. Once recovery is done, you must start up the database with the RESETLOGS option, and you have to take a complete backup of your database. For this reason, it's a better idea to use the first solution in this case.

### 33.6.1 Method 1:

The presence of a traced control file is very helpful in this case. But make sure that this traced control file is a latest copy of the latest structure of the database, i.e., no structural modifications to the database after the control file was traced. Finally, executed this traced controlfile at the NOMOUNT stage. If traced control file is not present, a CREATE CONTROLFILE script has to be built manually, taking into account all the datafiles and the redo-log files.

```
SQL> STARTUP NOMOUNT;
SQL> CREATE CONTROLFILE REUSE DATABASE "ACCT" ARCHIVELOG RESETLOGS
LOGFILE
 GROUP 1 (
 '/disk1/oradata/ACCT/redo1a.log' ,
 '/disk2/oradata/ACCT/redo1b.log'
) SIZE 4M,
 GROUP 2 (
 '/disk1/oradata/ACCT/redo2a.log' ,
 '/disk2/oradata/ACCT/redo2b.log'
) SIZE 4M
DATAFILE
 '/disk1/oradata/ACCT/sys01.dbf' ,
 '/disk1/oradata/ACCT/users01.dbf' ,
 '/disk1/oradata/ACCT/temp01.dbf' ,
 '/disk1/oradata/ACCT/rbs01.dbf' ;
SQL> RECOVER DATABASE;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Result: The database is back, without any data loss, and a little downtime.

### 33.6.2 Method 2:

```
$ cp /disk1/oradata/ACCT/BKUP/cont1.ctl /disk1/oradata/ACCT
SQL> RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL;
(If it needs more recovery then apply the online redo log files)
SQL> ALTER DATABASE OPEN RESETLOGS;
```

When you use a backup control file, you must use the USING BACKUP CONTROLFILE option with the **recover database** command.

## 34. Backup & Recovery Scenarios - 2

### 34.1 CASE 7: Loss of a Non-SYSTEM Data File with Rollback Segments

**PROBLEM:** On Monday morning, due to a media failure, all the data files that belong to the rollback segments tablespace ROLLBACK\_DATA were lost. Once we took the data file offline and opened the database, to select from a user table and got the Oracle error: that file cannot be read.

**SOLUTION:** If the data file that is lost belongs to a rollback segment tablespace, recovery could be tricky. For performing online recovery we have to take all the data files offline that were lost during the media failure. This means that all the rollback segments that belong to the data file need to be recovered.

In this test, to simulate the loss of data files, we will shut the database down with the ABORT option and delete the data file belonging to the ROLLBACK\_DATA tablespace at the OS level:

```
SQL> CONNECT SCOTT/TIGER
SQL> DELETE FROM EMP1 WHERE ROWNUM<10;
SQL> SHUTDOWN ABORT
SQL> EXIT
$ rm /disk1/oradata/DEMO/rbsdata01.dbf
```

Before we start recovery, a very important step needs to be performed here. The init.ora file needs to be modified, and the ROLLBACK\_SEGMENTS parameter needs to be commented out. If this is not done, Oracle will not be able to find the existing rollback segments and you won't be able to open the database.

```
SQL> STARTUP MOUNT
SQL> ALTER DATABASE DATAFILE '/disk1/oradata/DEMO/rbsdata01.dbf' OFFLINE;
SQL> ALTER DATABASE OPEN;
(If you need try to create temporary rollback segments in another
tablespace to process the running application users)
SQL> RECOVER DATAFILE 5; (File id of the lost rbsdata01.dbf data file)
SQL> ALTER DATABASE DATAFILE 5 ONLINE;
(If you check the status of the roll back segments that are created in
ROLLBACK DATA tablespace still it shows needs recovery)
SQL> ALTER ROLLBACK SEGMENT RBS1 ONLINE;
```

**OBSERVATION:** In this test, before we committed the transaction, we shut down the database with the ABORT option. This means that the rollback segment's transaction table will now show that the transaction is active. Now after the database open, Oracle cannot read the block because recovery has finished the roll forward but not the rollback of uncommitted transactions.

This means that Oracle cannot determine whether the row we have inserted is committed or rolled back. This is because we have taken the data files that contain the rollback segments offline. Once we have finished datafile/tablespace recovery, the rollback segments are still in NEEDS RECOVERY. At this point, we need to bring all the rollback segments online using the **alter rollback segment** command.

### 34.2 CASE 8: Database Crash During Hot Backups

On Friday afternoon, while taking hot backups, the machine crashed, bringing the database down. Once the machine was booted, he tried to start the database and Oracle asked for media recovery starting from log file sequence number 534. The current online log file had sequence number 566, which meant that about 25 logfiles needed to be applied before the database could be opened.

**SOLUTION:** Oracle gives the DBAs the ability to end the backups of the data files that are in hot backup mode by using the `ALTER DATABASE DATAFILE 'file_name' END BACKUP` command. You still need to apply recovery when the database crashes while taking online backups.

**EXAMPLE:**

```
SQL> CONN /AS SYSDBA
SQL> STARTUP
SQL> ALTER TABLESPACE TEST BEGIN BACKUP;
$ cp /disk1/oradata/DEMO/test01.dbf /disk1/oradata/DEMO/HBKUP
SQL> DELETE FROM TEST WHERE ROWNUM<101;
 SQL> ALTER SYSTEM SWITCH LOGFILE;
SQL> SHUTDOWN ABORT
```

Now we can perform two methods. The first method involves attempting to open the database with the current data file, test01.dbf. The second method involves replacing the current data file, test01.dbf, with the backup version and performing recovery.

**Method 1: Using the current data file**

```
SQL> STARTUP MOUNT
SQL> ALTER DATABASE DATAFILE '/disk1/oradata/DEMO/test01.dbf' END BACKUP;
SQL> ALTER DATABASE OPEN;
```

**Method 2: Using the Backup Data File**

```
$ cp /disk1/oradata/DEMO/HBKUP/test01.dbf /disk1/oradata/DEMO
SQL> STARTUP MOUNT
SQL> ALTER DATABASE DATAFILE '/disk1/oradata/DEMO/test01.dbf' END BACKUP;
SQL> RECOVER DATABASE;
SQL> ALTER DATABASE OPEN;
```

**OBSERVATION:** When a tablespace is in hot backup mode, only some data structures in the data file header (or headers, if multiple data files exist for that tablespace) are updated, while the others are frozen.

For example, if an update is done on a table in this data file, the update is not blocked. But when a checkpoint is done, the checkpointed at SCN value is not written to the file header. This means that if a crash occurs during a hot backup, Oracle really needs to update only the file header and not the contents of the data file while opening the database.

If you want to know if any data files are in hot backup mode, use the V\$BACKUP view, when the status says ACTIVE, then the file is in hot backup mode.

### 34.3 CASE 9: Recovery through RESETLOGS

**PROBLEM:** We take regular cold backups of the database. Figure 1.1 shows the time line and various events that occurred in a sequential order. The log sequence numbers at various times are pointed out as well.

At point A, we have taken a cold backup of the database and opened the database for normal operation. At point B, a media failure occurred and he lost all the online log files. Since we don't multiplex the online log files, we had to do incomplete recovery.

So, we restored the database as a point A and recovered the database using the `recover database until cancel` command. We applied recovery until the last archive log file and started up the database with RESETLOGS option at point B. As shown in figure 1.1, the current log sequence number was reset to 1 again at point B.

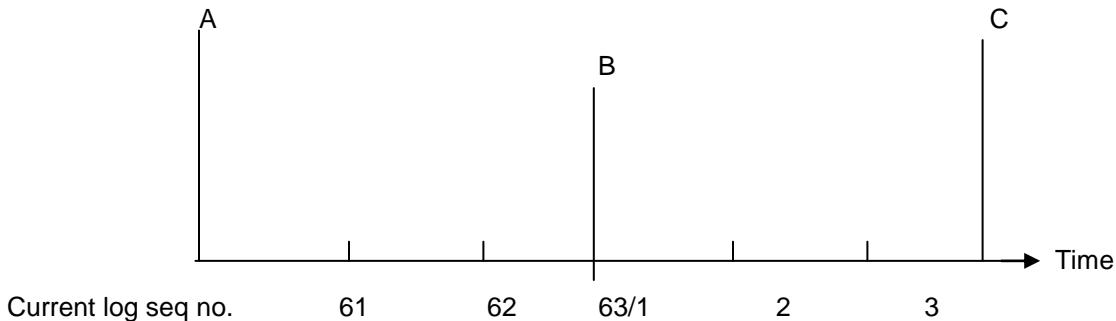


FIGURE 34.3.1. Database events at various points in time

After a few days (at point C), another media failure occurred and we lost data file 5. All the online log files and the control files were intact, so we restored the backup data file for file 5 from point A and tried to do recovery. Oracle complained that data file 5 is from a point before B. we tried to restore the backup control file and recover the database.

This time, Oracle complained that file 1 is from a point after B. Know we decided to restore all the data files and the control file from the cold backup (from point A) and tried to do recovery again. Recovery went fine until point B, but we couldn't go past point B, since Oracle didn't recognize the log files after point B. if we couldn't recover up to point C, we would lose all the data that we had inserted from point B to point C.

**CAUTION:** This is not an alternative for not backing up your database after a RESETLOGS! This method should be used only in emergencies!

**SOLUTION:** This is especially useful when there is an outage prior to successfully backing up your database after a RESETLOGS. Oracle Support Services recommends that a backup be taken after a RESETLOGS; however, this feature provides a workaround for customers who cannot take the time to do a cold backup immediately after doing recovery with RESETLOGS.

This feature should only be used under the following conditions:

- Oracle versions 7.3.3 and higher.
- Previous recovery with RESETLOGS has been successful.
- No consistent backup after the RESETLOGS is present.
- A consistent backup (cold or hot) prior to RESETLOGS is present.
- Control file after the RESETLOGS is present.
- All archive log files and online redo logs are present for recovery.

If you do not have any backups after point A, then recovery is a bit complicated, as you need to find out the exact SCN (at point B) at which RESETLOGS was done.

If the current control file (any control file after point B) is available, it is possible to use the backup taken before the RESETLOGS (at point A) and do the following:

1. Find the SCN immediately before the database was opened with RESETLOGS. This SCN is reported in the alert log just before the database is opened. Look for RESETLOGS after incomplete recovery UNTIL CHANGE xxxxxxx or RESETLOGS after complete recovery through change xxxxxxx. You should also query this information from any controlfile that is valid after the RESETLOGS point by using the following query:

```
SQL> SELECT RESETLOGS_CHANGE# - 1 FROM V$DATABASE;
```

The SCN in the alert log and the SCN from your query should return the same number. You will be recovering the database to this SCN.

2. Shut down the database. (At point C)

3. Copy the current control file to a safe location. (You must have a copy of the control file that was taken after the RESETLOGS was performed – it does not be the current control file).
4. Restore whole database and the control file from the backups before the RESETLOGS (point A of figure 1.1)
5. Recover the database using the command  
`SQL> RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CHANGE xxxxxxx`  
(Where xxxxxxx is the SCN found in step 1).
6. Shut down the database.
7. Copy the control file from after RESETLOGS to all locations in the CONTROL\_FILES parameter in the **init.ora**.
8. Start up mount the database.
9. Use the appropriate **recover database** command and recover normally, Oracle should now ask for archive log sequence number 1.

#### HINTS:

- Always take a backup of your controlfile after RESETLOGS.
- Always back up all archive logs before attempting this operation.

**Dangers:** It is possible (but not likely) that you may have conflicting archive log sequence numbers ( the same log sequence number from before the RESETLOGS point and after the RESETLOGS). You must be very careful not to overwrite one with the other, especially if they are not backed up.

## 34.4 CASE 10: Creating Data Files

**PROBLEM:** On a Monday afternoon, while running an application, we got an Oracle error saying that there is no more space in a specified tablespace. We then added a data file to that tablespace. Since the application does a lot of DML operations and modifies a lot of tables, he decided to wait until the application finished so that he could take a backup of the data file he just added.

On Friday morning, a media failure occurred, and we lost the new data file we had added on Monday. We then realized that we forgot to take a backup of the data file on Monday. We could take the file offline and start up the database, but we could lose a lot of data that the application had inserted into that data file. We could restore from the backup and roll forward, but this wouldn't work because the backup doesn't have the new data file we added on Monday.

**SOLUTION:** This is a perfect scenario for re-creating the data file. To re-create the data file, we need the current control file or a backup control file that recognizes the new data file that was added on Monday. Next, all the archive log files and online log files need to be available. All we need to do is mount the database and issue the **ALTER DATABASE CREATE DATAFILE** command to re-create the data file. Once the file is created, we need to apply the redo (roll forward) from the time the file was created to the present time.

**TEST:**

```
SQL> ALTER TABLESPACE USERS ADD
 DATAFILE '/disk1/oradata/DEMO/user02.dbf' SIZE 10M;
SQL> INSERT INTO TEST1 VALUES (222,'ABCD');
SQL> ALTER SYSTEM SWITCH LOGFILE;
SQL> ALTER SYSTEM SWITCH LOGFILE;
$ rm /disk1/oradata/DEMO/user02.dbf
Know we have to recover user02.dbf file.
SQL> ALTER DATABASE CREATE DATAFILE '/disk1/oradata/DEMO/user02.dbf';
SQL> RECOVER DATAFILE '/disk1/oradata/DEMO/user02.dbf';
```

**OBSERVATION:** The **alter database create datafile** command expects the use of a control file that has the file entry for the data file to be re-created. It is therefore important to have one of the following:

- The current control file.
- The backup control file that has the file entry for the data file to be re-created – this means you should take a backup of the control file immediately after the schema change.

Also, note in this test that the **ALTER DATABASE CREATE DATAFILE** command actually creates the data file for you at the OS level, and the **RECOVER DATAFILE** command reads the change from the redo log file and applies them to the data blocks.

## 35. Hot Backups

### 35.1 Why Hot Backups?

In some companies we can't afford to bring the Instance down (e.g. Credit Card Companies, Stock Market or where we have 24 hours of DB operations). We have to make the DB available to end-users for 24 hours. At the same time we need to backup the DB to protect our self from any type of disasters. So, Oracle is giving us an excellent option of backing up of DB even while it is up & running, which is known as HOT BACKUP. The recovery procedures are 100% alike in HOT-BACKUP when compared to COLD BACKUP. As we have done 4 types of recovery scenarios:

- Media Recovery (when we have lost everything and trying to build the most-possible from LN-Backup and applying AL-Files An example for In-Complete Recovery)
- File-Lost (Performing Complete Recovery)
- Time-Based Recovery
- Getting back lost objects (E.g. Table dropped by mistakenly)

All the above recoveries that we have performed under Cold Backup are exactly identical in Hot Backup also. So basically the steps we take in Backing up the DB is different in HOT BACKUP compared to Cold Backup, but Recoveries are exactly alike. Rules to implement HOT BACKUP.

1. The database must be in ALM.
2. While doing HB, users can perform reads/write operations on DB since HB it is transparent to users. So that it can be done at any time, even in Lunch hours. This operation may slow down the server's response time.
3. We do HB by TS (each one individually).

```
SQL> ALTER TABLESPACE user_data BEGIN BACKUP;
```

Now, at OS-Level, we have to take backup of all the files belonging to these TS. We cannot take the backup of multiple files that are belonging to these TS in multiple shots. All files must be backed up.

```
SQL> ALTER TABLESPACE user_data END BACKUP;
```

We do HB only for  
Controlfile (not at OS Level, but at SQL Level)  
DBFs (datafiles at OS Level)

We'll ignore REDO-LOG files, since these files are open while the HB is going on. Well, in case of DBFs, we are taking the TS in a special mode, where Oracle is letting you to backup the DBFs at OS Level. But Oracle generates excessive LOG, so it's recommended that we try to keep each TS in backup mode as little time as possible.

Controlfile cannot be backed up as it is since the DB is Up & Running. We do it by SQL (not at OS Level) after finishing with all TSs.

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO
 '/disk1/oradata/ORCL/control.ctl';
```

If we go with HB, we can only do Restore+Recovery. Only Restore is not possible. This leads us to have at least one AF after finishing with HB. So at the end of the HB, we do

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

In point #4, we are talking about keeping each TS in backup-mode as little time as possible. This is not possible in case we try to write all the Datafiles to the TAPE, which belong to this TS. So instead of writing to the TAPE (especially in case of HB), it is highly recommended that we write to another HD.

Likewise, we write all the DBFs to this HD, and at the end of the HB, we go to this HD and copy all files to TAPE in one shot. (Here we understand that writing to HD is much faster compared to TAPE).

The recovery procedures are same (including syntax) compared to CB, excepting the diff. that in CB, simple Restore is possible, while in HB, we have to perform Restore+Recovery which is demanding at least one Archivelog file (after performing Backup on all TSs, and Controlfile. SR is not possible in HB).

From 11g, we can place the full Database itself in begin backup mode using the following command:

```
SQL> ALTER DATABASE BEGIN BACKUP;
```

Once we put DB in Begin Backup mode, we need to backup datafiles followed by controlfile, then perform a switch. **Note:** This is recommended when there are less transactions going on DB. Once we are done with backup we perform:

```
SQL> ALTER DATABASE END BACKUP;
```

## 36. Recovery Manager (RMAN)

### 36.1 What is RMAN?

Recovery Manager is a tool that manages the process of creating backups and also manages the process of restoring and recovering from them. This is more powerful with a redesigned incremental backup scheme, offline recovery of incremental backups; previewing restore, recovering through reset logs, file compression, and much more.

### 36.2 Overview of RMAN Functional Components

The RMAN environment consists of the utilities and databases that play a role in backing up our data. At a minimum, the environment for RMAN must include the following:

- The target database to be backed up
- The RMAN client, which interprets backup and recovery commands, directs server sessions to execute those commands, and records our backup and recovery activity in the target database control file.
- Some environments will also use these optional components:
- A flash recovery area, a disk location in which the DB can store and manage files related to backup and recovery
- Media management software, required for RMAN to interface with backup devices such as tape drives
- A recovery catalog database, a separate database schema used to record RMAN activity against one or more target databases.

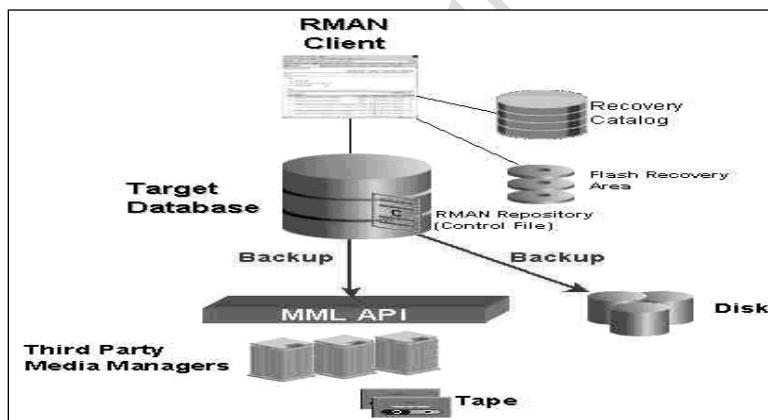


Figure 1.1 - Components of RMAN

#### 36.2.1 Target Database

The target database is the database that we are backing up, restoring, or recovering with RMAN.

#### 36.2.2 RMAN Client

RMAN is a command-line-oriented database client, much like SQL\*Plus, with its own command syntax. From the RMAN client we can issue RMAN commands and SQL statements to perform and report on backup and recovery operations. RMAN can take interactive input or read input from plain text files (called command files).

RMAN then communicates with one or more server processes on the target database server which actually perform the work. We can also access RMAN through the Enterprise Manager. The RMAN executable is typically installed in the same directory as the other database executables. On Unix systems, for example, the RMAN executable is located in \$ORACLE\_HOME/bin.

### 36.2.3 RMAN Repository

RMAN maintains metadata about the target database and its backup and recovery operations in the RMAN repository. Among other things, RMAN stores information about its own configuration settings, the target database schema, archived redo logs, and all backup files on disk or tape. RMAN's LIST, REPORT, and SHOW commands display RMAN repository information.

RMAN repository data is always stored in the control file of the target database. The CONTROL\_FILE\_RECORD\_KEEP\_TIME initialization parameter controls how long backup records are kept in the control file before those records are re-used to hold information about more recent backups. The repository can also be kept in a recovery catalog, a separate database that keeps historical data on backup activities much longer than the control file and preserves backup information if the control file is lost.

### 36.2.4 Flash Recovery Area

The Automatic Disk-Based Backup and Recovery feature simplifies managing disk space and files related to backup and recovery, by managing all backup and recovery related files in a flash recovery area. We set the flash recovery area size and location, using the DB\_RECOVERY\_FILE\_DEST and DB\_RECOVERY\_FILE\_DEST\_SIZE initialization parameters.

We also specify a retention policy that dictates when backups may be discarded. RMAN then manages our backup storage, deleting obsolete backups and backups already copied to tape when space is needed, but keeping as many backups on disk as space permits. This minimizes restores from tape during data recovery operations to shorten restore and recovery times.

### 36.2.5 Recovery Catalog

In addition to RMAN repository records, the recovery catalog can also hold RMAN stored scripts, sequences of RMAN commands for common backup tasks. Centralized storage of scripts in the recovery catalog can be more convenient than working with command files.

### 36.2.6 Media Managers

To access sequential media devices like tape libraries, RMAN uses third-party media management software. A media manager controls these devices during backup and recovery, managing the loading, labeling and unloading of media, among other functions. Oracle Corporation's Backup Solutions Program (BSP) works with vendors to help them produce media management software for their devices.

For enterprises that already use media management software in their enterprise, many of those software products can be directly integrated with RMAN. Contact your media management software vendor for details about whether they participate in the BSP and have an RMAN-compatible media management layer.

## 36.3 Why use RMAN?

- No extra costs ...Its available free
- RMAN introduced in Oracle 8 it has become simpler with newer versions and easier than user managed backups
- Proper security
- We are 100% sure our database has been backed up.
- Its contains detail of the backups taken etc in its central repository Facility for testing validity of backups also commands like crosscheck to check the status of backup.
- Faster backups and restores compared to backups without RMAN. RMAN is the only backup tool which supports incremental backups.
- Parallel operations are supported
- Better querying facility for knowing different details of backup
- No extra redo generated when backup is taken. Compared to online backup without RMAN which results in saving of space in hard disk
- RMAN an intelligent tool

- Maintains repository of backup metadata
- Remembers backup set location
- Knows what need to backed up
- Knows what is required for recovery
- Knows what backup are redundant

## 36.4 General Features & Benefits

Guaranteed, accurate backup and recovery

The Flash Recovery Area, a single directory on disk or ASM disk group, consolidates all recovery-related files, including control, data, archive log, RMAN backup set files; everything needed by RMAN upon recovery can be retrieved from this directory.

With a rapidly dropping price, disk has become a more attractive option for primary backup storage, in addition to faster read/write performance than tape. In addition, Flash Recovery Area can automatically warn administrators of disk capacity issues and obsolete outdated backup sets to reclaim space.

With the automatic channel failover feature, when an error occurs during a disk backup or restore, RMAN will attempt to complete as much of the job as possible, rather than aborting the job. RMAN accomplishes this by utilizing multiple channels, as specified by the CONFIGURE command. This offers a higher level of resiliency when streaming problems arise and better utilization of system resources, versus restarting jobs manually.

During a restore, when RMAN finds corruption in a backup, or finds that a backup cannot be accessed, RMAN will try to restore the file from a different backup. RMAN will try to restore the desired file from all possible backups before returning an error. This is done automatically whenever RMAN restores file(s) during the RESTORE or RECOVER commands, relieving the need to search for valid backups and performing a manual RESTORE or RECOVER.

These features are in addition to the Oracle9i RMAN resumeable backup/restore. In case of a failed backup job or restore, restarting the job will automatically pickup the operation from the last successfully backed up file.

### Automatic block corruption detection and repair

Block Media Recovery allows RMAN to fix a corrupted block (detected on backup) while the data file remains online, and non-affected data continues to be available for selecting and updating. This increases data availability and reduces mean time to recover by selectively restoring and recovering the damaged blocks. Minimal I/O is needed because redo is only applied to damaged blocks.

### Comprehensive reporting

Using special V\$ views, users can retrieve information on all currently executing and completed RMAN backup jobs, as well as details on all backed up files and obsolete backup sets. This output can also be easily viewed in Enterprise Manager, under Backup Set Management.

### Performance-optimized, space-saving operations

RMAN takes advantage of intimate knowledge of Oracle block structures to provide high backup and restore data streaming performance and efficient file compression. By default, when creating backup sets, RMAN backs up only blocks that are in use (or have ever been used) and saves disk space by merging blocks into as few backup pieces as necessary.

1. Binary compression technique reduces backup space usage by 50-75%.
2. Block change tracking for incremental backups allows the server to record and read only the changed blocks, rather than performing full data file scans on every operation, thus dramatically shortening completion time.
3. With the new DURATION option for the RMAN BACKUP command, DBAs can weigh backup performance against system service level requirements. By specifying duration, RMAN will automatically calculate the appropriate backup rate. In addition, DBAs can optionally specify whether backups should minimize time or system load.
4. For recovery, completion time is reduced through the use of the incrementally updated backup feature. This allows incremental backups to be continually rolled into data file image copies, thus eliminating the need to apply incrementals on recovery.

#### 5. Intelligent disk consumption monitoring

New in Oracle Database 11g Release 1, Flash Recovery Area allows administrators to setup notifications on disk space usage and automate obsolescence of expired backup sets, via RMAN client command-line or EM interfaces.

#### 6. Simple and centralized management

Administrators can take advantage of the RMAN client command-line interface, whose commands are written in an OS-independent scripting language, or by using the Enterprise Manager Backup and Recovery Management console. With EM, performing backup job scheduling and recovery operations is completely automated via step-by-step wizards.

In addition, EM includes:

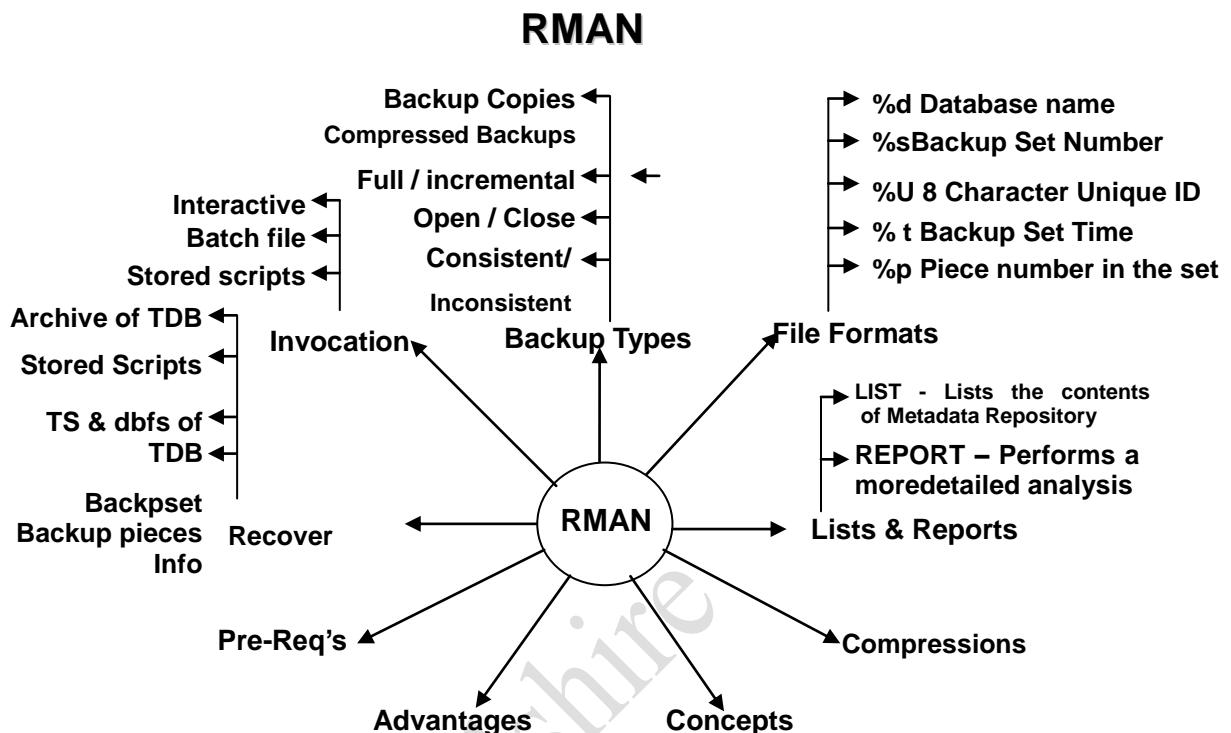
- Retention policy parameters
- Mean-time-to-recovery monitoring
- Archived log multiplexing
- Backup set and image copies management
- Flash recovery area settings
- Catalog synchronization
- One-button crosscheck

#### 7. Fine-granular data operations

RMAN provides fine-granular, on and offline backups at the database, tablespace, archive log, datafile, control file, and block level. Also, both up-to-the-present and point-in-time recoveries can be performed based on a date or SCN number, by rolling forward archived logs. RMAN in Oracle Database 11g Release 1 eliminates the need for a DBA to manually create an auxiliary instance for tablespace point-in-time recovery (TSPITR). RMAN takes care of instance creation on the same server as the target database, and removal upon completion of the tablespace recovery.

#### 8. Extensible to third party media managers

Through one standard Media Management Layer API (MML), third-party media management vendors can leverage the functionality of RMAN to provide robust backup solutions for Oracle databases.



## 37. Recovery Manager

### 37.1 Configure

To configure persistent settings for RMAN Backup, Restore, Duplication, and Maintenance jobs. These configurations are in effect for RMAN session until the configuration is cleared or changed. Use CONFIGURE to set the following:

- An ongoing retention policy that automatically determines which backups and copies are eligible for deletion because they are no longer needed
- The device type (for example, DISK or TAPE) for RMAN jobs.
- Default no. of channels each device type that RMAN should allocate for automated Backup & Restore jobs
- The settings for automatic channels for a specified device type.
- The maximum size of backup pieces and sets created on automatic channels
- Backup optimization either ON or OFF
- The exclusion policy for tablespaces in whole database backups
- The filename of the snapshot control file
- The control file auto backup feature to ON or OFF
- The default format for the control file auto backup output files

RMAN uses default settings for CONFIGURE options. You can return to default value for any CONFIGURE command by running the same command with the CLEAR option.

#### 37.1.2 Restrictions and usage notes

- Execute this command at the RMAN prompt and the target database must be mounted or open.
- It doesn't simultaneously allocate automatic channels for multiple device types in Backup and Copy jobs.
- To direct backups or restores to specific channels, use the RMAN generated channel names. If you specify channel numbers in the CONFIGURE CHANNEL command, the RMAN uses the same numbers in the system generated channel names.
- You cannot exclude the SYSTEM tablespace from whole database backups
- The REDUNDANCY and RECOVERY WINDOW options are mutually exclusive. Only one type of retention policy can be in effect at any time.
- The channel number is a manually numbered channel must be less than 255
- You must specify at least one channel option when running CONFIGURE CHANNEL, in other words, you cannot issue a command such as CONFIGURE CHANNEL 2 DEVICE TYPE DISK, but you can issue a command such as CONFIGURE CHANNEL 2 DEVICE TYPE DISK MAXPIECESIZE 2500K
- The CONFIGURE CONTROLFILE AUTOBACKUP FORMAT string must include %F substitution variable.

#### 37.1.3 Comparisons

| Features                  | RMAN                                                     | HB/CB                                                              | Logical (exp/imp)           |
|---------------------------|----------------------------------------------------------|--------------------------------------------------------------------|-----------------------------|
| Backup (Database is up)   | Yes (no need for taking tablespace to begin backup mode) | HB only (but generates lot of redologs when a tablespace is in BB) | Yes                         |
| Backup (Database is down) | Yes (but the database must be in a mounted stage)        | CB only                                                            | No                          |
| Backups Incremental       | Yes (only modified blocks will be backed up)             | No                                                                 | Yes (but the unit is table) |

|                            |                                                                           |     |                                                            |
|----------------------------|---------------------------------------------------------------------------|-----|------------------------------------------------------------|
| Backup against tape        | Yes (but needs media manager)                                             | Yes | Yes                                                        |
| O/S independent language   | Yes                                                                       | No  | Yes                                                        |
| Identifying corrupt blocks | Yes (bad blocks will be written to a table called v\$backup_crrorruption) | No  | Yes (bad blocks information will be written to a log file) |

### 37.1.4 Examples

**Configuring Backup Optimization:** This example configures RMAN so that the BACKUP command does not back up files to a device type if the identical file has already been backed up to the device type:

```
CONFIGURE BACKUP OPTIMIZATION ON;
```

**Configuring a Retention Policy:** This example configures a retention policy with a recovery window of 2 weeks, and then resets the retention policy to REDUNDANCY=1;

```
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 14 DAYS;
CONFIGURE RETENTION POLICY CLEAR;
```

**Configuring Automatic Disk and Tape Channels:** This example configures generic DISK and SBT channels, sets the default device type to SBT, and sets PARALLELISM to 3:

```
CONFIGURE CHANNEL DEVICE TYPE DISK RATE 5M;
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
CONFIGURE DEVICE TYPE sbt PARALLELISM 3;
```

**Overriding the Default Device Type:** This example configures the default device type to sbt, backs up the archived logs on the default SBT channel, and then backs up the database to disk on the default disk channel:

```
CONFIGURE DEFAULT DEVICE TYPE TO sbt;
BACKUP ARCHIVELOG ALL;
BACKUP DEVICE TYPE DISK DATABASE;
```

**Configuring Automatic Channels Across File Systems:** This example configures automatic disk channels across three file systems:

```
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '/disk1/oradata/ORCL/bkp/%U';
CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT '/disk2/oradata/ORCL/bkp/%U';
CONFIGURE CHANNEL 3 DEVICE TYPE DISK FORMAT '/disk3/oradata/ORCL/bkp/%U';
```

**Clearing Automatic Channels:** This example clears manually numbered DISK channels 2 and 3 and the generic sbt channel:

```
CONFIGURE CHANNEL 2 DEVICE TYPE DISK CLEAR;
CONFIGURE CHANNEL 3 DEVICE TYPE DISK CLEAR;
CONFIGURE CHANNEL DEVICE TYPE sbt CLEAR;
```

**Configuring and Clearing Parallelism:** This example sets DISK parallelism to 2, then changes it to 3, and then returns it to the default parallelism of 1;

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 2;
CONFIGURE DEVICE TYPE DISK PARALLELISM 3;
CONFIGURE DEVICE TYPE DISK CLEAR;
```

**Configuring Backup Copies:** This example configures duplexing to 3 for DISK backups of data files and control files and then runs a database backup;

```
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 3;
BACKUP DATABASE FORMAT '/fs1/%F', '/fs2/%F', '/fs3/%F';
```

**Configuring the Snapshot Control File Location:** This example configures a new location for the snapshot control file and then resynchronizes the recovery catalog.

```
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/disk1/oradata/bkp/snap.cf';
```

**Excluding a Tablespace from a Whole Database Backup:** This example excludes the read-only reports tablespace from whole DB backups, and then returns the reports tablespace to the default behavior of 'not excluded':

```
CONFIGURE EXCLUDE FOR TABLESPACE reports;
CONFIGURE EXCLUDE FOR TABLESPACE reports CLEAR;
```

**Specifying the Default Format for the Control File Autobackup:** This example turns on the autobackup feature, then changes the default format for the DISK and sbt devices, and then clears the autobackup settings:

```
CONFIGURE CONTROLFILE AUTOBACKUP ON;
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO
 '/disk1/oradata/bkp/%F';
CONFIGURE CONTROLFILE AUTOBACKUP CLEAR; -- returns to default setting
 of OFF
```

## 37.2 Crosscheck

The CROSSCHECK command checks only objects marked AVAILABLE or EXPIRED, either by examining the files on disk for DISK channels or by querying the media manager for sbt CHANNELS. THE crosscheck command only processes files created on the same device type as the channels running the crosscheck. RMAN does not delete any files that it is unable to find, but updates their repository records to EXPIRED.

You can determine which files are marked EXPIRED by issuing a LIST EXPIRED command. Then, you can run DELETE EXPIRED to remove the repository records for all expired files.

```
SQL> ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
SQL> CROSSCHECK BACKUP;
SQL> CROSSCHECK COPY;
```

### Crosschecking within a Range of Dates:

```
SQL> ALLOCATE CHANNEL FOR MAINTENANCE DEVICE TYPE sbt;
SQL> CROSSCHECK BACKUP DEVICE TYPE sbt COMPLETED BETWEEN '01-JAN-00' AND
 '00-JUL-00' ;
```

## 37.3 Delete

To delete physical backups and copies as well as do the following:

- Update their records in the target control file to status DELETED
- Remove their records from the recovery catalog ( if you use a catalog )

By default, DELETE displays a list of the files and prompts you for confirmation before deleting any file in the list, unless you are running a command line. If you specify the EXPIRED option, then DELETE only removes files marked EXPIRED, that is, "not found," by the CROSSCHECK command. If you specify the OBSOLETE option, then DELETE removes files considered OBSOLETE, that is, "not needed," by the retention policy or because it is orphaned. You can specify a retention policy by using CONFIGURE RETENTION POLICY, or specify the REDUNDANCY and RECOVERY WINDOW options in the DELETE command.

**Deleting Expired Backups:** it configures sbt channel to check the media manager for expired backups of the tablespace user\_data that are more than one month old and removes their catalog records:

```
CONFIGURE CHANNEL DEVICE TYPE sbt;
CROSSCHECK BACKUP OF TABLESPACE user_data COMPLETED BEFORE 'SYSDATE -
31';
DELETE NOPROMPT DXPIRED BACKUP OF TABLESPACE user_data COMPLETED
BEFORE 'SYSDATE - 31';
```

**Deleting Obsolete Backups:** It deletes backup copies that are needed to recover the database to a random point with in the week. RMAN also deletes archived redo logs that are no longer needed:

```
DELETE NOPROMPT OBSOLETE RECOVERY WINDOW OF 7 DAYS;
```

**Deleting a Backup Set Specified by Primary Key:** The following example deletes backup set 503 from disk:

```
DELETE BACKUPSET 503;
```

## 37.4 Drop Catalog

This command deletes all metadata from the recovery catalog. If you have no backups of the catalog, then all backups of all DB managed by this recovery catalog become unusable. Enter the command twice to confirm that you want to drop the schema.

```
DROP CATALOG;
DROP CATALOG;
```

## 37.5 Reset Database

To reset the target database in the RMAN repository, which means to do either of the following actions:

- Inform RMAN that the SQL statement ALTER DATABASE OPEN RESETLOGS has been executed and that a new incarnation of the target database has been created. By resetting the database, RMAN considers the new incarnation as the current incarnation of the database.
- To reset the database to a previous incarnation. Typically, you would reset the incarnation when performing incomplete recovery to a point before RESETLOGS operation or when attempting to undo the affects of a RESETLOGS operation, or when attempting to undo the affects of a RESETLOGS by restoring backups taken before a RESETLOGS.

```
RESET DATABASE TO INCARNATION;
Performing Regular RMAN Backups:
RMAN> BACKUP DATABASE;
Performing Compressed RMAN Backups:
RMAN> BACKUP AS COMPRESSED BACKUPSET DATABASE;
Performing Backups as Image copies:
RMAN> BACKUP AS COPY DATABASE;
Enabling Block Change Tracking:
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING USING
FILE '/disk1/oradata/ORCL/blk_cng_trk.dbf';
To check whether Block Change Tracking is enabled, query
V$BLOCK_CHANGE_TRACKING. Similarly you can disable Block Change Tracking
using the command
SQL> ALTER DATABASE DISABLE BLOCK CHANGE TRACKING;
```

### Performing regular Incremental Backups:

```
RMAN> BACKUP INCREMENTAL LEVEL 0 DATABASE; (Complete Backup)
RMAN> BACKUP INCREMENTAL LEVEL 1 DATABASE; (Cumulative Backup)
RMAN> BACKUP INCREMENTAL LEVEL 2 DATABASE; (Incremental Backup)
```

**Performing Compressed Incremental Backups:**

```
RMAN> BACKUP AS COMPRESSED BACKUPSET INCREMENTAL LEVEL 0 DATABASE;
(Complete Backup)
RMAN> BACKUP AS COMPRESSED BACKUPSET INCREMENTAL LEVEL 1 DATABASE;
(Cumulative Backup)
RMAN> BACKUP AS COMPRESSED BACKUPSET INCREMENTAL LEVEL 2 DATABASE;
(Incremental Backup)
```

**Using Flash Recovery Area for RMAN Backups:**

All files needed to recover a Database from a Media failure or a logical error is contained in the Flash Recovery Area. Flash Recovery Area directory structure is used by RMAN in a very organized fashion with separate directories for each file type such as Archive logs, Backup Sets, Image copies, Control file auto backup. For enabling Flash Recovery Area for an instance include the following parameters in init.ora. These Parameters are Dynamic.

```
DB_RECOVERY_FILE_DEST = /disk1/oradata/ORCL/flash
DB_RECOVERY_FILE_DEST_SIZE = 1000M
```

## 38. RMAN Enhancements in Oracle Database 10g

Oracle 10g includes many RMAN enhancements making it a more complete tool, including:

- Flash Recovery Area
- Incrementally Updated Backups
- Fast Incremental Backups
- BACKUP for Backupsets and Image Copies
- Cataloging Backup Pieces
- Improved RMAN Reporting Through V\$ Views
- Automatic Instance Creation for RMAN TSPITR
- Cross-Platform Tablespace Conversion
- Enhanced Stored Scripts Commands
- Backupset Compression
- Restore Preview
- Managing Backup Duration and Throttling

### Miscellaneous

- Disk Topology and Automatic Performance Tuning
- Automatic Datafile Creation
- Proxy Archived Log Backups
- Recovery Through Restelogs
- Restore Failover
- Channel Failover
- Deferred Error Reporting

### 38.1 Flash Recovery Area

Flash Recovery Area is a location on filesystem or on ASM disk group that holds files related to recovery including:

- Multiplexed controlfiles
- Multiplexed online redo logs
- Archived redo logs
- Flashback logs
- RMAN disk backups
- Files created by RESTORE and RECOVERY commands.

Space within the flash recovery area is managed by the database. If there is not enough space to complete an operation obsolete, backed up or redundant files are removed to free up some space.

The following example shows the parameters used to configure the flash recovery area:

```
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST =
 '/user01/app/oracle/flash_recovery_area';
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST_SIZE = 2G;
SQL> ALTER SYSTEM SET DB_FLASHBACK_RETENTION_TARGET = 1440;
```

### 38.2 Incrementally Updated Backups

Using this feature all changes between the SCN of the original image copy and the SCN of the incremental backup are applied to the image copy, winding it forward to make the equivalent of a new database image copy without the overhead of such a backup. The following example shows how this can be used:

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

The RECOVER COPY... line will not do anything until the script has been running for more than 7 days. The BACKUP INCREMENTAL line will perform a complete backup (level 0) the first day it is run, with all subsequent backups being level 1 incremental backup.

After 7 days, the RECOVER COPY... line will start to take effect, merging all incremental backups older than 7 days into the level 0 backup, effectively moving the level 0 backup forward. The effect of this is that you will permanently have a 7 day recovery window with a 7 day old level 0 backup and 6 level 1 incremental backups. Notice that the tag must be used to identify which incremental backups apply to which image copies.

If you wanted to keep your image copy as up to date as possible you might do the following:

```
RUN {
 BACKUP INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH
 TAG 'incr_backup' DATABASE;
 RECOVER COPY OF DATABASE WITH TAG 'incr_backup';
}
```

In this example the incremental backup is merged into the image copy as soon as it is completed.

### 38.3 Fast Incremental Backups

There is performance issues associated with incremental backups as the whole of each datafile must be scanned to identify changed blocks. In Oracle 10g it is possible to track changed blocks using a change tracking file.

Enabling change tracking does produce a small overhead, but it greatly improves the performance of incremental backups. The current change tracking status can be displayed using the following query:

```
SQL> SELECT status FROM v$block_change_tracking;
```

Change tracking is enabled using the ALTER DATABASE command:

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

By default the change tracking file is created as an Oracle Managed File (OMF) in the location pointed to by the DB\_CREATE\_FILE\_DEST parameter. An alternate location can be specified using the following command:

```
SQL> ALTER DATABASE ENABLE BLOCK CHANGE TRACKING
 USING FILE '/u01/oradata/MYSID/rman_change_track.f' REUSE;
```

The tracking file is created with a minimum size of 10M and grows in 10M increments. Its size is typically 1/30,000 the size of the data blocks to be tracked.

Change tracking can be disabled using the following command:

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

Renaming or moving a tracking file can be accomplished in the normal way using the ALTER DATABASE RENAME FILE command. If the instance cannot be restarted you can simply disable and re-enable change tracking to create a new file. This method does result in the loss of any current change information.

### 38.4 BACKUP for Backupsets and Image Copies

In Oracle 11g the BACKUP command has been extended to allow it to initiate backups of image copies in addition to backupsets. As a result the COPY command has been deprecated in favour of this new syntax.

```
SQL> BACKUP AS COPY DATABASE;
SQL> BACKUP AS COPY TABLESPACE users;
SQL> BACKUP AS COPY DATAFILE 1;
```

RMAN supports the creation of image copies of datafiles and datafile copies, control files and controlfile copies, archived redo logs, and backup pieces.

## 38.5 Cataloging Backup Pieces

It's now possible to manually catalog backup piece using the CATALOG commands in RMAN. This allows backup files to be moved to alternate locations or manually archived to tape and brought back for restore operations. In Oracle 9i this functionality was only available for controlfile copies, Archivelog copies and datafile copies. In addition, there are some shortcuts to allow multiple files to be cataloged using a single command. This example gives general idea:

```

Catalog specific backup piece.
CATALOG BACKUPPIECE '/backup/MYSID/01dmsbj4_1_1.bcp';

Catalog all files and the contents of directories which
begin with the pattern "/backup/MYSID/arch".
CATALOG START WITH '/backup/MYSID/arch';

Catalog all files in the current recovery area.
CATALOG RECOVERY AREA NOPROMPT;

Catalog all files in the current recovery area.
This is an exact synonym of the previous command.
CATALOG DB_RECOVERY_FILE_DEST NOPROMPT;
The NOPROMPT clause suppresses user confirmation for all matching files.

```

## 38.6 Improved RMAN Reporting Through V\$ Views

Oracle 11g includes additional V\$ views making the reporting of backup operations more transparent.

- V\$RMAN\_OUTPUT - This is an in-memory view of the messages reported by RMAN holding a maximum of 32767 rows. Since this information is not recorded in the controlfile it is lost on instance restart.
- V\$RMAN\_STATUS - This view displays progress and status information for in-progress and complete RMAN jobs. The information for the in-progress jobs is memory only, while the complete job information comes from the controlfile.
- V\$BACKUP\_FILES - This view display information about RMAN image copies, backupsets and archived logs, similar to the information listed by the RMAN commands LIST BACKUP and LIST COPY. This view relies on the DBMS\_RCMAN.SETDATABASE procedure being run to set the database.

The V\$RMAN\_CONFIGURATION view from Oracle 9i is still available in Oracle 11g.

## 38.7 Automatic Instance Creation for RMAN TSPITR

If a tablespace point-in-time recovery (TSPITR) is initiated with no reference to an auxiliary instance RMAN now automatically creates a one. The auxiliary instance configuration is based on that of the target database.

As a result, any channels required for the restore operations must be present in the target database so they are configured correctly in the auxiliary instance. The location of the datafiles for the auxiliary instance are specified using the AUXILIARY DESTINATION clause shown below.

```
SQL> RECOVER TABLESPACE users UNTIL LOGSEQ 2400 THREAD 1
 AUXILIARY DESTINATION '/u01/oradata/auxdest';
```

The TS is taken offline, restored from a backup, recovered to the specified point-in-time in the auxiliary instance and re-imported into the target DB. The TS in the target DB should then be backed up and the TS brought back online.

```
SQL> BACKUP TABLESPACE users;
SQL> SQL "ALTER TABLESPACE users ONLINE";
```

On successful completion the auxiliary instance will be cleaned up automatically. In the event of errors the auxiliary instance is left intact to aid troubleshooting.

## 38.8 Cross-Platform Tablespace Conversion

The CONVERT TABLESPACE allows TS to be transported between platforms with different byte orders. The mechanism for transporting TS is unchanged; this command merely converts TS to allow the transport to work. The platform of the source and destination platforms can be identified using the V\$TRANSPORTABLE\_PLATFORM view. The platform of the local server is not listed as no conversion is necessary for a matching platform.

```
SQL> SELECT platform_name FROM v$transportable_platform;
PLATFROM_NAME

Solaris[tm] OE (32-bit)
...
...
Microsoft Windows 64-bit for AMD
15 rows selected.
```

The tablespace conversion can take place on either the source or the destination server. The following examples show how the command is used in each case:

# Conversion on a Solaris source host to a Linux destination file.

```
CONVERT TABLESPACE my_tablespace TO PLATFORM 'Linux IA (32-bit)'
FORMAT='/tmp/transport_linux/%U';
```

# Conversion on a Linux destination host from a Solaris source file.

```
CONVERT DATAFILE='/tmp/transport_solaris/my_ts_file01.dbf',
'/tmp/transport_solaris/my_ts_file02.dbf'
FROM PLATFORM 'Solaris[tm] OE (32-bit)'
DB_FILE_NAME_CONVERT '/tmp/transport_solaris','/u01/oradata/MYDB';
```

In the first example the converted files are placed in the directory specified by the FORMAT clause. In the second example the specified datafiles are converted to the local server's platform and placed in the correct directory specified by the DB\_FILE\_NAME\_CONVERT clause.

## 38.9 Enhanced Stored Scripts Commands

Scripts can now be defined as global allowing them to be accessed by all databases within the recovery catalog. The syntax for global script manipulation is the same as that for regular scripts with the addition of the GLOBAL clause prior the word SCRIPT. Examples of its usage are shown below:

```
CREATE GLOBAL SCRIPT full_backup
{
 BACKUP DATABASE PLUS ARCHIVELOG;
 DELETE FORCE NOPROMPT OBSOLETE;
}
CREATE GLOBAL SCRIPT full_backup FROM FILE 'full_backup.txt';
RUN { EXECUTE GLOBAL SCRIPT full_backup; }
PRINT GLOBAL SCRIPT full_backup;
LIST GLOBAL SCRIPT NAMES;
LIST ALL SCRIPT NAMES; # Global and local scripts.
REPLACE GLOBAL SCRIPT full_backup
{
 BACKUP DATABASE PLUS ARCHIVELOG;
 DELETE FORCE NOPROMPT OBSOLETE;
}
REPLACE GLOBAL SCRIPT full_backup FROM FILE 'full_backup.txt';
DELETE GLOBAL SCRIPT 'full_backup';
```

## 38.10 Backupset Compression

The AS COMPRESSED BACKUPSET option of BACKUP command allows RMAN to perform binary compression of backupsets. The resulting backupsets don't need to be uncompressed during recovery. It is most useful in the following circumstances:

```
When we are performing disk-based backup with limited disk space.
When we are performing backups across a network where network bandwidth
is limiting.
When we are performing backups to tape, CD or DVD where hardware
compression is not available.
The following examples assume that some persistent parameters are
configured in a similar manner to those listed below:
SQL> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
SQL> CONFIGURE DEFAULT DEVICE TYPE TO DISK;
SQL> CONFIGURE CONTROLFILE AUTOBACKUP ON;
SQL> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT
 '/backups/MYSID/%d_DB_%u_%s_%p';
```

The AS COMPRESSED BACKUPSET option can be used explicitly in the backup command:

```
Whole database and archivelogs.
BACKUP AS COMPRESSED BACKUPSET DATABASE PLUS ARCHIVELOG;
Datafiles 1 and 5 only.
BACKUP AS COMPRESSED BACKUPSET DATAFILE 1,5;
Alternatively the option can be defined using the CONFIGURE command:
Configure compression.
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO COMPRESSED
BACKUPSET;
Whole database and archivelogs.
BACKUP DATABASE PLUS ARCHIVELOG;
```

Compression requires additional CPU cycles which may affect the performance of the database. For this reason it should not be used for tape backups where hardware compression is available.

## 38.11 Restore Preview

The PREVIEW option of the RESTORE command allows you to identify the backups required to complete a specific restore operation. The output generated by the command is in the same format as the LIST command. In addition the PREVIEW SUMMARY command can be used to produce a summary report with the same format as the LIST SUMMARY command. The following examples show how these commands are used:

```
Preview
RESTORE DATABASE PREVIEW;
RESTORE TABLESPACE users PREVIEW;

Preview Summary
RESTORE DATABASE PREVIEW SUMMARY;
RESTORE TABLESPACE users PREVIEW SUMMARY;
```

## 38.12 Managing Backup Duration and Throttling

The DURATION clause of the BACKUP command restricts the total time available for a backup to complete. At the end of the time window backup is interrupted with any incomplete backupsets discarded. All complete backupsets are kept and used for future restore operations. The following examples show how it is used:

```
SQL> BACKUP DURATION 2:00 TABLESPACE users;
SQL> BACKUP DURATION 5:00 DATABASE PLUS ARCHIVELOGS;
```

## 38.13 RMAN Tablespace Point-in-Time Recovery (TSPITR)

Recovery Manager (RMAN) automatic tablespace point-in-time recovery (commonly abbreviated TSPITR) enables us to quickly recover one or more tablespaces in an Oracle database to an earlier time, without affecting the state of the rest of the tablespaces and other objects in the database.

This chapter explains when we can and cannot use TSPITR, what RMAN actually does to our database during TSPITR, how to prepare a database for TSPITR, how to run TSPITR, and options for controlling the TSPITR process.

## 38.14 Understanding RMAN TSPITR

In order to use TSPITR effectively, we need to understand what problems it can solve for us, what the major elements used in TSPITR are, what RMAN does during TSPITR, and limitations on when and how it can be applied.

### 38.14.1 RMAN TSPITR Concepts

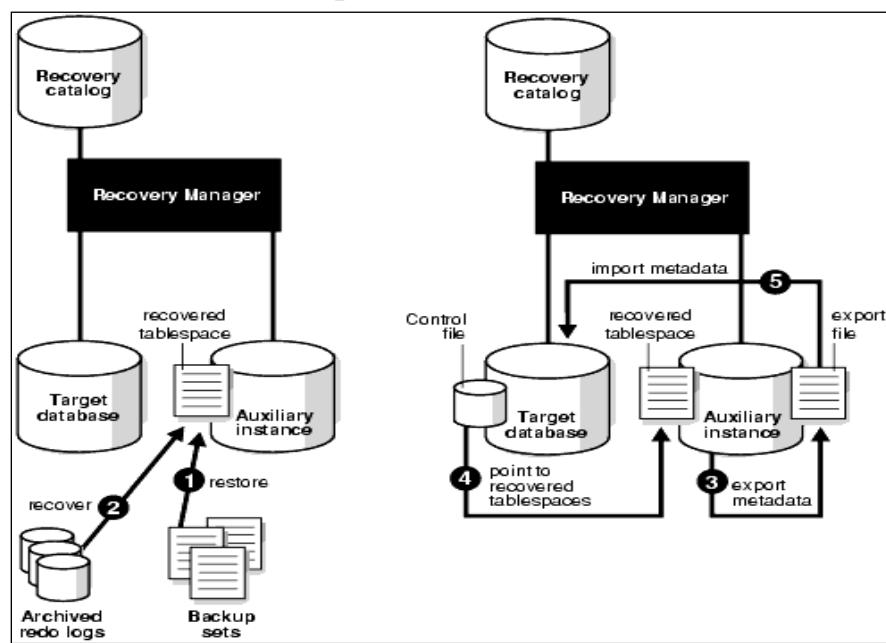


Figure 1 - Tablespace Point-in-Time Recovery (TSPITR) Architecture

The above figure contains the following entities:

- The target instance, containing the tablespace to be recovered
- The Recovery Manager client
- The control file and (optional) recovery catalog, used for the RMAN repository records of backup activity
- Archived redo logs and backup sets from the target database, which are the source of the reconstructed tablespace.
- The auxiliary instance, an Oracle database instance used in the recovery process to perform the actual work of recovery.

There are four other important terms related to TSPITR, which will be used in the rest of this discussion:

The target time, the point in time or SCN that the tablespace will be left at after TSPITR

The recovery set, which consists of the datafiles containing the tablespaces to be recovered;

The auxiliary set, which includes datafiles required for TSPITR of the recovery set which are not themselves part of the recovery set. The auxiliary set typically includes:

- A copy of the SYSTEM tablespace
- Datafiles containing rollback or undo segments from the target instance
- In some cases, a temporary tablespace, used during the export of database objects from the auxiliary instance

The auxiliary instance has other files associated with it, such as a control file, parameter file, and online logs , but they are not part of the auxiliary set.

The auxiliary destination, an optional location on disk which can be used to store any of the auxiliary set datafiles, control files and online logs of the auxiliary instance during TSPITR. Files stored here can be deleted after TSPITR is complete.

### 38.14.2 How TSPITR Works With an RMAN-Managed Auxiliary Instance

To perform TSPITR of the recovery set using RMAN and an automated auxiliary instance, we carry out the preparations for TSPITR described in "Planning and Preparing for TSPITR", and then issue the RECOVER TABLESPACE command, specifying, at a minimum, the tablespaces of the recovery set and the target time for the point-in-time recovery, and, if desired, an auxiliary destination as well.

RMAN then carries out the following steps:

1. If there is no connection to an auxiliary instance, RMAN creates the auxiliary instance, starts it up and connects to it.
2. Takes the tablespaces to be recovered offline in the target database
3. Restores a backup controlfile from a point in time before the target time to the auxiliary instance
4. Restores the datafiles from the recovery set and the auxiliary set to the auxiliary instance. Files are restored either in locations we specify for each file, or the original location of the file (for recovery set files) or in the auxiliary destination (for auxiliary set files, if we used the AUXILIARY DESTINATION argument of RECOVER TABLESPACE)
5. Recovers the restored datafiles in the auxiliary instance to the specified time
6. Opens the auxiliary database with the RESETLOGS option
7. Exports the dictionary metadata about objects in the recovered tablespaces to the target database
8. Shuts down the auxiliary instance
9. Issues SWITCH commands on the target instance, so that the target database control file now points to the datafiles in the recovery set that were just recovered at the auxiliary instance.
10. Imports the dictionary metadata from the auxiliary instance to the target instance, allowing the recovered objects to be accessed.
11. Deletes all auxiliary set files.

At that point the TSPITR process is complete. The recovery set datafiles are returned to their contents at the specified point in time, and belong to the target database.

### 38.14.3 Deciding When to Use TSPITR

Like a table import, RMAN TSPITR enables us to recover a consistent data set; however, the data set recovered includes an entire tablespace rather than one object.

RMAN TSPITR is most useful for situations such as these:

Recovering data lost after an erroneous TRUNCATE TABLE statement;

Recovering from logical corruption of a table;

Undoing the effects of an incorrect batch job or other DML statement that has affected only a subset of the database;

Recovering a logical schema to a point different from the rest of the physical database, when multiple schemas exist in separate tablespaces of one physical database.

Note that, as with database point-in-time recovery (DBPITR), we cannot perform TSPITR if we do not have our archived redo logs. For databases running in NOARCHIVELOG mode, we cannot perform TSPITR.

### 38.14.4 Limitations of TSPITR

There are a number of situations which we cannot resolve by using TSPITR.

We cannot recover dropped tablespaces.

We cannot recover a renamed tablespace to a point in time before it was renamed. If we try to perform a TSPITR to an SCN earlier than the rename operation, RMAN cannot find the new tablespace name in the repository as of that earlier SCN (because the tablespace did not have that name at that SCN).

In this situation, we must recover the entire database to a point in time before the tablespace was renamed. The tablespace will be found under the name it had at that earlier time.

We cannot recover tables without their associated constraints, or constraints without the associated tables.

We cannot use TSPITR to recover any of the following:

- Replicated master tables
- Partial tables (for example, if we perform RMAN TSPITR on partitioned tables and spread partitions across multiple tablespaces, then we must recover all tablespaces which include partitions of the table.)
- Tables with VARRAY columns, nested tables, or external files
- Snapshot logs and snapshot tables
- Tablespaces containing undo or rollback segments
- Tablespaces that contain objects owned by SYS, including rollback segments

#### TSPITR has some other limitations:

- If a datafile was added after the point to which RMAN is recovering, an empty datafile by the same name will be included in the tablespace after RMAN TSPITR.
- TSPITR will not recover query optimizer statistics for recovered objects. We must gather new statistics after the TSPITR.
- Assume that we run TSPITR on a tablespace, and then bring the tablespace online at time t. Backups of the tablespace created before time t are no longer usable for recovery with a current control file. We cannot run TSPITR again on this tablespace to recover it to any time less than or equal to time t, nor can we use the current control file to recover the database to any time less than or equal to t. Therefore, we must back up the tablespace as soon as TSPITR is complete.

### 38.14.5 Limitations of TSPITR Without a Recovery Catalog

If we do not use a recovery catalog when performing TSPITR, then note the following special restrictions:

- The undo segments at the time of the TSPITR must be part of the auxiliary set. Because RMAN has no historical record of the undo in the control file, RMAN assumes that the current rollback or undo segments were the same segments present at the time to which recovery is performed. If the undo segments have changed since that time, then TSPITR will fail.
- TSPITR to a time that is too old may not succeed if Oracle has reused the control file records for needed backups. (In planning our database, set the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter to a value large enough to ensure that control file records needed for TSPITR are kept.)
- When not using a recovery catalog, the current control file has no record of the older incarnation of the recovered tablespace. Thus, recovery with a current control file that involves this tablespace can no longer use a backup taken prior to time  $t$ . We can, however, perform incomplete recovery of the whole database to any time less than or equal to  $t$ , if we can restore a backup control file from before time  $t$ .

## 38.15 Performing Basic RMAN TSPITR

Having selected our tablespaces to recover and our target time, we are now ready to perform RMAN TSPITR. We have a few different options available to us:

- Fully automated TSPITR--in which we specify an auxiliary destination and let RMAN manage all aspects of the TSPITR. This is the simplest way to perform TSPITR, and is recommended unless we specifically need more control over the location of recovery set files after TSPITR or auxiliary set files during TSPITR, or control over the channel configurations or some other aspect of our auxiliary instance.
- Customized TSPITR with an automatic auxiliary instance--in which we base our TSPITR on the behavior of fully automated TSPITR, possibly still using an auxiliary destination, but customize one or more aspects of the behavior, such as the location of auxiliary set or recovery set files, or specifying initialization parameters or channel configurations for the auxiliary instance created and managed by RMAN.
- TSPITR with our own auxiliary instance--in which we take responsibility for setting up, starting, stopping and cleaning up the auxiliary instance used in TSPITR, and possibly also manages the TSPITR process using some of the methods available in customized TSPITR with an automatic auxiliary instance.

### 38.15.1 Fully Automated RMAN TSPITR

When performing fully automated TSPITR, letting RMAN manage the entire process, there are only two requirements beyond the preparations in "Planning and Preparing for TSPITR":

We must specify the auxiliary destination for RMAN to use for the auxiliary set datafiles and other files for the auxiliary instance.

We must configure any channels required for the TSPITR on the target instance.

(The auxiliary instance will use the same channel configuration as the target instance when performing the TSPITR.)

RMAN bases as much of the configuration for TSPITR as possible on our target database. During TSPITR, the recovery set datafiles are written in their current locations on the target database. The same channel configurations in effect on the target database are used on the auxiliary instance when restoring files from backup. Auxiliary set datafiles and other auxiliary instance files, however, are stored in the auxiliary destination.

#### 38.15.1.1 Using an Auxiliary Destination

Oracle Corporation recommends that we use an auxiliary destination with our auxiliary instance. Even if we use other methods to rename some or all of the auxiliary set datafiles, specifying an `AUXILIARY DESTINATION` parameter provides a default location for auxiliary set datafiles for which names are not specified. This way, TSPITR will not fail if we inadvertently do not provide

names for all auxiliary set datafiles.

To specify an auxiliary destination, find a location on disk where there is enough space to hold our auxiliary set datafiles. Then, use the AUXILIARY DESTINATION parameter in our RECOVER TABLESPACE command to specify the auxiliary destination location, as shown in the next section.

### 38.15.1.2 Performing Fully Automated RMAN TSPITR

To actually perform automated RMAN TSPITR, start the RMAN client, connecting to the target database and, if applicable, a recovery catalog. This example shows connecting in NOCATALOG mode, using operating system authentication:

```
% rman TARGET /
```

Note: Do not connect to an auxiliary instance when starting the RMAN client for automated TSPITR. If there is no connected auxiliary instance, RMAN constructs the automatic auxiliary instance for us when carrying out the RECOVER TABLESPACE command. (If there is a connected auxiliary instance, RMAN will assume that we are trying to manage our own auxiliary instance, and try to use the connected auxiliary for TSPITR.)

If we have configured channels that RMAN can use to restore from backup on the primary instance, then we are ready to perform TSPITR now, by running the RECOVER TABLESPACE... UNTIL... command.

This example returns the users and tools tablespaces to the end of log sequence number 1300, and stores the auxiliary instance files (including auxiliary set datafiles) in the destination /disk1/auxdest:

```
RMAN> RECOVER TABLESPACE users, tools
 UNTIL LOGSEQ 1300 THREAD 1
 AUXILIARY DESTINATION '/disk1/auxdest';
```

Assuming the TSPITR process completes without error, the tablespaces are taken offline by RMAN, restored from backup and recovered to the desired point in time on the auxiliary instance, and then re-imported to the target database. The tablespaces are left offline at the end of the process. All auxiliary set datafiles and other auxiliary instance files are cleaned up from the auxiliary destination.

### 38.15.1.3 Tasks to Perform After Successful TSPITR

If TSPITR completes successfully, we must back up the recovered tablespaces, and then we can bring them online.

#### Backing Up Recovered Tablespaces After TSPITR

It is very important that we backup recovered tablespaces immediately after TSPITR is completed.

After we perform TSPITR on a tablespace, we cannot use backups of that tablespace from before the TSPITR was completed and the tablespace put back on line. If we start using the recovered tablespaces without taking a backup, we are running our database without a usable backup of those tablespaces. For this example, the users and tools tablespaces must be backed up, as follows:

```
RMAN> BACKUP TABLESPACE users, tools;
```

We can then safely bring the tablespaces online, as follows:

```
RMAN> SQL "ALTER TABLESPACE users, tools ONLINE";
```

Our recovered tablespaces are now ready for use.

### Handling Errors in Automated TSPITR

In the event of an error during automated TSPITR, we should refer to "Troubleshooting RMAN TSPITR". The auxiliary set datafiles and other auxiliary instance files will be left in place in the auxiliary destination as an aid to troubleshooting. The state of the recovery set files is determined by the type of failure. Once we resolve the problem, we can try our TSPITR operation again.

## 38.16 Miscellaneous

- Disk Topology and Automatic Performance Tuning - RMAN includes a new disk topology API allowing it to work with more platforms and file types. The information from this API allows RMAN to automatically tune some parameters related to multiplexing and disk buffers, decreasing the need to human intervention.
- Automatic Datafile Creation - RMAN will automatically create missing datafiles in two circumstances. First, when the backup controlfile contains a reference to a datafile, but no backup of the datafile is present. Second, when a backup of the datafile is present, but there is no reference in the controlfile as it was not backed up after the datafile addition.
- Proxy Archived Log Backups - During a proxy backup the media manager takes over full control of the backup process. RMAN is now able to backup and restore proxy copies of archived redo logs if a suitable media manager is used. If a suitable media manager is not available PROXY clause is ignored and a regular backup is performed. Using the PROXY ONLY results in an error of a proxy backup cannot be performed.
- Restore Failover - RMAN now allows recovery process to proceed from one incarnation through to another. The contents of online redo logs are archived before being cleared when an OPEN RESTLOGS operation is issued. As a result it's no longer necessary to create a new backup after OPEN RESTLOGS operations.
- Recovery Through Resetlogs - When a backup file contains corrupt blocks or is inaccessible during restore operations (RECOVER, BLOCKRECOVER, and FLASHBACK DATABASE) RMAN automatically looks for another copy of the file. If one is not available RMAN will use older versions of the file if available. Only if a suitable copy of the file cannot be found will an error be produced. Successful failover operations result in an associated output message.
- Channel Failover - When multiple channels are available for the same device type retriable errors in a backup step will automatically be retried on another channel. Retriable errors are usually associated with media managers failing to access tapes or instance failures in RAC environments.
- Deferred Error Reporting - In addition to error messages during the job execution, the error stack at the end of the command execution now displays errors for all failed steps, making identification of failed step easier.

## 39. RMAN Enhancements in Oracle 11g

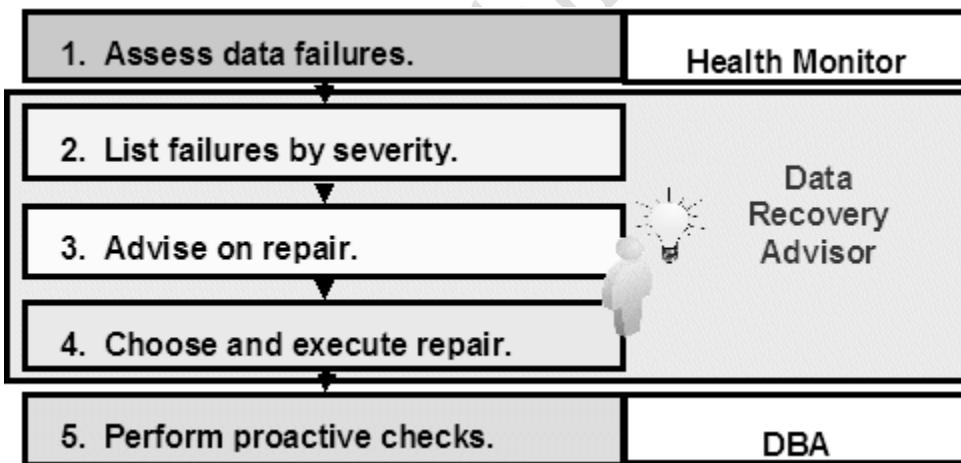
### 39.1 Data Recovery Advisor

A sound database backup and recovery plan is crucial to protecting data in the event of media loss and physical corruption on production storage systems. Oracle backup and recovery technologies enable data to be recovered from media loss and corruption, as well as from logical errors at the row, table, and database level.

Successfully diagnosing a data failure and devising an optimal strategy for repair is critical to maintaining high availability in the enterprise. Data recovery advisor is an Oracle Database tool that automatically diagnoses corruption or loss of persistent data on a disk and gathers data failure information. It then determines the best repair option and checks the feasibility of implementing the repair in your environment.

Figure shows the automatic diagnostic workflow in Oracle Database 11g. Health monitor automatically diagnoses failures and logs them into the automatic diagnostic repository. Data recovery advisor consolidates these failures and attaches a severity to them, such as critical or high. If a DBA requests repair advice, data recovery advisor maps any failures to automatic and manual repair options, checks basic feasibility, and presents the repair advice.

The DBA can choose to manually execute a repair or ask data recovery advisor to perform it. In addition to the automatic, reactive checks of health monitor and data recovery advisor, Oracle recommends that DBAs use the `VALIDATE` command with the `BACKUP` and `RESTORE` commands to proactively check for corrupt blocks and missing files and also to confirm that backups can be restored.



### 39.2 ZLIB Compression

RMAN offered compression of backup pieces in Oracle Database 11g to conserve network bandwidth but many people were slow to use it. Why? Because third-party compression utilities provided faster alternatives to RMAN's own.

Nevertheless, RMAN 11g compression has some neat features that third-party ones do not provide. For instance, when RMAN 11g restores datafiles, it doesn't need to uncompress the files first, provided it performed the compression. This approach offers significant bandwidth savings during restores.

In Oracle Database 11g, RMAN offers another algorithm, ZLIB, in addition to the previously available BZIP2. ZLIB is a much faster algorithm but it does not compress a whole lot. On the other hand, it does not consume much CPU either.

So if you are CPU starved, it's a blessing to have ZLIB compression. (Note that BZIP2 is default in version 15.1; you need to license a new option called Advanced Compression Option to use ZLIB.)

To use ZLIB compression, merely set the RMAN configuration parameter:

```
RMAN> configure compression algorithm 'ZLIB' ;
```

You have to issue the above command if you have changed it earlier. To change it to BZIP2, issue:

```
RMAN> configure compression algorithm 'bzip2';
```

All the compressed backups will use the new algorithm now.

### 39.3 Virtual Private Catalog

The recovery catalog can be a base recovery catalog, which is a database schema that contains RMAN metadata for a set of target databases. A virtual recovery catalog is a set of synonyms and views that enable user access to a subset of a base catalog.

- Execute this command only at the RMAN prompt. RMAN must be connected to the recovery catalog database either through the CATALOG command-line option or the CONNECT CATALOG command, and the catalog database must be open. A connection to a target database is not required.
- The recovery catalog owner, whether the catalog is a base catalog or a virtual catalog, must be granted the RECOVERY\_CATALOG\_OWNER role. This user must also be granted space privileges in the tablespace where the recovery catalog tables will reside. The recovery catalog is created in the default tablespace of the recovery catalog owner.
- If you are creating a virtual recovery catalog, then the base recovery catalog owner must have used the GRANT command to grant either the CATALOG or REGISTER privilege.
- See the CONNECT CATALOG description for restrictions for RMAN client connections to a virtual catalog when the RMAN client is from release Oracle Database 11g or earlier.
- Typically, you create the recovery catalog in a database created especially for this purpose. It is not recommended to create the recovery catalog in the SYS schema.
- The best practice is to create one recovery catalog that serves as the central RMAN repository for many databases. For this reason it is called the base recovery catalog.
- The owner of the base recovery catalog can GRANT or REVOKE restricted access to the catalog to other database users. Each restricted user has full read/write access to his own metadata, which is called a virtual private catalog. The RMAN metadata is stored in the schema of the virtual private catalog owner. The owner of the base recovery catalog controls what each virtual catalog user can access.
- You must take an extra step when intending to use a 8.2 or earlier release of RMAN with a virtual catalog. Before using the virtual private catalog, this user must connect to the recovery catalog database as the virtual catalog owner and execute the following PL/SQL procedure (where base\_catalog\_owner is the database user who owns the base recovery catalog):

```
base_catalog_owner.DBMS_RVCAT.CREATE_VIRTUAL_CATALOG
```

### 39.4 Merging Catalogs

While on the subject of multiple catalogs, let's consider another issue. you may see the need to consolidate all these independent repositories into a single one. One option is to deregister the target databases from their respective catalogs and re-register them to this new central catalog.

However, doing so also means losing all those valuable information stored in those repositories. You can, of course, sync the controlfiles and then sync back to the catalog, but that will inflate the controlfile and be impractical. Oracle Database 11g offers a new feature: merging the catalogs. Actually, it's importing a catalog from one database to another, or in other words, "moving" catalogs.

## 39.5 Proactive Health Checks

It helps you find whether the database is healthy and has no bad blocks. But how can you ensure that? Bad blocks show themselves only when they are accessed so you want to identify them early and hopefully repair those using simple commands before the users get an error. The tool `dbverify` can do the job but it might be a little inconvenient to use because it requires writing a script file containing all datafiles and a lot of parameters.

The output also needs scanning and interpretation. In Oracle Database 11g, a new command in RMAN, `VALIDATE DATABASE`, makes this operation trivial by checking database blocks for physical corruption. If corruption is detected, it logs into the Automatic Diagnostic Repository. RMAN then produces an output that is partially shown below:

```
RMAN> validate database;
```

You can also validate a specific tablespace:

```
RMAN> validate tablespace users;
Or, datafile:
```

```
RMAN> validate datafile 1;
```

```
Or, even a block in a datafile:
```

```
RMAN> validate datafile 4 block 56;
```

The `VALIDATE` command extends much beyond datafiles however. You can validate spfile, controlfile copy, recovery files, Flash Recovery Area, and so on.

## 39.6 Parallel Backup of the Same Datafile

You probably already know that you can parallelize the backup by declaring more than one channel so that each channel becomes a RMAN session. However, very few realize that each channel can back up only one datafile at a time.

So even though there are several channels, each datafile is backed by only one channel, somewhat contrary to the perception that the backup is truly parallel. In Oracle Database 11g RMAN, the channels can break the datafiles into chunks known as "sections." You can specify the size of each section.

Here's an example:

```
RMAN> run {
 allocate channel c1 type disk format '/disk1/bkp/rman/%U';
 allocate channel c2 type disk format '/disk2/bkp/rman/%U';
 backup
 section size 500m
 datafile 6;
 }
```

## 40. Duplicate Database Using RMAN

### 40.1 Starting and Configuring RMAN before Duplication

Before executing the DUPLICATE DATABASE command, perform the following tasks:

Step 1: Start RMAN and Connect to the Database Instances

Step 2: Mount or Open the Source Database

Step 3: Configure RMAN Channels for Use in the Duplication

#### Step 1: Start RMAN and Connect to the Database Instances

Start RMAN and connect to the source database as TARGET, the duplicate database instance as AUXILIARY, and, if applicable, the recovery catalog database. You can start the RMAN client on any host so long as it can connect to all of the database instances. If the auxiliary instance requires a text-based initialization parameter file, then this file must exist on the same host that runs the RMAN client application.

To start RMAN and connect to the target and auxiliary instances:

Start the RMAN client.

For example, enter the following command at the operating system prompt:

```
% rman
```

At the RMAN prompt, execute CONNECT commands for each database instance.

In this example, a connection is established to three database instances, all through the use of net service names:

```
RMAN> CONNECT TARGET SYS/password@prod # source database
RMAN> CONNECT AUXILIARY SYS/password@aux # duplicate database instance
RMAN> CONNECT CATALOG rman/password@catdb # recovery catalog database
```

#### Step 2: Mount or Open the Source Database

Before beginning RMAN duplication, mount or open the source database if it is not already mounted or open. If the source database is open, then archiving must be enabled. If the source database is not open, and if it is not a standby database, then it must have been shut down consistently.

#### Step 3: Configure RMAN Channels for Use in the Duplication

If necessary for backup-based duplication, configure channels to be used on the auxiliary instance. Note that it is the channel on the auxiliary instance, not the target instance that restores backups. RMAN can use the same channel configurations set on the source database for duplication even if they do not specify the AUXILIARY option.

If the auxiliary channels need special parameters (for example, to point to a different media manager), however, then you can configure an automatic channel with the AUXILIARY option of the CONFIGURE command.

In backup-based duplication, the channel type (DISK or sbt) of the channel must match the media where the backups of the source database are located. If the backups reside on disk, then the more channels you allocate, the faster the duplication will be. For tape backups, limit the number of channels to the number of devices available.

In active database duplication, you do not have to change your source database channel configuration or configure AUXILIARY channels. However, you may want to increase the parallelism setting of your source database disk channels so that RMAN copies files over the network in parallel.

## 40.2 Duplicating a Database

This section describes the most basic procedure to create a duplicate database. The procedure depends on how your databases and hosts are configured. This section contains the following topics:

Duplicating a Database to a Remote Host with the Same Directory Structure  
 Duplicating a Database to a Remote Host with a Different Directory Structure  
 Creating a Duplicate Database on the Local Host

### 40.2.1 Duplicating a Database to a Remote Host with the Same Directory Structure

The simplest case is to use active database duplication to duplicate the database to a different host and use the same directory structure. If the source database uses a server parameter file (or a backup is available), then you can create a temporary initialization parameter file on the destination host and set only the DB\_NAME parameter. You do not have to set additional parameters for specifying duplicate database filenames or transfer backups to the destination host.

In this example, the database is open. RMAN has automatic channels already configured. You connect to the database instances as follows:

```
CONNECT TARGET SYS/password@prod
CONNECT AUXILIARY SYS/password@dupdb
```

Execute the DUPLICATE command.

Example illustrates how to use DUPLICATE for active duplication. This example requires the NOFILENAMECHECK option because the source database files have the same names as the duplicate database files.

#### Example Duplicating to a Host with the Same Directory Structure

```
DUPLICATE TARGET DATABASE
 TO dupdb
 FROM ACTIVE DATABASE
 SPFILE
 NOFILENAMECHECK;
```

RMAN automatically copies the server parameter file to the destination host, starts the auxiliary instance with the server parameter file, copies all necessary database files and archived redo logs over the network to the destination host, and recovers the database. Finally, RMAN opens the database with the RESETLOGS option to create the online redo logs.

Assume a slightly different case in which you want to recover the duplicate database to one week ago in order to view the data in the source database as it appeared at that time. Example 23-4 uses backup-based duplication to create a duplicate of the source database as it appeared one week ago. The UNTIL clause is not supported in active database duplication.

#### Example Duplicating a Database to a Past Point in Time

```
CONNECT TARGET SYS/password@prod # source database
CONNECT AUXILIARY SYS/password@dupdb # duplicate database instance
DUPLICATE TARGET DATABASE
 TO dupdb
 PASSWORD FILE
 SPFILE
 NOFILENAMECHECK
 UNTIL TIME 'SYSDATE-7';
```

Note that the command in Example 23 specifies the PASSWORD FILE option to indicate that RMAN should duplicate the password file from the source database.

## 40.2.2 Duplicating a Database to a Remote Host with a Different Directory Structure

If you create the duplicate database on a host with a different directory structure, then you must use some technique to generate filenames for the duplicate database datafiles. The simplest technique is to use active database duplication and to use the SPFILE clause to rename files as explained in "Choosing a Strategy for Naming Duplicate Files". If the source database uses a server parameter file (or a backup is available), then you can create a temporary initialization parameter file on the destination host and set only the DB\_NAME parameter.

Assume that the source database datafiles reside in /oracle/oradata/prod/, and you want to duplicate them to /scratch/oracle/oradata/dupdb/. All of the source database online redo logs reside in /oracle/oradata/prod/redo/, and you want to duplicate them to /scratch/oracle/oradata/dupdb/redo/.

To duplicate a database on a remote host with a different directory structure:

In this example, you create a temporary initialization parameter file with DB\_NAME as the only parameter. You can set this parameter to any arbitrary value.

For this example, assume that the database is open and you have automatic channels configured. You connect to the instances as follows:

```
CONNECT TARGET SYS/password@prod
CONNECT AUXILIARY SYS/password@dupdb
```

Execute the DUPLICATE command.

Example 23-5 specifies names for the duplicate database files with the SPFILE clause in conjunction with the DB\_FILE\_NAME\_CONVERT parameter. Note that the PARAMETER\_FILE\_CONVERT parameter does not affect LOG\_FILE\_NAME\_CONVERT or DB\_FILE\_NAME\_CONVERT, which is why these two parameters must be set separately.

Example Duplicating to a Host with a Different Directory Structure

```
DUPLICATE TARGET DATABASE
 TO dupdb
 FROM ACTIVE DATABASE
 DB_FILE_NAME_CONVERT
 '/oracle/oradata/prod/' , '/scratch/oracle/oradata/dupdb/'
 SPFILE
 PARAMETER_VALUE_CONVERT '/oracle/oradata/prod/' ,
 '/scratch/oracle/oradata/dupdb/'
 SET SGA_MAX_SIZE '300M'
 SET SGA_TARGET '250M'
 SET LOG_FILE_NAME_CONVERT '/oracle/oradata/prod/redo/' ,
 '/scratch/oracle/oradata/dupdb/redo/' ;
```

RMAN automatically copies the server parameter file to the destination host and updates the server parameter file on the destination host based on values provided in PARAMETER\_VALUE\_CONVERT and SET. RMAN then starts the auxiliary instance with the server parameter file; copies all necessary database files and archived redo logs over the network to the destination host, and recovers the database. Finally, RMAN opens the database with the RESETLOGS option to create the online redo logs.

## 40.2.3 Creating a Duplicate Database on the Local Host

When creating a duplicate database on the same host as the source database, follow the same procedure as for duplicating to a remote host with a different directory structure. You can duplicate the database to the same Oracle home as the source database, but you must use a different database name from the source database, and convert the filenames by means of the same methods used for conversion on a separate host.

Caution:

Do not use the NOFILENAMECHECK option when duplicating to the same Oracle home as the primary database. If you do, then the DUPLICATE command may overwrite the datafiles of the source database.

## 41. RMAN Enhancements in Oracle 11g R2

Oracle 11gR2 extends the already-robust suite of backup and recovery tools that comprise *Recovery Manager* (RMAN) with some notable new enhancements, many of which are obvious extensions of earlier features:

### 41.1 Automatic Block Recovery.

In my humble opinion, Oracle 11gR1's introduction of *lock-level media recovery* (BMR) should have been enough to convince even the most stalwart advocate of user-managed recovery to at least consider using RMAN instead of the antiquated **ALTER TABLESPACE ... BEGIN BACKUP** method to back up the database.

BMR meant it was no longer necessary to restore and recover an entire datafile when only a few blocks needed to be recovered. Oracle 11gR2 has now automated BMR so that if one or more corrupted blocks within a datafile are detected, RMAN will automatically restore and recover the blocks without the need for DBA intervention.

And if Data Guard's Real Time Query mode is enabled, Oracle 11gR2 can check the corresponding physical standby for a more recent version of the non-corrupted block and transmit that block to the primary database.

### 41.2 Flexible Set Newname Directives.

If we have several dozen datafiles while restoring a database to a different platform or file system using either RMAN, **DUPLICATE DATABASE**, or tablespace point-in-time recovery. In Oracle 11gR2 accepts either a tablespace-level or database-level **FORMAT** specification for datafile names. In addition, it's now possible to list either a *unique identifier* (%U) or the *base name* (%b) for easier specification of new datafile names.

### 41.3 Enhancement Tablespace Point In Time Recovery (TSPITR)

Oracle 11gR2 has removed some glaring limitations from Tablespace Point-In-Time Recovery (TSPITR) – an extremely useful tool for recovering a tablespace set to a previous point in time earlier than the database's current SCN.

TSPITR can now also be performed multiple times against the same tablespace set regardless if a recovery catalog was available. Also, TSPITR can also be used to recover a tablespace that's been dropped

The scenario here is when we drop a tablespace and want it back! We're testing out the 11gR2 new feature that allows us to **recover a dropped tablespace** from a previous backup.

**Note:** Please look at Demo on duplicate database command enhancements in oracle 11g R2 Key Book.

### 41.4 Enhancement Compression Algorithm levels

Oracle 11g Release 2 introduced compression algorithm levels which can be used for compressing table data, Data Pump exports and RMAN backups as well.

The compression levels are BASIC, LOW, MEDIUM and HIGH and each affords a trade off related to backup throughput and the degree of compression afforded.

## 41.5 Duplicate Without Connection to Target Database

DUPPLICATE can be performed without connecting to a target database. This requires connecting to a catalog and auxiliary database.

The benefit is improved availability of a DUPPLICATE operation by not requiring connection to a target database. This is particularly useful for DUPPLICATE to a destination database where connection to the target database may not be available at all times.

### 41.5.1 Purpose of Database Duplication

A duplicate database is useful for a variety of purposes, most of which involve testing. You can perform the following tasks in a duplicate database:

- Test backup and recovery procedures
- Test an upgrade to a new release of Oracle Database
- Test the effect of applications on database performance
- Create a standby database
- Generate reports

For example, you can duplicate the production database on `host1` to `host2`, and then use the duplicate database on `host2` to practice restoring and recovering this database while the production database on `host1` operates as usual.

If you copy a database by means of operating system utilities rather than with DUPPLICATE command, then the DBID of the copied database remains the same as the original database. To register the copy database in the same recovery catalog with the original, you must change the DBID with the DBNEWID utility. The DUPPLICATE command automatically assigns the duplicate database a different DBID so that it can be registered in the same recovery catalog as the source database.

The DUPPLICATE command can create a fully functional copy of your database or a physical standby database, which serves a very different purpose. A standby database is a copy of the primary database that you update continually with archived log files from the primary database.

If the primary database is inaccessible, then you can fail over to the standby database, which becomes the new primary database. A database copy, however, cannot be used in this way: it is not intended for failover scenarios and does not support the various standby recovery and failover options.

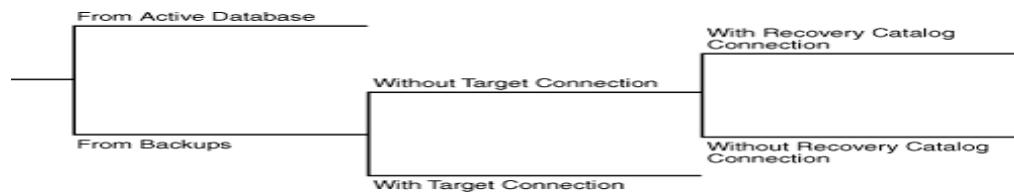
### 41.5.2 Techniques for Duplicating a Database

RMAN supports two basic types of duplication: active database duplication and backup-based duplication. RMAN can perform backup-based duplication with or without either of the following connections:

Target  
Recovery catalog

A connection to both is required for active database duplication.

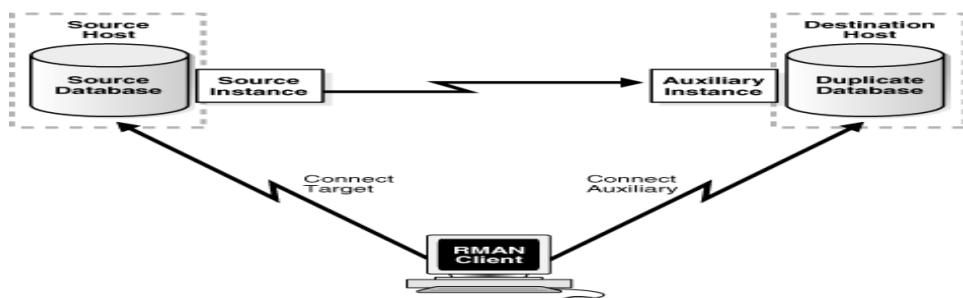
the decision tree for the two duplication techniques.



#### 41.5.2.1 Active Database Duplication

RMAN connects as **TARGET** to the source database instance and as **AUXILIARY** to the auxiliary instance. RMAN copies the live source database over the network to the auxiliary instance, thereby creating the duplicate database. No backups of the source database are required.

#### Active Database Duplication.



#### 41.5.2.2 Backup-Based Duplication

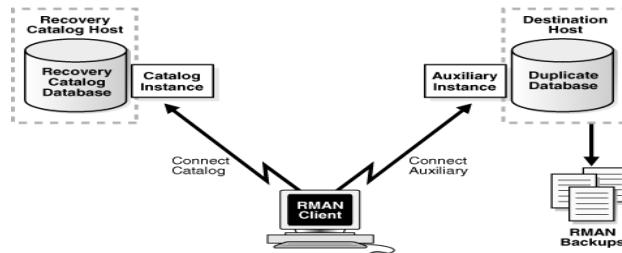
RMAN creates the duplicate database by using pre-existing RMAN backups and copies. This technique of duplication uses one of the following mutually exclusive sub techniques:

Duplication without a target database connection, RMAN obtains metadata about backups from a recovery catalog.

- Duplication without a target database connection and without a recovery catalog. RMAN obtains metadata about where backups and copies reside from BACKUP LOCATION.
- Duplication with a target database connection. RMAN obtains metadata about backups from the target database control file or from the recovery catalog.

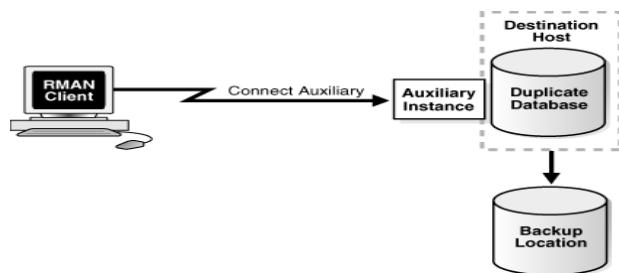
Backup-based duplication without a target connection. RMAN connects to a recovery catalog database instance and the auxiliary instance. The destination host must have access to the RMAN backups required to create the duplicate database.

#### 41.5.3 Backup-Based Duplication without a Target Connection



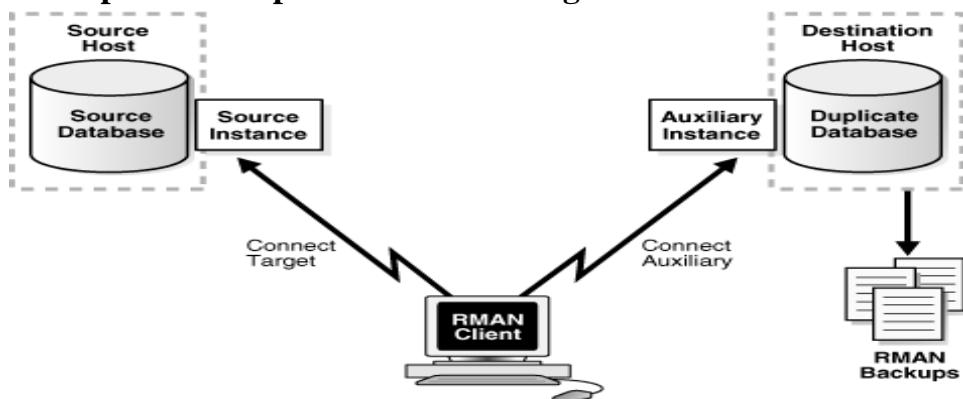
Backup-based duplication without connections to the target or to the recovery catalog database instance. RMAN connects to the auxiliary instance of the duplicate database on the destination host. A disk backup location containing all the backups or copies for duplication must be available to the destination host.

#### 41.5.4 Backup-Based Duplication without a Target Connection or Recovery Catalog Connection



Backup-based duplication with a target connection. RMAN connects to the source database instance and the auxiliary instance. Optionally, RMAN can connect to a recovery catalog database (not shown in the figure). The destination host must have access to the RMAN backups required to create the duplicate database.

#### 41.5.5 Backup-Based Duplication with a Target Connection



#### 41.5.6 Contents of a Duplicate Database

A duplicate database can include the same contents as the source database or only a subset of the tablespaces in the source database. For example, you can use the `TABLESPACE` option to duplicate only specified tablespaces, or the `SKIP READONLY` option to exclude read-only tablespaces from the duplicate database.

#### 41.5.7 How RMAN Duplicates a Database

For backup-based duplication, the principal work of the duplication is performed by the auxiliary channels. These channels correspond to a server session on the auxiliary instance on the destination host. For active database duplication the primary work is performed by target channels.

RMAN must perform database point-in-time recovery, even when no explicit point in time is provided for duplication. Point-in-time recovery is required because the online redo log files in the source database are not backed up and cannot be applied to the duplicate database. The farthest point of recovery of the duplicate database is the most recent redo log file archived by the source database.

As part of the duplicating operation, RMAN automates the following steps:

1. Creates a default server parameter file for the auxiliary instance if the following conditions are true:
  - o Duplication does not involve a standby database.
  - o Server parameter files are not being duplicated.
  - o The auxiliary instance was not started with a server parameter file.
2. Restores from backup or copies from active database the latest control file that satisfies the UNTIL clause requirements.
3. Mounts the restored or copied backup control file from the active database.
4. Uses the RMAN repository to select the backups for restoring the datafiles to the auxiliary instance. This step applies to backup-based duplication.
5. Restores and copies the duplicate datafiles and recovers them with incremental backups and archived redo log files to a noncurrent point in time.
6. Shuts down and restarts the database instance in NOMOUNT mode.
7. Creates a new control file, which then creates and stores the new DBID in the datafiles.
8. Opens the duplicate database with the RESETLOGS option and creates the online redo log for the new database.

## 42. RMAN Enhancements in Oracle 12c

### 42.1 SQL Interface Improvements

In Oracle 12c, you can run SQL commands in RMAN without preceding the command with the SQL keyword. You also no longer need to enclose the SQL command in quotes.

The RMAN DESCRIBE provides the same functionality of SQL\*Plus DESCRIBE:

You can run DDL/DML Commands from RMAN Command prompt, but note that in order to insert you need to use.

The user can SHUTDOWN/STARTUP the database and also can use ALTER commands.

With this new SQL Interface improvement's, users don't need to switch between RMAN and SQL\* Plus prompts during Backup & Recovery, administration...etc.

```
RMAN> select name from v$datafile;
```

```
NAME
```

```

```

```
/disk2/oradata/anu99/system.dbf
```

```
/disk2/oradata/anu99/sysaux.dbf
```

```
/disk2/oradata/anu99/undotbs.dbf
```

```
/disk2/oradata/anu99/userdata.dbf
```

```
RMAN> alter database mount;
```

```
RMAN> select sysdate from dual;
```

```
SYSDATE
```

```

```

```
27-DEC-14
```

```
RMAN> create table wst1.exam (eid number, ename varchar2(10)) tablespace users;
```

Using target database control file instead of recovery catalog  
Statement processed

```
RMAN> desc wst1.exam
```

| Name  | Null? | Type         |
|-------|-------|--------------|
| EID   |       | NUMBER       |
| ENAME |       | VARCHAR2(10) |

## 42.2 Sysbackup Privilege

Prior to 12c, users needed SYSDBA privilege to backup the database. The new **Sysbackup** privilege allows the user the permissions to perform only backup operations.

The **Sysbackup** privilege allows the DBA to perform RMAN backup commands without additional privileges. Using this new role in 12c, you can segregate Administration and Backup operations.

With RMAN you have same authentication options that are available with SQL\*Plus, which are Operating system authentication and password file authentication.

- **To connect to RMAN using OS Authentication, Authentication with the Sysbackup privilege use:**

```
$ rman target ' / as sysbackup' '
 ▪ Authentication with the SYSDBA privilege use:
$ rman target ' / as sysdba' '
 ▪ You can also implicitly connect using below command:
$ rman target /
```

To connect to RMAN using password file authentication with the SYSBACKUP privilege use:

```
$ rman target1 ' "bkpadm@DB1 as sysbackup" '
```

Where bkpadm is the user and should have SYSDBA privilege.

Authentication with the SYSDBA privilege

```
$ rman target ' "sysadm@DB1 as sysdba" '
```

You can implicitly connect using below command, where **SYSADM** is the user and should have SYSDBA privilege:

```
$ rman target sysadm@DB1
```

Note that SYSBACKUP does not include data access privilege, such as SELECT ANY TABLE. When you don't specify the role explicitly then the default used is AS SYSDBA.

## 42.3 Recover a single Table

Oracle 12c is now providing a packaged technique which recovers only the necessary tables so as to get the data dictionary and any undo/redo segments necessary to get all the data back to a specific point in time whether that be a SCN or a timestamp.

**Special Note:** The default size of the SGA is 1G. It is a hard-coded parameter as it is not dependent on the existing database SGA size. The file **recover.bsq** file in **\$ORACLE\_HOME/rdbms/admin/** contain the functions used to size the SGA .

**Steps involved to bring back the table T1 into database using RMAN:**

A Full Database backup exists

The dropped table is "T1"

The owner of the table is "TEST"

Recovery is done as user "SYSBACKUP"

Auxiliary Destination used is "AUX"

Table is restored as "T1\_PREV" to check the contents of dropped table

Directory "TSPITR\_DIROBJ\_DPDIR" is automatically created by RMAN

Creates a export dumpfile tspitr\_<name>.dmp  
Imports the table as T1\_PREV into the database  
Deletes all auxiliary destination files used for the recovery.

```
SQL> select current_scn from v$database;
```

```
CURRENT_SCN
```

```

1423211 ---- This is the time the table TEST.T1 is dropped from the database.
```

```
[oracle12c@wil ~]$rman target ' "/ as sysbackup" '
RMAN> run
{
 Allocate auxiliary channel ch1 device type disk format '%U.BKP';
 RECOVER TABLE 'TEST'.T1' until scn 1423211
 AUXILIARY DESTINATION '/disk1/oradata/cdb/aux'
 REMAP TABLE 'TEST'.T1':T1_PREV';
}
```

## 43. Performance Tuning

Tuning a Database Server is done in various levels.

- Hardware Level
- OS Level
- Oracle Database Level

### 43.1 Hardware Level

#### 43.1.1 CPU

- Usually in production DB systems, we go with SMP (Symmetrical Multi-Processing) two ways, 4 ways, 8 ways etc.
- These servers are scalable which means as the load increases on the server we increase the number of CPU to avoid any CPU level bottlenecks.
- In UNIX systems we use commands like SAR (System Activity Report) to find out the performance delays are because of CPU or not.
- We use cron facility to take the statistics of this. SAR (System Activity Report) may be executed, a report during business hours, like wise at least a month. At the end of the month we can determine whether the available CPU is being sufficient or not.

#### 43.1.2 MEMORY

- We need to observe whether the primary memory available in the server is being sufficient or not. Keeps an eye on the amount of swap space being utilized?
- In Unix systems we can find out this by using commands like:
- VM (virtual Memory) stat: - vmstat
- Swap: swap -l
- Once in a while usage of swap may not cause big damage to the performance but it's being continuously used. We need to consider adding some more RAM, provided the server is scalable.
- The size of the oracle's SGA also depends on how much primary memory we have in the box and its needless to mention, the big the SGA the better the performance.
- Our main intention is to reduce the I/O by carrying maximum possible data in the shape of blocks within SGA. (We don't like most frequently used blocks to be flushed out which greatly causes performance degradation)

#### 43.1.3 I/O

- The Database Servers more than CPU or memory are I/O bound as ultimately the data has to come from/return to I/O (Disk).
- We need to optimize the I/O subsystem to handle several reads/write per second in a production environment.
- Because this is a database server we use high-end I/O controllers small computer system Interface (SCSI), fiber channel (FC).
- Instead of carrying the entire data in one large disk it's highly recommended to go with multiple smaller drives so that read write operations can be supported even parallel.
- Plan on using higher RPM (Rotation per Minute) (15000 RPM), which offers better average seek time.  
E.g.: A 15k RPM SCSI drive offers about 6 to 7 million secs of AST (average seek time) compared to 7200 RPM of enhanced EIDE (enhanced Integrated Drive) which offers 16 to 18 million secs of AST.
- Consider implementing RAID (Redundant Array of Inexpensive Drives) so that we can implement
- Availability (RAID 1 and RAID 5)
- Performance with striping (RAID 0, RAID 5)
- RAID can be implemented at
- Hardware level
- O/S level. Using volume manager software like VERITAS. (Using SCSI RAID controller).
- Although we like multiple drives on a server, let's not connect all the drives to a single SCSI controller instead plan on having multiple SCSI controllers each one supporting 4 to 6 drives.

## 43.2 Operating System Level

- Optimize the O/S kernel with appropriate Kernel Tunable Parameters (KTP) so that the available resource can be effectively utilized by Oracle Instance.  
E.g.: SHMMAX (Shared Maximum memory)  
SEMMNI (semaphore)  
NPROC (Number of processes etc.)
- Any unutilized services which are still running must be turned off. (Visit /etc/rc2.d) and rename any "S" script to "s" (so that at the boot this won't start).
- Consider executing bigger jobs during the nights which can be scheduled by cron - effective usage of the sever for day/night.
- Use ps -ef to find out details about all the currently running processes along with their Resource Utilization
- Use the following commands to get more insight on performance at OS level.
- sar top vmstat iostat netstat  
nfsstat
- ps -ef swap perfmetre lpcsv -b df -k du (disk usage)
- Optimize filesystem for better throughput of I/O.
- Oracle offers 10 to 15% better I/O performance by using RAW devices over traditional filesystem.
- Make sure every filesystem has at least 20% free space as the performance drastically comes down when the usage reaches beyond this point.
- Make sure no filesystem is fragmented beyond 8% (use fsck -m to find out level of fragmentation).
- Just like we perform the database RE-ORG, it is also necessary to go for filesystem RE-ORG.

## 43.3 Performance Tuning at Database Level

### 43.3.1 Oracle Versions

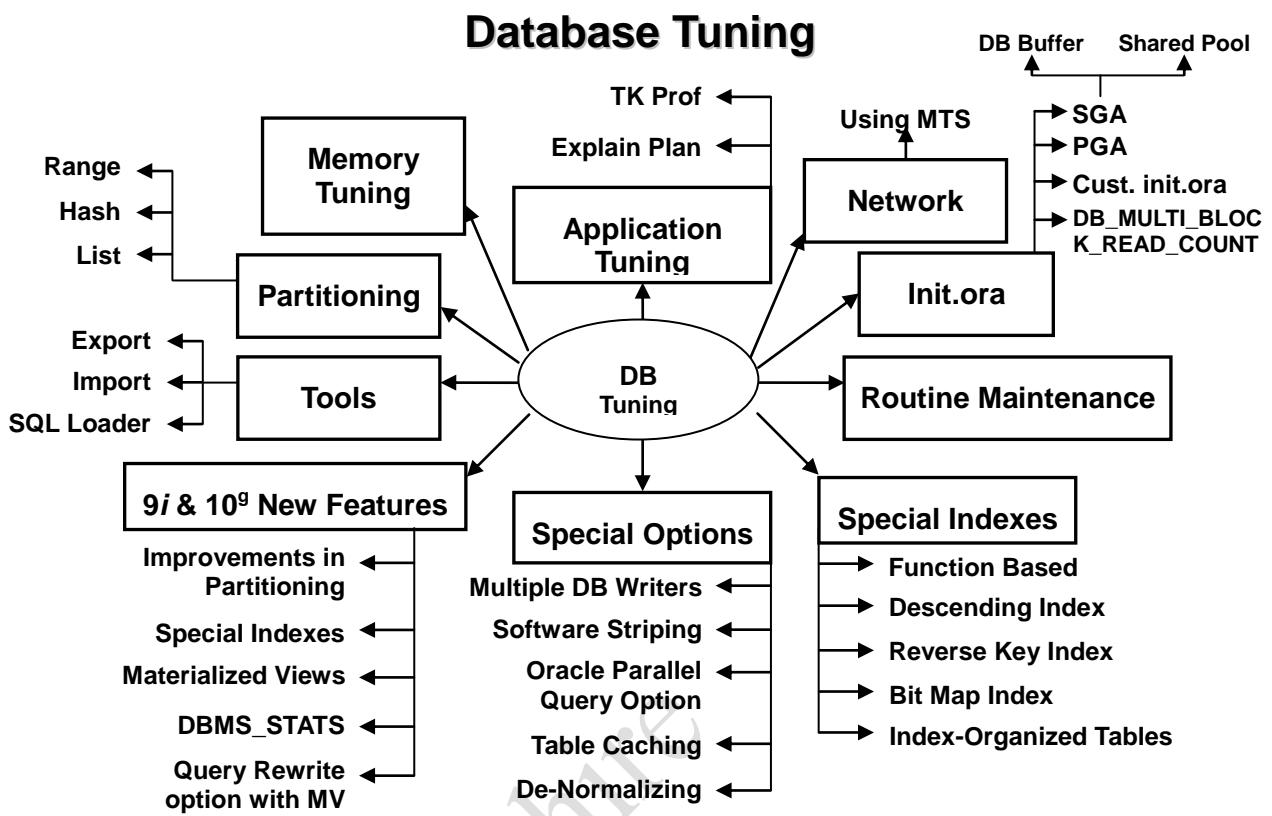
| 8i Features                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 9i Features                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | 10g & 11g New Features                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 12c Features                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>Materialized View with Query Re-write.</li> <li>Special Indexes (total of 5)</li> <li>Index rebuild</li> <li>Resumable Tablespace</li> <li>Implement No-logging at TS - level</li> <li>Implementing ASSM (Automatic Segment Space Mgmt) to avoid "Freelists" to improve the Performance</li> <li>Use DBMS_STATS package to transfer collected statistics from standby to production server.</li> <li>Implementing T-TS (Transportable-Tablespace) to copy an entire TS from one DB to another (E.g. Copying a TS from PROD to DEV)</li> <li>Implementing LM-TS to reduce pressure on System-TS</li> </ul> | <ul style="list-style-type: none"> <li>Table compression (11.2)</li> <li>Online Segment Redefinition</li> <li>Multiple block size support at TS level</li> <li>AUM (automatic Undo Management)</li> <li>Caching Sequences</li> <li>List-Partition can be implemented where applicable</li> <li>Can create Temporary Tables from PL*SQL procedures which are created in Memory for faster access and are available only for that session (gets dropped once the session ends).</li> <li>Resumable Space Allocation</li> <li>Online Table Redefinition</li> </ul> | <ul style="list-style-type: none"> <li>Dynamic SGA</li> <li>ASSM (Automatic Segment Space Mgmt)</li> <li>AOS (Automatic Optimizer Statistics)</li> <li>AWR (Automatic Work Load Repository)</li> <li>ADDM (Automatic DB Diagnostic Monitor)</li> <li>ASH (Active Session Report)</li> <li>ASMM (Automatic Shared Memory Management) with MMAN Background Process</li> <li>Temp TS Group Concept.</li> <li>Compressed RMAN Backups</li> <li>Using DataPump (expdp/impdp) - Usage of multiple threads (Parallel Processing using multiple CPUs)</li> <li>Advisors (Memory, Space, Tuning)</li> </ul> | <ul style="list-style-type: none"> <li>Adaptive SQL Plan Management (SPM)</li> <li>Adaptive query optimization</li> <li>Adaptive plans</li> <li>Automatic re-optimization</li> <li>SQL plan directives</li> <li>Dynamic statistics enhancements</li> <li>Online statistics gathering for bulk loads</li> <li>SQL Tuning Sets</li> <li>Automatic column group creation</li> </ul> |

### 43.3.2 Routine Maintenance:-

- Database Level RE-ORG at least once in 4 months to get rid of Fragmentation
  - 1. Full-Export
  - 2.Drop Database
  - 3.Re-Create Empty Database
  - 4.Full Import
- Make sure each Tablespace has at least 20% free space.
- Every two weeks rebuild all indexes, to get rid of leaf level fragmentation, which is caused because of insert/update/delete.
- Make sure no segment has beyond 30 extents as the performance comes down when “selecting” against a table, which has too many extents.
- Always keep an eye on the hit ratio, which should never be less than 90%.
- As oracle CBO (cost based optimizer) hugely depends on the available statistics to come with best possible execution plan – make sure we collect statistics on daily basis.
- Make sure row chaining/row migration is avoided, by properly planning the block size and PCT free.
- Depending on the database style/insert oriented or update oriented, we need to optimize the database accordingly (PCTfree)
- Convert slowly running complex views into de-normalized tables with appropriate DB triggers/Indexes. Even better thing would be implementing Materialized Views (MV)
- Instead of using default buffer size (which is only 256KB) while using oracle tools like – Exp, Imp and SQL\*Loader mention bigger buffer size (may be 5MB or more).
- Drop Unused Indexes. This can be find by using
  - `SQL> ALTER INDEX index_name MONITOR USAGE;`
- While loading Bulk-Data into a Table, consider drop Index and re-create after loading data successfully into tables.
- Avoid Full-Table-Scans by having indexes on appropriate columns based on Report's criteria.

### 43.3.3 Special Features: -

- To reduce the I/O we can implement Table Caching technique for MFU (Most Frequently Used) when the table is smaller in size (e.g. Look-Up Tables)
- If no volume manager has be implemented no raid levels get configured, we can implement oracle's software stripping so that one segment can have extents on different datafiles belonging to same TS which offers better I/O throughput.
- Consider choosing multiple database-writers when having multiple drives in the server so that the write operations can go on parallel.
- When having multiple CPU's in the sever consider implementing O-PQO (Oracle Parallel Query Option).
- Consider having enough PGA so that Sort-operations can take place in Memory rather than in Disk (using Temporary-Tablespace)



## 44. SQL Tuning

Can you influence database performance by the SQL you choose? Yes, we can. Mostly, this means knowing what indexes you have and how SQL code does or does not take advantage of them.

There are many other factors database design, hardware configuration, system architecture, network setup that have profound effects on performance. While it's true that a poorly written query can slow performance by invalidating relevant indexes, it's equally certain that a really well crafted query cannot overcome fundamental system limitations.

Unfortunately, SQL tuning is never cut-and-dried. It's a constant weighing of one factor against another, requiring endless measuring and attention and frequent changes. In addition, you won't find the msdpn database much use in testing different approaches to queries.

Most optimizers ignore indexes when the data set is very small and simply read the tables sequentially (do table scans). Review the queries in this section to make sure you understand the principles of writing efficient SQL, but be sure to test specifics on your system with something much closer to real data.

Understanding indexes and writing queries to take advantage of indexes can give you the best performance improvements on an existing database. These are the steps you go through when you start thinking about query performance:

- Find out what kind of work your database is tuned for and what it looks like the tables, views, and other objects in it. An entity relationship diagram is very helpful. If you don't have one, use system tools to sketch out the major tables.
- Get a list of indexes for the tables you'll be using via system catalogs, GUI interfaces, or commands.
- Look into the details of your optimizer and identify tools for seeing how the optimizer handles different queries.
- Armed with these tools, examine specific "slow" queries. First of all, make sure they are returning the correct results.
- Review the WHERE clause. Are relevant indexes useful? Up to date? Is any construction nullifying an available index? Could you use a covering index?
- Check joins are join columns compatible?
- Is there any unnecessary or redundant action? Are DISTINCT and UNION overused? Do you see HAVING used like WHERE? Are queries complicated by going through views?

There is a list of index/query areas to consider in the chapter summary.

Since index and optimizer tools vary greatly from system to system (they are not SQL commands), you'll find brief instructions on using the Adaptive Server Anywhere tools in this chapter just enough to let you follow the examples if you are using the software on the CD. For details on Adaptive Server Anywhere (or basics on any other system), you're on your own!

### 44.1 Defining the Basic Problem

Proper indexing can have a huge effect on database efficiency. Just like an index in a book, an index in a database makes it easy to find data. Instead of paging through the book, hoping to find information about Emu Sisters Productions, you can go directly to the location the index gives you. Now think about what happens when you remove Chapter 3 from a book or add 20 pages to Chapter 7. The book index is invalidated from the point of change on and needs to be rebuilt.

In much the same way, adding, removing, and changing rows in a database leads to index maintenance tasks. Even a relatively small edit, such as changing Emu Sisters to E-Gals, is likely to cause some index differences, as the shorter size of the new entry affects the page splits.

If you are changing book text or database rows all the time and you have very detailed indexes, you'll need to get the text pages and index back in synch. So what's good for one kind of work (finding information/querying/decision support) is bad for another (modifying information/updating, inserting, and deleting/transaction processing).

It's not just indexes that show this dichotomy. Database design, storage allocations, memory, and many other choices reflect the work a database is supposed to do. These decisions are beyond the scope of this book, and you're best off looking at information specifically for the system you are using.

However, returning to the question of indexes, there are some general rules the SQL writers can follow to improve performance (at least with sorted or B-tree indexes, common in many systems). First, find out what indexes you have and put together WHERE clauses that can use the indexes rather than WHERE clauses that can't use the indexes.

Often, the difference is minor. For example, in just about every system, the queries in the below will give different performance, assuming an index on deptno and a nontrivial number of rows.

### WHERE and Performance

```
SQL> SELECT empno,ename,job,sal FROM emp WHERE deptno in (10,20,30);
```

Better performance with BETWEEN

```
SQL> SELECT empno,ename,job,sal FROM emp WHERE deptno BETWEEN 10 AND 30;
```

Why? IN processes (a form of OR) are often slow, requiring multiple reads of the table. BETWEEN is easier for most SQL engines to handle. After all, it's just a range. With an index, it's a matter of finding the low boundary and following the index until encountering the high value.

Once you have general guidelines on efficient SQL, you can look for places where you can substitute faster syntax for slower. Sometimes you won't have the option often IN (or some other "slow" choice) is the best answer. However, as you review code you've written yourself or inherited, you'll find places where you can improve performance with relatively minor changes.

## 44.2 Understanding the Optimizer and Associated Tools

To see how query variants affect performance, you need two things: information about indexes and a way to see the choices the optimizer made. Most systems provide tools for both these functions, but they vary widely.

### Monitoring Performance Table

| ANSI | ASA                 | ASE                                 | MS SQL Server           | Oracle          | Informix          |
|------|---------------------|-------------------------------------|-------------------------|-----------------|-------------------|
|      | PLAN ( 'query'<br>) | SET SHOWPLAN<br>ON<br>SET NOEXEC ON | SET SHOWPLAN _ALL<br>ON | EXPLAIN<br>PLAN | SET EXPLAIN<br>ON |

### 44.2.1 SQL Conventions

Before diving into performance in SQL queries, consider how your code looks. Code is easier to read and understand if you present it consistently. In some systems, reusability of cached code may depend on the various copies being identical.

Differences as small as a single space character may be relevant. In addition, training time for new employees is shorter if they can expect consistent patterns. For your sanity, develop coding guidelines. Here are some common suggestions.

Start each line with a SQL verb (SELECT, FROM).

```
SQL> SELECT empno,ename,job,sal FROM emp WHERE deptno BETWEEN 10 AND 30;
```

Indent continued lines.

```
SQL> SELECT empno,ename,job,sal FROM emp
WHERE deptno BETWEEN 10 AND 30 AND sal>1000;
```

- Be consistent in naming tables and columns don't make some table names singular and others plural, don't use case randomly, and don't call one column hiredate and a related column in another table hire\_date.
- If you use table aliases, stick to the same ones, and don't use nonmnemonic aliases such as a, b, and c for emp, salgrade, dept.
- Put in lots of comments: the date, your name, what the query or script is about everything you want to know.

#### 44.2.2 Managing the WHERE Clause

The WHERE clause is the place to start your search for SQL you can change to get better performance. You already know what indexes you have. Now look at the optimizer plan for a problem query and see if the indexes are getting used or if the optimizer is searching the table row by row.

#### 44.2.3 Why a Table Scan?

Just because you have an index on a column doesn't mean your optimizer will use it.

- If the amount of data is trivial (as in the msdpn tables), a search using an index may not be faster than a table scan and the optimizer may choose not to use the index.
- If the query includes all rows in the table (no WHERE clause), the optimizer does a table scan.
- If you're retrieving a lot of data from the table, an index may give no advantage.
- If the optimizer does not have accurate information on data distribution, it may pick the wrong index or no index at all.
- If you include certain elements in the WHERE clause, you may make the index unavailable. Of these elements, only the last two are under your control.

#### 44.2.4 Data Distribution Statistics

First, check your system documents to see if the DBA needs periodically to run a command to keep the optimizer current (UPDATE STATISTICS in Transact-SQL and Informix, ANALYZE in Oracle). Microsoft SQL Server also supports a command that tells you when the command was last run (STAT\_DATE—there's more information on it in "Using System Functions" in Chapter 7). See Table 6–2 for a list of commands associated with index statistics.

The SQL Anywhere ESTIMATE command is included in the table, but it works differently than the UPDATE STATISTICS or ANALYZE commands, allowing the user to give the optimizer hints on data distribution.

##### Index Statistics Table

| ANSI | SQL Anywhere | SAE               | MS SQL Server                | Oracle  | Informix          |
|------|--------------|-------------------|------------------------------|---------|-------------------|
|      | ESTIMATE     | UPDATE STATISTICS | UPDATE STATISTICS STATS_DATE | ANALYZE | UPDATE STATISTICS |

#### 44.2.5 Disabling an Index with a Bad WHERE

Second, don't disable a valid index by the way you construct your WHERE clause. The easiest elements to optimize are comparison operators (=, >, <, and variants) or operators that can be translated to comparison operators (BETWEEN and some LIKE clauses).

## 44.3 Asking Performance Questions

Don't come out of this chapter feeling that you shouldn't use functions or DISTINCT or views. Instead, train yourself to look for the fastest way to achieve what you want.

Start by asking the big questions:

1. Does the query return useful results?
2. Is it lacking anything? Does it bring back extraneous information?
3. Is it still needed? Could it be run less frequently or against smaller amounts of data?
4. Is the sense that this query is "slow" realistic? Why?
5. Next, get some information on the indexes.
6. Are there indexes on columns frequently used in queries?
7. Is the index on the right column or group of columns?
8. Is the optimizer using the indexes?
9. Are there unused indexes?
10. If the optimizer is not using existing indexes, find out why.
11. Does the optimizer have up-to-date information on the indexed column's statistics?
12. Does a table scan make more sense than using an index (lots of rows coming back)?
13. Is the query written so that it invalidates an available index (math, functions, and conversions on indexed columns)?
14. Is IN used where a range or an equal sign would return the same results?
15. Are the columns in multicolumn indexes in the wrong order-built on fname, lname when queries usually look for lname?
16. Finally, look for ways you can speed up processing.
17. Would it help to add an index?
18. Could indexes that are never used be dropped?
19. Could common queries benefit from covering indexes?
20. Are there unnecessary DISTINCTs?
21. Does the query use UNION where UNION ALL would work?
22. Are views more complex than necessary?
23. Are hardwired index choices now less than optimal?

## 44.4 Tips on working Effective SQL Statements

Anything else may make your indexes unavailable. Here are some suspicious areas to investigate. Since architecture and optimizers vary so much, you'll have to check your system documentation to find out just how these areas affect (or don't affect!) your queries.

1. Comparing columns in the same table
2. Choosing columns with low-selectivity indexes
3. Doing math on a column before comparing it to a constant
4. Applying a function to column data before comparing it to a constant
5. Finding ranges with BETWEEN
6. Matching with LIKE
7. Comparing to NULL
8. Negating with NOT
9. Converting values
10. Using OR
11. Finding sets of values with IN
12. Using multicolumn indexes

#### 44.4.1 Comparing Columns in the Same Table

In many systems, comparing columns in same table makes an index useless. For Ex, consider matching empnum column to a string constant versus matching it to another column, bosnum. The two queries return same results.

```
SQL> SELECT fname, lname, empnum, bosnum FROM employee WHERE empnum =
bosnum;
SQL> SELECT fname, lname, empnum, bosnum FROM employee WHERE empnum
='443232366';
 fname lname empnum bosnum
----- ----- ----- -----
Scorley Blake-Pipps 443232366 443232366
1 row
```

With a nontrivial number of rows, the first query scans the table sequentially. The second uses the empix index. If you test this query on another system, you may not see this difference, because the employee table is so small.

However, it's clear that an index on empid helps find the employee with a particular identification number and is less useful in finding one with the same number as the boss. Why? Because a constant (443232366) is constant. An index points to a known value, not an unknown value.

#### 44.4.2 Using Nonselective Indexes

A unique index is 100% selective. Every index entry points to a single location in the data. A nonselective index is just the opposite each index entry points to multiple data locations. You can think of selectivity as roughly the number of distinct index entries divided by the number of data rows. Optimizers don't get much advantage from nonselective indexes, and they often don't use them.

Returning to the book index comparison, imagine an index that listed every occurrence of the word "and." Because "and" is so common, you'd find an index reference to just about every page in the book. Using the index does not make finding "and" faster than paging through the book, because "and" has low selectivity.

Consider attributes such as gender, where there are only two choices. There's no point indexing this kind of column unless the data distribution is much skewed. If 90% of the employees are male and 10% are female, the optimizer might use the index where gender is female. It would not use it in searches for males.

#### 44.4.3 Doing Math on a Column

Another WHERE clause element to watch out for is doing math on an indexed column before comparing it to a constant. The index can find the column value but not the column value \* 2. To test this out on the msdpn database on the Adaptive Server Anywhere CD, run these two queries.

```
SQL> SELECT prodnum, price FROM product WHERE price * 2 > 200;
SQL> SELECT prodnum, price FROM product WHERE price > 100;

 prodnum price
----- -----
 2111 1115.99
 1106 144.00
 1794 400.00

3 rows
```

First one does a table scan. Second takes advantage of the pricix index. This change is relatively easy to enforce. Simple indexed columns on the left! Computations on the right! Get in habit of changing WHERE clauses as in.

#### Calculation in the WHERE Clause

```
WHERE price +5 < 20 → WHERE price < 15
WHERE price - 400 = 0 → WHERE price = 400
WHERE price - 2 = 117 → WHERE price = 119
```

#### 44.4.4 Using Functions

Functions have the same effect on an indexed column in a WHERE clause as math does. The index points to a value but understands nothing about calculations or functions. You already know that a search for employees by number uses the empix index.

```
SQL> SELECT fname, lname, empnum, bosnum FROM employee WHERE empnum
= '443232366';
```

If you modify the column name (empnum) with a substring function, even one that does exactly the same match and produces the same results, you'll get a table scan.

```
SQL> SELECT fname, lname, empnum, bosnum FROM employee
WHERE substr(empnum, 1, 9) = '443232366';
```

| fname   | lname       | empnum    | bosnum    |
|---------|-------------|-----------|-----------|
| Scorley | Blake-Pipps | 443232366 | 443232366 |

1 row

Here's another example. The first query does a table scan. The second uses the empid index.

```
SQL> SELECT fname, lname, empnum, bosnum FROM employee
WHERE empnum || ' ' || fname = '443232366 Scorley';
```

```
SQL> SELECT fname, lname, empnum, bosnum FROM employee
WHERE empnum = '443232366' AND fname = 'Scorley';
```

| fname   | lname       | empnum    | bosnum    |
|---------|-------------|-----------|-----------|
| Scorley | Blake-Pipps | 443232366 | 443232366 |

#### 44.4.5 Finding Ranges with BETWEEN

If there is an index on a column, BETWEEN will not disable it. BETWEEN is treated as a pair of comparison operators. The low value must precede the high value.

The first of the following queries uses the prodix index on prodnum. The second uses the index on price.

```
SQL> SELECT prodnum, name FROM product WHERE prodnum BETWEEN '1110' and
 '1357';
 prodnum name

 1110 star systems
 1255 bug stories
 1357 nt guru

3 rows

SQL> SELECT prodnum, name, price FROM product WHERE price BETWEEN 300 and
 400;

 prodnum name price

 1794 memory8 400.00

1 row
```

NOT BETWEEN is not as easy to resolve with an index. An index points to a specific value. In the following case, you get a table scan.

```
SQL> SELECT prodnum, name, price FROM product WHERE price NOT BETWEEN 10
 AND 100;

 prodnum name price

 2111 memory tripler 1115.99
 1794 memory8 400.00
 1106 z_connector 144.00

3 rows
```

#### 44.4.6 Matching with LIKE

Most systems can take advantage of an index with LIKE as long as you provide the first character in the pattern. To follow this example, start by creating an index on the name columns of customer, last name first.

```
CREATE UNIQUE INDEX custnmix ON customer (lname, fname);
SQL> SELECT lname, fname FROM customer WHERE lname like 'Pe%';
The SQL engine translates the LIKE into a range comparison and is able to
take advantage of the index.
SQL> SELECT lname, fname FROM customer WHERE lname >= 'Pe' and lname <
 'Pf';
 lname fname

 Peters Pete

1 row
```

However, the picture is different if you start with a wildcard. The optimizer does not use an index—the index cannot point to an unknown value.

```
SQL> SELECT lname, fname FROM customer WHERE lname like '%ete%';
You get the same answer, of course, but the method is a table scan.
```

#### 44.4.7 Comparing to NULL

Adaptive Server Anywhere uses the index on product.price in both the following queries, but not all systems do. Check vendor documentation to find out how nulls and indexes relate. If your system takes the "a null is not equal to anything" dictum into the realm of indexes, you may want to campaign for defaults instead of nulls at design time.

```
SQL> SELECT prodnum, price FROM product WHERE price = 44.99;
SQL> SELECT prodnum, price FROM product WHERE price is null;
```

Another issue to check out is whether or not null returns a value in an IN. The following query may return one or two rows, depending on how your system handles null values in this situation.

```
SQL> SELECT prodnum, price FROM product WHERE price in (null, 400.00);
```

#### 44.4.8 Negating with NOT

A related area is how NOT and negatives such as `<>` and `!=` affect index use. Indexes, after all, point to entries. What do they know about nonentries? The two following queries return the same results here (they could return quite disparate results with different data). The first uses the custnmix on last and first names. The second does not.

```
SQL> SELECT lname, fname FROM customer WHERE lname = 'WONG';
SQL> SELECT lname, fname FROM customer WHERE lname not between ' ' and 'W';
```

| lname | fname  |
|-------|--------|
| WONG  | LI-REN |

1 row

Get in the habit of converting NOT phrases when you can. In most cases, you'll be better off looking for rows with prices greater than zero rather than rows with prices not equal to zero. In addition to its effect on index uses, negative logic is sometimes hard to understand and therefore open to error.

#### Negatives in WHERE

This SQL Statement is wrong

```
SQL> SELECT name
 FROM product
 WHERE prodnum !> 90
```

Name

-----

money master

memory manager

this is a valid SQL Statement

```
SQL> SELECT name
 FROM product
 WHERE prodnum <= 90
```

Conversions, like math or functions, can disable an index, and this is true for auto conversions the system handles as well as conversions you perform. For starters, keep the conversion on the right side of the equation.

#### Conversions in WHERE

This SQL Statement is wrong

```
SQL> SELECT prodnum,price FROM product
 WHERE CONVERT(varchar(6),price) = '400.00'
```

This is a valid SQL Statement

```
SQL> SELECT prodnum,price FROM product
 WHERE price = CONVERT(numeric(10,2), '400.00')
```

Prodnum Price

-----

1794 400.00

In ASE, character data defined to allow null is actually stored as VARCHAR data, so joining two columns that differ only in whether or not they allow nulls may cause a conversion. The optimizer cannot use an index on the converted column. In Oracle, some auto conversions deactivate an index, others don't.

#### 44.4.10 Using OR

An OR clause returns results if any one of the conditions is true (A = 1 or B > A or C = 3). If columns in an OR clause have indexes, the optimizer can use the relevant indexes or do a table scan. Using the indexes means dumping results from each clause into an intermediate table and then removing duplicates from the intermediate table.

##### OR Processing: Indexes

Indexes on both prodnum and price, choose them or scan table

```
SQL> SELECT prodnum, name, price FROM product WHERE prodnum=2110 OR
 price=215.99;
 Prodnum Name Price
 ----- -----
 1099 typing test 215.99
 1108 blood & guts 215.99
 2110 landlord logs 815.99
```

In cost-based system, it's important that table statistics be up to date. Otherwise, optimizer may make wrong choice.

##### OR Processing: No Indexes

No indexes → must do table scan.

```
SQL> SELECT prodnum, name, price, weight FROM product WHERE name='landlord
 logs' OR weight = 1
 Prodnum Name Price Weight
 ----- -----
 2113 bugbane 44.00 1.00
 2110 landlord logs 815.99 2.50
```

#### 44.4.11 Finding Sets of Values with IN

Many 3<sup>rd</sup> party front-end applications overuse IN, sometimes employing it to find a single value or to return values in a range. This can be a problem, as IN is a form of OR processing, and tends to be expensive. If you are lucky, an IN with multiple terms can be re-stated as a range. This works only when the elements in the IN are the only ones in the range. If there were a \$200 price in the second query, the BETWEEN translation wouldn't be equivalent to the IN version.

This SQL Statement is wrong

```
SQL> SELECT prodnum, price
 FROM product
 WHERE price IN (400.00)
 1115.99
 AND 400.00
```

This is a valid statement

```
SQL> SELECT prodnum, price
 FROM product
 WHERE price = 400.00
 SQL> SELECT prodnum, price
 FROM product
 WHERE price BETWEEN
 1115.99
 AND 400.00
```

Check IN clauses carefully. Could you do the same work with a simple equals comparison? With a range?

#### 44.4.12 Using Multicolumn Indexes

In many cases, indexes consist of two or more columns. The ordprodix index in orderdetail is an example: it includes the ordnum and prodnum columns, in that order, and the order matters. The optimizer can use the index to find ordnum or combinations of ordnum and prodnum, but it can't follow pointers to go directly to prodnum values.

Think of a phone book entry. It helps you find all the subscribers surnamed Smith and all the Smiths with Heather as a first name. It's no good for finding people with an unknown last name whose first name is Heather, though.

##### Multicolumn Indexes

Could use ordprodix index.

```
SQL> SELECT ordnum,prodnum,unit FROM orderdetail WHERE ordnum = 87
 ordnum prodnum unit

 87 1083 1
 87 1105 20
 87 1106 1
 87 1794 1
 87 2000 20
 5 rows
SQL> SELECT ordnum,prodnum FROM orderdetail WHERE ordnum=87 AND prodnum =
1083
 ordnum prodnum

 87 1083
 1 row
Would not use ordprodix index
SQL> SELECT ordnum,prodnum,unit FROM orderdetail WHERE prodnum = 83
 ordnum prodnum unit

 87 1083 1
 86 1083 7
 94 1083 5
 95 1083 2
```

When you construct WHERE clauses, keep in mind the order of the columns in the index. Order of columns in the SELECT list has no effect.

#### 44.4.13 Creating Covering Indexes

If you take multicolumn index awareness a step further, you can greatly boost performance in some situations: when all the data you need is in index, the optimizer may retrieve it from the index without going to data pages.

This index is sometimes called a "covering" index and can be a big plus. Why? Because index entries are shorter than data rows and there are more of them on a page; the SQL engine can get to the covering index faster than to the data.

To use a covering index, you must meet the following conditions.

1. Every column in the SELECT list and every column in every other clause (WHERE, ORDER BY, GROUP BY, HAVING) must be in the index.

2. The index must not be disabled by functions or math or conversions—or any of the constructions covered earlier in this chapter.
3. The WHERE clause must use columns in the order in which they appear in the index (it can go directly to a specific ordnum or combination of ordnum and prodnum but not to prodnum alone, as discussed in "Using Multicolumn Indexes" at the end of the previous section).

For example, to resolve a query including only the orderdetail ordnum and prodnum columns, everything you're asking for (prodnum and ordnum) is in the ordprodix index in ordnum order. If you specify ordnum or ordnum and prodnum in the WHERE clause, the SQL engine can find the index rows that contain that value and return results from there, never touching the data pages.

```
SQL> SELECT prodnum, ordnum FROM orderdetail WHERE ordnum = 84
 prodnum ordnum

 1099 84
 1255 84
 2050 84
3 rows
```

A less efficient variation results when all the columns in the query are in index but you specify only a no leading column (prodnum) in the WHERE clause. The SQL engine may resolve the query by reading entire index sequentially. In some performance monitoring tools, this is identified as index scan, parallel to a table scan but faster.

```
SQL> SELECT prodnum, ordnum FROM orderdetail WHERE prodnum = 1099
 prodnum ordnum

 1099 84
 1099 89
2 rows
```

But back to the leading column situation: if you add a column that is not part of the index (here unit), the optimizer will still be able to use the index, but it'll have to get the nonindex values from the data pages. The same thing happens when you add nonindex conditions to the WHERE clause. The index no longer "covers" the query.

### Covering Indexes

SQL> SELECT prodnum,ordnum,unit FROM orderdetail WHERE ordnum = 84  
 ordprodix index doesn't cover query when nonindex columns are in the SELECT or WHERE clause.

```
SQL> SELECT prodnum,ordnum,unit FROM orderdetail
 WHERE ordnum = 84 AND unit = 1

 prodnum ordnum unit

 1099 84 1
 1255 84 1
 2050 84 1
```

You can cover queries by understanding your indexes and avoiding adding extra columns to your SELECT list. Don't do a SELECT \* if all you want is the product number! Remember, the index covers the query only if everything in the SELECT and WHERE clauses is in the index, if you haven't disabled the index, and if you pay attention to order in multicolumn indexes.

### 44.4.14 Joining Columns

All the previous hints on WHERE clause construction also apply to joins. Watch out for joins with functions or math on one or both sides. Avoid datatype mismatches, even subtle ones. In addition, experiment with redundant joins. For example, if you join three tables, two joins are adequate. Adding a third may give the optimizer more options. In this example, backorder is a

new table, similar to orderdetail, but it has one more column.

```
SQL> CREATE TABLE backorder
 (ordnum int not null, prodnum int not null, unit smallint not null,
 shipdate date null, backnotedate date not null)
```

table created

```
SQL> CREATE UNIQUE INDEX bkopix ON backorder(ordnum, prodnum)
```

index created

```
SQL> INSERT INTO backorder
SQL> SELECT ordnum, prodnum, unit, shipdate, '1999-09-14'
 FROM orderdetail WHERE shipdate IS NULL
15 rows
```

The standard join between three tables looks like this:

```
SQL> SELECT om.ordnum FROM ordermaster om, orderdetail od, backorder bo
 WHERE om.ordnum = od.ordnum AND od.ordnum = bo.ordnum AND om.ordnum
 = 81
```

Adding one more loop to complete the circle may help some optimizers by providing more choices. The results of the two queries are the same.

```
SQL> SELECT om.ordnum FROM ordermaster om, orderdetail od, backorder bo
 WHERE om.ordnum = od.ordnum AND od.ordnum = bo.ordnum
 AND bo.ordnum = om.ordnum AND om.ordnum = 81
 ordnum

 81
 81
 81
3 rows
```

#### 44.4.15 Sorting with DISTINCT and UNION

Sorting shows up in a number of places in addition to the ORDER BY clause. Here, check how it works in DISTINCT, UNION, and WHERE.

##### 44.4.13.1 DISTINCT

Some SQL users automatically throw a DISTINCT into every query. Don't do it unless you really need to get rid of duplicates! DISTINCT means that result are generated and then sorted.

```
SQL> SELECT DISTINCT state FROM supplier
Estimate 3 I/O operations
Summarize Subquery 1 grouping by supplier, state
Subquery1:
Estimate 3 I/O operations
Temporary table on (supplier, state)
Scan supplier sequentially
Estimate getting here 7 times
state

CA
CA
(NULL
BY
WA
MA
PA
7 rows
```

```
SQL> SELECT state FROM supplier
```

Estimate 1 I/O operations

Scan supplier sequentially

Estimate getting here 7 times

In addition, DISTINCT can serve as a cover-up for "broken" queries. Don't reach for a DISTINCT every time you get multiples instead of singles. Think through the logic first. Make sure your query is really asking the question you have in your mind.

When DISTINCT means more work, and no real increase in result coherence, do without it. This doesn't mean to avoid DISTINCT it's a very useful element. It just means to weigh the cost against the advantage.

#### 44.4.13.2 UNION

In the same way, use UNION ALL rather than UNION unless you need to eliminate duplicates. UNION returns rows from each query included in the statement, puts them in a worktable, and then sorts them to remove duplicates. UNION ALL skips the last step. Here's a UNION ALL query and the ASA PLAN it generated:

```
SQL> SELECT plan ('SELECT name, state FROM supplier
 UNION ALL SELECT lname, state FROM customer ')
```

Estimate 5 I/O operations

Take all rows from Subquery1, Subquery2

Subquery1:

Estimate 1 I/O operations

Scan supplier sequentially

Estimate getting here 7 times

Subquery2:

Estimate 3 I/O operations

Scan customer sequentially

Estimate getting here 12 times

Following is the plan for the same query with the ALL removed. Without knowing anything about PLAN messages, it's clear that UNION takes more processing than UNION ALL. An additional subquery is listed, for example.

Estimate 11 I/O operations

Summarize Subquery1 grouping by expr,expr

Subquery1:

Estimate 11 I/O operations

Temporary table on (expr,expr)

Take all rows from Subquery2,Subquery3

Subquery2:

Estimate 1 I/O operations

Scan supplier sequentially

Estimate getting here 7 times

Subquery3:

Estimate 3 I/O operations

Scan customer sequentially

Estimate getting here 12 times

In this case, the two queries actually produce the same results. You see differences only if the UNIONed queries contain duplicate result rows.

| name               | state  |
|--------------------|--------|
| Connectix Co.      | CA     |
| Soft Stuff         | CA     |
| Total Recall       | (NULL) |
| Hi Finance!        | NY     |
| TrendMaster        | WA     |
| Above Average Arts | MA     |
| Emu Sister Prdctns | PA     |
| McBaird            | CA     |
| aziz               | MA     |
| khandasamy         | NY     |
| mokoperto          | MA     |
| Peters             | NY     |
| WONG               | MD     |
| archer             | CA     |
| le blanc           | MA     |
| sato               | WA     |
| deathmask-z        | MA     |
| rs                 | TX     |
| Menendez           | NY     |

19 rows

#### 44.4.16 WHERE

Some Oracle references suggest avoiding a sort by using a meaningless condition on a WHERE clause, forcing the data into some order. Let's say you want to see customers in last-name order. There is a unique index on customer number. You add one (not unique, given the nature of names) on lname fname. In this section, there is a leading space on lname rs and fname SAM.

```
SQL> CREATE INDEX custnmix ON customer(lname, fname);
```

Index created.

A query with no WHERE or ORDER BY clause shows data in this order:

```
SQL> SELECT lname, fname, custnum FROM customer;
LNAME FNAME CUSTNUM
----- ----- -----
McBaird geoff lowell 111222222
archer ruby 111223333
aziz phillip 111333333
le blanc felipe 111334444
sato kimiko 111444444
khandasamy SAM 223456789
deathmask-z merit 777777777
mokoperto pete pete 776677778
rs Pete 776667778
Peters Pete 776667778
Menendez lauren 923457789
WONG LI-REN 999456789
```

12 rows selected.

Without using an ORDER BY, add a WHERE clause on the column in question (lname), assuming that there are no conflicting WHERE conditions that call for some other index and the indexed column does not use nulls. Oracle (and some other systems) may produce results in lname order. If you get the results you want, use your performance tools to measure the difference between this technique and using ORDER BY. However, this technique can fail if the indexes change. Be sure to document it and note it's a trick.

```
SQL> SELECT lname, fname, custnum FROM customer WHERE lname >= ' ';
LNAME FNAME CUSTNUM
----- ----- -----
rs pete pete 776677778
McBaird geoff lowell 111222222
Menendez lauren 923457789
Peters Pete 776667778
WONG LI-REN 999456789
archer ruby 111223333
aziz phillip 111333333
deathmask-z SAM 223456789
khandasamy felipe 111334444
le blanc merit 777777778
mokoperto kimiko 111444444
```

12 rows selected.

#### 44.4.17 Choosing between HAVING and WHERE

WHERE puts conditions on the table rows, determining which should be returned. HAVING puts conditions on grouped result rows. The processing order is as follows:

```
Select rows with WHERE.
Divide rows into sets with GROUP BY.
Calculate aggregate values for each group.
Eliminate unwanted group result rows with HAVING.
Any rows you can remove with WHERE, rather than HAVING, make your query
more efficient. There are fewer rows to group and fewer to aggregate. In
it makes sense to remove books before rather than after grouping and
counting. You save a lot of work.
WHERE and HAVING
This is a wrong SQL Statement
Statement
This is a valid SQL
```

```

SQL> SELECT type, count(*)
 FROM product
 GROUP BY type
 HAVING type <> 'book'

SQL> SELECT type, count(*)
 FROM product
 WHERE type <> 'book'
 GROUP BY type

type count(*)

application 8
game 4
education 4
hardware 2

4 rows
Use HAVING to limit group result rows, as in the following query:
SQL> SELECT type, count(*) FROM product GROUP BY type HAVING COUNT(*) > 5

type count(*)

application 8

1 row

```

#### 44.4.18 Looking at Views

When you write a query on a view, the view is translated into its underlying SELECT statements. If you need information from one table but go through a multitable view to get it, you'll pay a price.

```

SQL> CREATE view ordervu AS SELECT od.ordnum, od.prodnum,
 substr(p.name, 1, 20) AS name, p.price * od.unit AS cost
 FROM orderdetail od, product p WHERE od.prodnum = p.prodnum

view created
To find the name of a particular product, you might write a query like
this:
SQL> SELECT distinct prodnum, name FROM ordervu WHERE prodnum = 2050

prodnum name

2050 tax time

1 row

```

It gives you exactly the results you want, but a look at the PLAN output is daunting.

Estimate 13 I/O operations

Summarize Subquery1 grouping by orderdetail.prodnum,expr

Subquery1:

Estimate 13 I/O operations (best of 2 plans considered)

Temporary table on (orderdetail.prodnum,expr)

Scan product AS p sequentially

Estimate getting here 21 times

Scan orderdetail AS od sequentially

Estimate getting here 693 times

Removing DISTINCT cuts down on the processing a bit, but it also returns multiple copies of the product number and name. It shows one row for every order that includes the product number 2044.

```
SQL> SELECT prodnum, name FROM ordervu WHERE prodnum = 2050

 prodnum name
 ----- -----
 2050 tax time
 2050 tax time

 7 rows
```

Estimate 5 I/O operations (best of 2 plans considered)

Scan product AS p sequentially

Estimate getting here 21 times

Scan orderdetail AS od sequentially

Estimate getting here 693 times

Compare the ordervu view PLANS to the product table PLAN for essentially the same query.

```
SQL> SELECT prodnum, name FROM product WHERE prodnum = 2050

 prodnum name
 ----- -----
 2050 tax time
```

Estimate 4 I/O operations

Scan product using unique index prodix for rows where prodnum equals 2050

Estimate getting here 1 times

Don't use a view if all the data you want is in a single underlying table, and don't make frequently queried views unnecessarily complex.

#### 44.4.19 Forcing Indexes

Many systems permit overriding the optimizer's choices by forcing use of a specified index. This is a good tool for testing, particularly when you are exploring how the optimizer behaves. It gives you a way to see how different indexes work with the same data.

However, hardwiring your queries to use a particular index is dangerous. You lose flexibility. When the data changes, you're tied to the index that worked best last Tuesday—quite possibly not the right choice today. If you decide to force index choices, be sure to document the decision and the cause in the code, and plan to check the SQL regularly.

There are plenty of stories about queries that were slow until an expert was called in. Performance improved! . . . for a while . . . and then dropped to abysmal. Another expert removed the index forcing elements, cleaned up some messy WHERE clauses, avoided a view, eliminated a sort, and ended up with longer-lasting improvements.

## 44.5 SQL Tuning Enhancements in 12c

### 44.5.1 Adaptive SQL Plan Management (SPM)

The SPM Evolve Advisor is a task infrastructure that enables you to schedule an evolve task, rerun an evolve task, and generate persistent reports. The new automatic evolve task, **SYS\_AUTO\_SPM\_EVOLVE\_TASK**, runs in the default maintenance window. This task ranks all unaccepted plans and runs the evolve process for them. If the task finds a new plan that performs better than existing plan, the task automatically accepts the plan. You can also run evolution tasks manually using the **DBMS\_SPM** package.

#### 44.5.1.1 Adaptive query optimization

Adaptive query optimization is a set of capabilities that enable the optimizer to make run-time adjustments to execution plans and discover additional information that can lead to better statistics. The set of capabilities include:

#### 44.5.1.2 Adaptive plans

An **adaptive plan** has built-in options that enable the **final plan** for a statement to differ from the **default plan**. During the first execution, before a specific subplan becomes active, the optimizer makes a final decision about which option to use. The optimizer bases its choice on observations made during the execution up to this point. The ability of the optimizer to adapt plans can improve query performance.

#### 44.5.1.4 Automatic re-optimization

When using automatic optimization, the optimizer monitors the initial execution of a query. If the actual execution statistics vary significantly from the original plan statistics, then the optimizer records the execution statistics and uses them to choose a better plan the next time the statement executes. The database uses information obtained during automatic re-optimization to generate SQL plan directives automatically.

#### 44.5.1.4 SQL plan directives

In releases earlier than Oracle Database 12c, the database stored compilation and execution statistics in the shared SQL area, which is non-persistent. Starting in this release, the database can use a SQL plan directive, which is additional information and instructions that the optimizer can use to generate a more optimal plan. The database stores SQL plan directives persistently in the **SYSAUX** tablespace. When generating an execution plan, the optimizer can use SQL plan directives to obtain more information about the objects accessed in the plan.

#### 44.5.1.5 Dynamic statistics enhancements

In releases earlier than Oracle Database 12c, Oracle Database only used dynamic statistics (previously called *dynamic sampling*) when one or more of the tables in a query did not have optimizer statistics. Starting in this release, the optimizer automatically decides whether dynamic statistics are useful and which dynamic statistics level to use for all SQL statements. Dynamic statistics gathers are persistent and usable by other queries.

## 44.6 New types of histograms

This release introduces top frequency and hybrid histograms. If a column contains more than 254 distinct values, and if the top 254 most frequent values occupy more than 99% of the data, then the database creates a top frequency histogram using the top 254 most frequent values. By ignoring the non-popular values, which are statistically insignificant, the database can produce a better quality histogram for highly popular values. A hybrid histogram is an enhanced height-based histogram that stores the exact frequency of each endpoint in the sample, and ensures that a value is never stored in multiple buckets.

Also, regular frequency histograms have been enhanced. The optimizer computes frequency histograms during NDV computation based on a full scan of the data rather than a small sample (when AUTO\_SAMPLING is used). The enhanced frequency histograms ensure that even highly infrequent values are properly represented with accurate bucket counts within a histogram.

## 44.7 Monitoring database operations

Real-Time Database Operations Monitoring enables you to monitor long running database tasks such as batch jobs, scheduler jobs, and Extraction, Transformation, and Loading (ETL) jobs as a composite business operation. This feature tracks the progress of SQL and PL/SQL queries associated with the business operation being monitored. As a DBA or developer, you can define business operations for monitoring by explicitly specifying the start and end of the operation or implicitly with tags that identify the operation.

### 44.7.1 Concurrent statistics gathering

You can concurrently gather optimizer statistics on multiple tables, table partitions, or table subpartitions. By fully utilizing multiprocessor environments, the database can reduce the overall time required to gather statistics. Oracle Scheduler and Advanced Queuing create and manage jobs to gather statistics concurrently. The scheduler decides how many jobs to execute concurrently, and how many to queue based on available system resources and the value of the `JOB_QUEUE_PROCESSES` initialization parameter.

### 44.7.2. Reporting mode for `DBMS_STATS` statistics gathering functions

You can run the `DBMS_STATS` functions in reporting mode. In this mode, the optimizer does not actually gather statistics, but reports objects that would be processed if you were to use a specified statistics gathering function.

### 44.7.3 Reports on past statistics gathering operations

You can use `DBMS_STATS` functions to report on a specific statistics gathering operation or on operations that occurred during a specified time.

### 44.7.4 Automatic column group creation

With column group statistics, the database gathers optimizer statistics on a group of columns treated as a unit. Starting in Oracle Database 12c, the database automatically determines which column groups are required in a specified workload or SQL tuning set, and then creates the column groups. Thus, for any specified workload, you no longer need to know which columns from each table must be grouped.

### 44.7.5 Session-private statistics for global temporary tables

Starting in this release, global temporary tables have a different set of optimizer statistics for each session. Session-specific statistics improve performance and manageability of temporary tables because users no longer need to set statistics for a global temporary table in each session or rely on dynamic statistics. The possibility of errors in cardinality estimates for global temporary tables is lower, ensuring that the optimizer has the necessary information to determine an optimal execution plan.

### 44.7.6 SQL Test Case Builder enhancements

SQL Test Case Builder can capture and replay actions and events that enable you to diagnose incidents that depend on certain dynamic and volatile factors. This capability is especially useful for parallel query and automatic memory management.

#### 44.7.7 Online statistics gathering for bulk loads

A **bulk load** is a CREATE TABLE AS SELECT or INSERT INTO ... SELECT operation. In releases earlier than Oracle Database 12c, you needed to manually gather statistics after a bulk load to avoid the possibility of a suboptimal execution plan caused by stale statistics. Starting in this release, Oracle Database gathers optimizer statistics automatically, which improves both performance and manageability.

#### 44.7.8 Reuse of synopses after partition maintenance operations

**ALTER TABLE EXCHANGE** is a common partition maintenance operation. During a partition exchange, the statistics of the partition and the table are also exchanged. A **synopsis** is a set of auxiliary statistics gathered on a partitioned table when the **INCREMENTAL** value is set to **true**. In releases earlier than Oracle Database 12c, you could not gather table-level synopses on a table. Thus, you could not gather table-level synopses on a table, exchange the table with a partition, and end up with synopses on the partition. You had to explicitly gather optimizer statistics in incremental mode to create the missing synopses. Starting in this release, you can gather table-level synopses on a table. When you exchange this table with a partition in an incremental mode table, the synopses are also exchanged.

#### 44.7.9 Automatic updates of global statistics for tables with stale or locked partition statistics

Incremental statistics can automatically calculate global statistics for a partitioned table even if the partition or sub-partition statistics are stale and locked.

#### 44.7.10 Cube query performance enhancements

These enhancements minimize CPU and memory consumption and reduce I/O for queries against cubes.

### 44.10 SQL Tuning Sets

A **SQL tuning set (STS)** is a database object that includes:

A set of SQL statements

Associated execution context, such as user schema, application module name and action, list of bind values, and the environment for **SQL compilation** of the cursor

Associated basic execution statistics, such as elapsed time, CPU time, buffer gets, disk reads, rows processed, cursor fetches, the number of executions, the number of complete executions, optimizer cost, and the command type

Associated execution plans and row source statistics for each SQL statement (optional)

The database stores SQL tuning sets in a database-provided schema.

An STS enables you to group SQL statements and related metadata in a single database object, which you can use to meet your tuning goals. Specifically, SQL tuning sets achieve the following goals:

Providing input to the performance tuning advisors

You can use an STS as input to multiple database advisors, including SQL Tuning Advisor, SQL Access Advisor, and SQL Performance Analyzer.

Transporting SQL between databases

You can export SQL tuning sets from one database to another, enabling transfer of SQL workloads between databases for remote performance diagnostics and tuning. When suboptimally performing SQL statements occur on a production database, developers may not want to investigate and tune directly on the production database. The DBA can transport the problematic SQL statements to a test database where the developers can safely analyze and tune them.

To create an STS, you must load SQL statements into an STS from a source

SQL tuning sets can do the following:

Filter SQL statements using the application module name and action, or any execution statistics

Rank SQL statements based on any combination of execution statistics

Serve as input to the advisors or transport it to a different database

## 44.11 Adaptive Plans in Oracle Database 12c Release 1

The cost-based optimizer uses database statistics to determine the optimal execution plan for a SQL statement. If those statistics are not representative of the data, or if the query uses complex predicates, operators or joins the estimated cardinality of the operations may be incorrect and therefore the selected plan is likely to be less than optimal. In previous database releases, once the execution plan was determined there was no possible deviation from it at runtime.

Adaptive Plans in Oracle Database 12c allow runtime changes to execution plans. Rather than selecting a single "best" plan, the optimizer will determine the default plan, and can include alternative subplans for each major join operation in the plan. At runtime the cardinality of operations is checked using statistics collectors and compared to the cardinality estimates used to generate the execution plan. If the cardinality of the operation is not as expected, an alternative subplan can be used. For example, if the statistics suggest two small sets are to be joined, it is likely the optimizer will choose a nested loops join. At runtime, if the fetch operation of the first set returns more than the expected number of rows, the optimizer can switch to a subplan using a hash join instead. This same adaptation can happen for every join operation in the query if an alternative subplan is present. Once the query has run to completion and the optimal plan is determined, the final plan is fixed until it is aged out of the shared pool or reoptimized for some other reason? The statistics collectors can also be used to influence the parallel distribution method used for parallel queries.

Adaptive Join Method

Adaptive Parallel Distribution Method

Miscellaneous Points

### 44.15.1 Adaptive Join Method

It is important to remember the adaptive join method functionality is only used during the first execution of the statement, so subsequent executions will follow the final plan determined by the first execution. For adaptive plans, once the final plan is determined, the `IS_RESOLVED_ADAPTIVE_PLAN` column of the `V$SQL` view will be marked as "Y".

In this example, the data is created to favour a nested loops join and statistics are gathered.

### 44.11.2 Adaptive Parallel Distribution Method

For a parallel query, the optimizer must decide how rows are distributed (broadcast or hash for the left input, round-robin or hash for the right input) to slaves between one operation and the next. Picking the wrong distribution method can have an impact on the performance of the parallel query. Oracle Database 12c introduces an adaptive parallel distribution method called hybrid hash, where the decision on the distribution method is delayed until execution time, based on the results of statistics collectors. Unlike the adaptive join method, which is limited to the first

execution, the adaptive parallel distribution method is used for each execution of the statement.

The hybrid hash adaptive distribution method assumes a hash distribution is required. If the number of rows returned on the producer side of the parallel operation is less than a threshold value, the distribution method is switched to broadcast. The threshold value is twice the degree of parallelism (DOP) for the query.

We can use the data from the previous example to show the presence of the hybrid hash in a parallel query. All we need to do is gather statistics again, but we will also flush the shared pool to keep things clean.

## 44.12 Summary

This chapter is about making sure your SQL is as effective as possible from a performance point of view. Optimizers vary a lot, so use this information as a starting point, and remember, there are many other factors that can influence performance.

- Make sure you understand what indexes you have.
- Next, find out what tools let you look at the optimizer's choices.
- Once you know your indexes and have a way of finding out whether or not they are being used in a particular query, check the WHERE clause in problem queries. There are many ways you can structure a WHERE clause to invalidate available queries. Often, you can fix a performance problem by making a simple change.
- Indexes that cover queries (contain all the information you need for the results) can be very useful. There are some strict rules to follow, however, in both creating and using them.
- Good joins are critical to relational database performance. Joins have many of the strictures of ordinary WHERE clauses.
- Sorting always means another pass: get the data, then put it in order. Understand when you may be doing unnecessary sorting (with DISTINCT and UNION) and avoid it.
- HAVING and WHERE are not the same. WHERE eliminates rows before you form groups. HAVING limits the group results. Doing WHERE work in HAVING can slow performance down.
- Views are handy, but they may be expensive. Understand the cost of view queries, and create views accordingly.
- Many systems provide ways to go around the optimizer in index choice. Be cautious when you do this.

## 45. Application Tuning

In Tuning of SQL Statements we use 2 types of tools:

- TKPROF (Usually at the Back-End Level)
- EXPLAIN PLAN (Usually at the Front-End Level / also can be done at backend)

Tkprof can give us exact statistics about how long this SQL-Stmt took to run (in Sec), and also what was the "Access Path" it has used. So we can understand exactly what went thru our Oracle-Engine when our SQL-Stmt is fired-up.

The Syntax is:

```
SQL> ALTER SESSION SET TIMED_STATISTICS=true;
SQL> ALTER SESSION SET SQL_TRACE=true;
SQL> SELECT * FROM emp WHERE empno = 1001; # this is our SQL-Stmt
```

After performing the above command, we need to go to our UDUMP dir, and see if a brand-new Trace file has generated (since we said sql\_trace=true). Once we identify our Trace file, we need to run "TKPROF" utility on top of this Trace file (since we can't directly understand this Trace file).

```
E.g. $ tkprof abcd.trc junk.log explain=scott/tiger
(Here Scott/Tiger is User-Id/Passwd)
```

So this TKPROF utility is again asking our Oracle-Engine about the content of the "abcd.trc" file, and writing the output to "junk.log". This output file is English-like file, which we can understand. So now go to this file (using "vi" editor), and see what kind of "Access-Path" Oracle has used for our SQL-Stmt. It also tells us how long these took in sec. and how many rows were retrieved and so on. This is the greatest utility we can ever think of, since without knowing what is happening at the Oracle-Instance level, we can't pinpoint where the problem is.

Now that we know what kind of "Access-Path" is used, maybe we can modify our "SQL-Stmt", and try to "IMPROVE" the Performance. The great thing about this utility is, we can even "TURN-ON" at the "Database" Level, so that whatever "SQL" is going thru our Database, Oracle is going to generate "TRC" files, which can be visited later and see what's happening.

Basically this is useful in case if we don't have the "Source-Code" (since we purchased a 3rd party software, which comes only in Binary fashion), then it would be very difficult in finding out where we are having this performance problem (again, because we haven't written the SQL). In such cases, we simply put 2 lines in our INIT.ORA and shut and start the database.

```
SQL_TRACE=true
TIMED_STATISTICS=true
```

From this point, ask all users not to connect (otherwise, since the sql\_trace is true for every session, everyone will be generating TRC files automatically), but only run our specific FORM or REPORT against the Database. This should generate a TRC file. Now use "TKPROF" and try to see what this "FORM or REPORT" is trying to do. In case if we find any problem in Oracle's "Access-Path", or if we feel like just by creating an Index on appropriate Table, may be this performance problem can be resolved.

But once the issue is taken care, make sure we comment the line in our INIT.ORA "sql\_trace=true". We may leave the other parameter, since it is only activated if "sql\_trace=true" is running or a programmer is using "alter session set sql\_trace=true". If we don't take out this "sql\_trace=true" from INIT.ORA, then for everything that is going thru our Database, Oracle is going to generate TRC files, which is an extra burden for Oracle, and at the same time, this may fill up our file-system. There are basically 14 "Access-Paths" defined in Oracle. Please see our Lab-Notes for more details.

## 45.1 Explain-Plan

One disadvantage in using "TKPROF" is, we need to wait until our "SQL-Stmt" is completed. Which means I have to run my SQL and then go to this TRC file and analyze the file using "TKPROF". Sometimes this is stupid because may be our SQL is taking more than 2 hours.

In such case, I would rather like to know what Access-Path Oracle is using more than how many seconds its taking. Right here, we have another way of getting it by using "EXPLAIN-PLAN" In Explain-Plan; we can't get the timings like we do in TKPROF. We are giving our SQL-Stmt as the input to Oracle and asking it to give the "Access-Path". Oracle, in turn, is dumping the Access-Path in a table called "PLAN\_TABLE", which should exist prior to our "Explain-Plan" request. This table can be created by running \$O\_H/rdbms/admin/utlxplan.sql. The syntax is very easy for asking Oracle to give "Explain-Plan". All we have to do is:

```
SQL> EXPLAIN PLAN
 SET statement_id='Some-Name'
 FOR
 SELECT * FROM emp WHERE empno = 1001;
```

Oracle is not going to "EXECUTE" our SQL-Stmt, instead it's giving the "Action-Plan" by writing into "Plan\_Table". So now we go to Plan\_Table and we can use SELECT to get the information from it. We can't really say which is the best tool (whether TKPROF or EXPLAIN-PLAN), since they are doing the same thing differently.

If we want to know how much exact time this SQL-Stmt took with how many rows, then TKPROF is the only choice, but we are just interested in knowing the "Access-Path", then EXPLAIN-PLAN is better. For Explain-Plan, we don't need INIT.ORA parameters, whereas for TKPROF we need "timed\_statistics=true" parameter.

## 45.2 SQL Trace

The SQL Trace facility provides performance information on individual SQL statements. It generates the following statistics for each statement:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback
- Wait event data for each SQL statement, and a summary for each trace file

Although it is possible to enable the SQL Trace facility for a session or for an instance, it is recommended that we use the DBMS\_SESSION or DBMS\_MONITOR packages instead. When the SQL Trace facility is enabled for a session or for an instance, performance statistics for all SQL statements executed in a user session or in the instance are placed into trace files.

Using the SQL Trace facility can have a severe performance impact and may result in increased system overhead, excessive CPU usage, and inadequate disk space.

### Enable the SQL Trace facility for the session by using one of the following:

```
SQL> DBMS_SESSION.SET_SQL_TRACE procedure
SQL> ALTER SESSION SET SQL_TRACE = TRUE;
```

Caution: Because running the SQL Trace facility increases system overhead, enable it only when tuning SQL statements, and disable it when we are finished. It is recommended that we use the DBMS\_SESSION or DBMS\_MONITOR packages to enable SQL tracing for a session or an instance instead.

We might need to modify an application to contain the ALTER SESSION statement. For example, to issue the ALTER SESSION statement in Oracle Forms, invoke Oracle Forms using the `-s` option, or invoke Oracle Forms (Design) using the statistics option.

**To disable the SQL Trace facility for the session, enter:**

```
SQL> ALTER SESSION SET SQL_TRACE = FALSE;
```

The SQL Trace facility is automatically disabled for the session when the application disconnects from Oracle. We can enable the SQL Trace facility for an instance by setting the value of the `SQL_TRACE` initialization parameter to `TRUE` in the initialization file.

```
SQL> SQL_TRACE = TRUE
```

After the instance has been restarted with the updated initialization parameter file, SQL Trace is enabled for the instance and statistics are collected for all sessions. If the SQL Trace facility has been enabled for the instance, we can disable it for the instance by setting the value of the `SQL_TRACE` parameter

## 46. AUTOTRACE

The SQL Trace facility and TKPROF let us accurately assess the efficiency of the SQL statements an application runs. For best results, use these tools with EXPLAIN PLAN rather than using EXPLAIN PLAN alone.

The SQL Trace facility provides performance information on individual SQL statements. It generates the following statistics for each statement:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback
- Wait event data for each SQL statement, and a summary for each trace file

If the cursor for the SQL statement is closed, SQL Trace also provides row source information that includes:

Row operations showing the actual execution plan of each SQL statement

Number of rows, number of consistent reads, number of physical reads, number of physical writes, and time elapsed for each operation on a row

Although it is possible to enable the SQL Trace facility for a session or for an instance, it is recommended that we use the DBMS\_SESSION or DBMS\_MONITOR packages instead. When the SQL Trace facility is enabled for a session or for an instance, performance statistics for all SQL statements executed in a user session or in the instance are placed into trace files.

Using the SQL Trace facility can have a severe performance impact and may result in increased system overhead, excessive CPU usage, and inadequate disk space.

We can control the report by setting the AUTOTRACE system variable.

|                             |                                                                                                             |
|-----------------------------|-------------------------------------------------------------------------------------------------------------|
| SET AUTOTRACE OFF           | No AUTOTRACE report is generated. This is the default.                                                      |
| SET AUTOTRACE EXPLAIN ON    | The AUTOTRACE report shows only the optimizer execution path.                                               |
| SET AUTOTRACE STATISTICS ON | The AUTOTRACE report shows only the SQL statement execution statistics.                                     |
| SET AUTOTRACE ON            | The AUTOTRACE report includes both the optimizer execution path and the SQL statement execution statistics. |
| SET AUTOTRACE RACEONLY      | Like SET AUTOTRACE ON, but suppresses the printing of the user's query output, if any.                      |

### Example: Tracing Statements for Performance Statistics and Query Execution Path

If the SQL buffer contains the following statement:

## 47. Memory Tuning

This is one of the more challenging and critical aspects of the DBA job: **analyzing, diagnosing, and fixing database internals performance problems**. The application tuning. We will get a majority of the performance gains in an application from proper database configuration and application tuning. However, where you will be most exposed will be in the area of internals tuning. Squeezing that last bit of performance from the database seems to be the one area managers like to focus on when there are problems.

### Steps to Internals Tuning

As said at the end of the last chapter, once the application has been tuned, the DBA's job really begins. Now you can begin tuning the Oracle system itself to take advantage of the tuned application. This step of the tuning process is typically a five-part process:

- Review and set all initialization parameters for your application and operating system.
- Tune memory allocation.
- Eliminate I/O bottlenecks.
- Tune resource contention.
- Tune sorts, freelists, and checkpoints.

**Step 1:** Involves reading the operating system specific release manual and database readme files for any new, changed, or improved initialization parameters. Using your knowledge of the number of users, size of the system memory, number and configuration of disks, sizing of tables, and other system and application parameters, you must do your best to set all of the initialization parameters that will help your system perform better.

Parameters can be reviewed by looking at initialization file or by querying **V\$PARAMETER** file. Perform a control file dump using the command:

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

**Step 2:** Requires an up-and-operating database against which you run various performance monitoring scripts and tools; then you readjust the initialization parameters. You should also examine database and session level waits using queries against **V\$WAITSTAT** and **V\$SESSION\_WAIT** dynamic performance views.

**Step 3:** The requires monitoring your disk assets and their performance. Your system administrator will be critical to assuring the success of this step. Hopefully, if you were able to have a hand in designing the system layout, you won't have much I/O-related tuning. An inherited database (especially those from aftermarket products) usually requires extensive file movements and optimizations, so this step could actually give the most performance gains.

In one system you inherited, a well-meaning DBA had rebuilt the application indexes, by disabling and then re-enabling the primary keys, without specifying the location for the indexes. Of course, you will remember what this causes: all of the indexes were in the same tablespace as the data tables. Simply moving the indexes to their (empty) tablespace resulted in an over 300 percent performance gain (one 30-minute query dropped to less than a minute).

You were an instant hero. What this story should tell you is to carefully examine any inherited database for badly placed indexes, tablespaces, rollback segments, and redo logs. Just putting everything where it should be can provide dramatic improvements in performance for badly laid-out systems.

**Step 4:** Involves more monitoring with tools or scripts. Contention for system resources (latches, rollbacks, logs, memory, etc.) can be a real performance drain. Always review the alert log for all databases you inherit, as they will tell you if there is some forms of contention such as for redo logs. The scripts that follow will help determine if there are other types of contention.

**Step 5:** Involve monitoring system statistics on a running application. There are numerous tools, as well as the scripts included, that will tell you if you have problems with sorts, freelists, and

checkpoints. Tuning sorts is especially important in DSS and reporting databases. In one case, a 10-minute sort dropped to less than a minute by bumping up the SORT\_AREA\_SIZE parameter 2 to 3 megabytes, thus preventing disk sorts.

## 47.1 Flash Cache

New in Oracle Database 11g Release 2 (11.2), the Database Smart Flash Cache feature is a transparent extension of the database buffer cache using solid state device (SSD) technology.

The SSD acts as a Level 2 cache to the (Level 1) SGA.

Database Smart Flash Cache can greatly improve the performance of Oracle databases by reducing the amount of disk I/O at a much lower cost than adding an equivalent amount of RAM.

### 47.1.1 Tuning Memory for the Flash Cache

For each database block moved from the buffer cache to the flash cache, a small amount of metadata about the block is kept in the buffer cache. For a single instance database, the metadata consumes approximately 100 bytes. For an Oracle Real

Application Clusters (Oracle RAC) database, it is closer to 200 bytes. We must take this extra memory requirement into account when adding the flash cache.

**Note:** we choose to not increase the buffer cache size to account for the flash cache. In this case, the effective size of the buffer cache is reduced. However, we can offset this loss by using a larger flash cache.

For each database block moved from the buffer cache to the flash cache, a small amount of metadata about the block is kept in the buffer cache

- If we are managing memory manually, increase the size of the buffer cache by an amount approximately equal to the number of database blocks that fit into the flash cache multiplied by 100 (or 200 for Oracle RAC).
- If we are using automatic memory management, increase the size of MEMORY\_TARGET
- using the algorithm described above. We may first have to increase the size of MEMORY\_MAX\_TARGET.
- If we are using automatic shared memory management, increase the size of SGA\_TARGET.
- For the MEMORY\_MAX\_TARGET initialization parameter, decide on a maximum amount of memory that we would want to allocate to the database for the foreseeable future. That is, determine the maximum value for the sum of the SGA and instance PGA sizes. This number can be larger than or the same as the
- MEMORY\_TARGET value that you chose in the previous step.

## 47.2 Configuring the In-Memory Column Store (12c Enhancement)

The In-Memory Column Store is an optional area of the SGA that stores copies of tables, partitions, and other database objects in a columnar format that is optimized for rapid scans.

The In-Memory Column Store stores copies of tables, partitions, and other database objects in the SGA. The In-Memory Column Store does not replace the database buffer cache. Instead, they complement each other so that both memory areas can store the same data in different formats. Rows stored in the In-Memory Column Store are divided into large memory regions in a columnar format. Within each region, a column resides separately in a contiguous area of memory.

You can enable the In-Memory Column Store for any of the following database objects:

Tables

Materialized views

Partitions

Tablespaces

You can choose to store all columns of a table or a materialized view in the In-Memory Column Store, or only a subset of its columns. Similarly, for a partitioned table, you can choose to store all of the table's partitions in the In-Memory Column Store, or only a subset of the partitions. Enabling the In-Memory Column Store at the tablespace level automatically enables all tables and materialized views in the tablespaces for the In-Memory Column store.

By storing database objects in memory, Oracle Database can perform scans, queries, joins, and aggregates much faster than on disk. The In-Memory Column Store can drastically improve performance when:

Scanning a large number of rows and applying filters, such as `<`, `>`, `=`, and `IN`.

Querying a small subset of columns from a large number of columns, such as selecting 5 columns from a table with 100 columns.

Joining a small table to a large table, particularly when join conditions filter most of the rows.

Aggregating data in a query.

The In-Memory Column Store also improves the performance of data manipulation language (DML) statements. Online transaction processing (OLTP) systems typically require many indexes to be created on commonly accessed columns. These indexes can have a negative performance impact on DML statements. When a database object is stored in the In-Memory Column Store, these indexes can be reduced or eliminated because scans run much faster. Reducing the number of indexes improves the performance of DML statements because fewer indexes need to be updated.

Using the In-Memory Column Store requires a great deal of memory. To reduce its memory requirements, the In-Memory Column Store enables you to compress the data it stores. Queries are then executed directly on the compressed data. You can specify the compression method for each database object to be stored in the In-Memory Column Store. The amount of memory required by the In-Memory Column Store thus depends on the database objects that you want to store in it and their individual compression methods.

On one hand, choosing a high compression method reduces the amount of memory required by the In-Memory Column Store, but does not provide the greatest performance benefits. On the other hand, choosing a low compression method provides the greatest performance benefits, but also requires more memory. In choosing the compression method for the database objects you want to store in the In-Memory Column Store, balance the performance benefits you want to achieve with the amount of available memory.

The In-Memory Column Store supports the following compression methods:

**NO MEMCOMPRESS**

This compression method does not compress data.

**MEMCOMPRESS FOR DML**

This compression method optimizes the data for DML operations and compresses In-Memory Column Store data the least.

### MEMCOMPRESS FOR QUERY LOW

This compression method results in the best query performance. This method compresses In-Memory Column Store data more than MEMCOMPRESS FOR DML but less than MEMCOMPRESS FOR QUERY HIGH.

### MEMCOMPRESS FOR QUERY HIGH

This compression method results in excellent query performance. This method compresses In-Memory Column Store data more than MEMCOMPRESS FOR QUERY LOW but less than MEMCOMPRESS FOR CAPACITY LOW.

### MEMCOMPRESS FOR CAPACITY LOW

This compression method results in good query performance. This method compresses In-Memory Column Store data more than MEMCOMPRESS FOR QUERY HIGH but less than MEMCOMPRESS FOR CAPACITY HIGH.

### MEMCOMPRESS FOR CAPACITY HIGH

This compression method results in fair query performance. This method compresses In-Memory Column Store data the most.

Enabling a Table for the In-Memory Column Store with MEMCOMPRESS FOR CAPACITY HIGH Compression

```
SQL>> ALTER TABLE oe.product_information INMEMORY MEMCOMPRESS FOR CAPACITY HIGH;
```

Once you have estimated the amount of memory needed to store each database object, you can calculate the total amount of memory required by the In-Memory Column Store as roughly the sum of the amount of memory required by all database objects that you want to store. You should also include some additional space to allow for the growth of the database objects, and to store updated versions of rows after DML operations.

### Sizing the In-Memory Column Store

After you have determined the memory required to store your database objects in the In-Memory Column Store based on their compression methods, you can set its size by using the `INMEMORY_SIZE` initialization parameter.

To set the size of the In-Memory Column Store:

Set the `INMEMORY_SIZE` initialization parameter to the required size.

The default value of this parameter is 0, which means that the In-Memory Column Store is not used. To enable the In-Memory Column Store, set this parameter to a nonzero value.

In a multitenant environment, you can set this parameter per pluggable database (PDB) to specify the size of the In-Memory Column Store for each PDB. The sum of the PDB values does not have to equal the value for the container database (CDB), and may even be greater.

### Setting the Size of the In-Memory Column Store

```
SQL>> ALTER SYSTEM SET INMEMORY_SIZE = 100G;
```

After setting the size of the In-Memory Column Store, you must restart your database instance to enable the database objects to be stored in it.

## 48. Parsing in Oracle

Whenever a statement is executed, Oracle follows a methodology to evaluate the statement in terms of syntax, validity of objects being referred and of course, privileges to the user. Apart from this, Oracle also checks for identical statements that may have been fired, with the intention of reducing processing overheads.

All this takes place in a fraction of a second, even less, without the user knowing what is happening to the statement that was fired. This process is known as Parsing.

### 48.1 Types of Parsing

All statements, DDL or DML, are parsed whenever they are executed. The only key fact is that whether it was a Soft (statement is already parsed and available in memory) or a Hard (all parsing steps to be carried out) parse.

Soft parse will considerably improve the system performance whereas frequent Hard parsing will affect the system. Reducing hard parsing will improve the resource utilization and optimize the SQL code.

### 48.2 Parsing process

Oracle internally does the following to arrive at the output of an SQL statement.

- Syntactical check. The query fired is checked for its syntax.
- Semantic check. Checks on the validity of the objects being referred in the statement and the privileges available to the user firing the statement. This is a data dictionary check.
- Allocation of private SQL area in the memory for the statement.
- Generating a parsed representation of the statement and allocating Shared SQL area. This involves finding an optimal execution path for the statement.

In point four, Oracle first checks if the same statement is already parsed and existing in the memory. If found, the parsed representation will be picked up and the statement executed immediately (Soft parse). If not found, then the parsed representation is generated and stored in a shared SQL area (Part of shared pool memory in SGA), the statement is then executed (Hard parse). This step involves the optimization of the statement, the one that decides the performance.

### 48.3 Identical statements

Oracle does the following to find identical statements to decide on a soft or a hard parse.

- When a new statement is fired, a hash value is generated for the text string. Oracle checks if this new hash value matches with any existing hash value in the shared pool.
- Next, the text string of the new statement is compared with the hash value matching statements. This includes comparison of case, blanks and comments present in the statements.
- If a match is found, the objects referred in the new statement are compared with the matching statement objects. Tables of the same name belonging to different a schema will not account for a match.
- The bind variable types of the new statement should be of same type as the identified matching statement.
- If all of the above is satisfied, Oracle re-uses the existing parse (soft). If a match is not found, Oracle goes through the process of parsing the statement and putting it in the shared pool (hard).

### 48.4 Reduce hard parsing

The shared pool memory can be increased when contention occurs, but more important is that such issues should be addressed at the coding level. Following are some initiatives that can be taken to reduce hard parsing.

- Make use of bind variables rather than hard-coding values in your statements.
- Write generic routines that can be called from different places. This will also eliminate code repetition.
- Even with stringent checks, it may so happen that same statements are written in different formats. Search the SQL area periodically to check on similar queries that are being parsed separately. Change these statements to be look-alike or put them in a common routine so that a single parse can take care of all calls to the statement.

#### Identifying unnecessary parse calls at system level

```
SQL> SELECT parse_calls, executions, substr(sql_text, 1, 300)
 FROM v$sqlarea WHERE command_type in (2, 3, 6, 7);
```

Check for statements with a lot of executions. It is bad to have the PARSE\_CALLS value in the above statement close to the EXECUTIONS value. The above query will fire only for DML statements (to check on other types of statements use the appropriate command type number). Also ignore Recursive calls (dictionary access), as it is internal to Oracle.

#### Identifying unnecessary parse calls at session level

```
SQL> SELECT b.sid, a.name, b.value FROM v$sesstat b, v$statname a
 WHERE a.name in ('parse count (hard)', 'execute count')
 AND b.statistic# = a.statistic# ORDER BY sid;
```

Identify the sessions involved with a lot of re-parsing (VALUE column). Query these sessions from V\$SESSION and then locate the program that is being executed, resulting in so much parsing.

```
SQL> SELECT a.parse_calls, a.executions, substr(a.sql_text, 1, 300)
 FROM v$sqlarea a, v$session b
 WHERE b.schema#=a.parsing_schema_id AND b.sid=<:sid> ORDER BY 1 DESC;
```

The above query will also show recursive SQL being fired internally by Oracle.

Provide enough private SQL area to accommodate all of the SQL statements for a session.

Depending on the requirement, the parameter OPEN\_CURSORS may need to be reset to a higher value. Set the SESSION\_CACHED\_CURSORS to a higher value to allow more cursors to be cached at session level and to avoid re-parsing.

#### Identify how many cursors are being opened by sessions

```
SQL> SELECT a.username, a.sid, b.value
 FROM v$session a, v$sesstat b, v$statname c
 WHERE b.sid = a.sid AND c.statistic# = b.statistic#
 AND c.name = 'opened cursors current' ORDER BY 3 DESC;
```

The VALUE column will identify how many cursors are open for a session and how near the count is to the OPEN\_CURSORS parameter value. If the margin is very small, consider increasing the OPEN\_CURSORS parameter.

#### Evaluate cached cursors for sessions as compared to parsing

```
SQL> SELECT a.sid, a.value parse_cnt,
 (SELECT x.value FROM v$sesstat x, v$statname y
 WHERE x.sid = a.sid AND y.statistic# = x.statistic#
 AND y.name = 'session cursor cache hits') cache_cnt
 FROM v$sesstat a, v$statname b
 WHERE b.statistic#=a.statistic# AND b.name='parse count(total)'
 AND value > 0;
The CACHE_CNT ('session cursor cache hits') of a session should be
compared to the PARSE_CNT ('parse count (total)'), if the difference is
high, consider increasing the SESSION_CACHED_CURSORS parameter.
The following parse related information is available in V$SYSSTAT and
V$SESSTAT views, connect with V$STATNAME using STATISTIC# column.
```

```
SQL> SELECT * FROM v$statname WHERE name LIKE '%parse%';
```

| STATISTIC# | NAME                   | CLASS |
|------------|------------------------|-------|
| 217        | parse time cpu         | 64    |
| 218        | parse time elapsed     | 64    |
| 219        | parse count (total)    | 64    |
| 220        | parse count (hard)     | 64    |
| 221        | parse count (failures) | 64    |

- Shared SQL area may be further utilized for not only identical but also for some-what similar queries by setting the initialization parameter CURSOR\_SHARING to FORCE. The default value is EXACT. Do not use this parameter in Oracle 8i, as there is a bug involved with it that hangs similar query sessions because of some internal processing. If you are on 9i, try out this parameter for your application in test mode before making changes in production.
- Prevent large SQL or PL/SQL areas from ageing out of the shared pool memory. Ageing out takes place based on Least recently used (LRU) mechanism. Set the parameter SHARED\_POOL\_RESERVED\_SIZE to a larger value to prevent large packages from being aged out because of new entries. A large overhead is involved in reloading a large package that was aged out.
- Pin frequent objects in memory using the DBMS\_SHARED\_POOL package. This package is created by default. It can also be created explicitly by running DBMSPOOL.SQL script; this internally calls PRVTPPOOL.PLB script. Use it to pin most frequently used objects that should be in memory while the instance is up, these would include procedure (p), functions (p), packages (p) and triggers (r). Pin objects when the instance starts to avoid memory fragmentation (Even frequently used data can be pinned but this is a separate topic).

#### To view a list of frequently used and re-loaded objects

```
SQL> SELECT loads, executions, SUBSTR(owner, 1, 15) "Owner",
 SUBSTR(namespace, 1, 20) "Type", SUBSTR(name, 1, 100) "Text"
 FROM v$db_object_cache ORDER BY executions DESC;
```

#### To pin a package in memory

```
SQL> EXEC dbms_shared_pool.keep('standard', 'p');
```

#### To view a list of pinned objects

```
SQL> SELECT SUBSTR(owner, 1, 15) "Owner",
 SUBSTR(namespace, 1, 20) "Type", SUBSTR(name, 1, 100) "Text"
 FROM v$db_object_cache WHERE kept = 'YES';
```

- Increasing the shared pool size is an immediate solution, but the above steps need to be carried out to optimize the database in the long run. The size of the shared pool can be increased by setting the parameter SHARED\_POOL\_SIZE in the initialization file.

## 49. Automatic Workload Repository (AWR)

### 49.1 Overview

Usually DBAs collect elaborate statistics during database operation and derive performance metrics from some third party tools. In a crisis, they access those metrics for comparisons to the present. Replaying these past events can shed light on current problems, so continuously capturing relevant statistics becomes important for performance analysis.

For some time, Oracle's solution in this area has been its built-in tool, Statspack. While it can prove invaluable in certain cases, it often lacks the robustness required by performance troubleshooting exercises. Oracle Database 10g offers a significant improvement: the Automatic Workload Repository (AWR). The AWR installs along with the database and captures not only statistics, but the derived metrics as well.

AWR capability is best explained quickly by the report it produces from collected statistics and metrics, by running the script `awrrpt.sql` in the `$ORACLE_HOME/rdbms/admin` directory. This script, in its look and feel, resembles Statspack; it shows all the AWR snapshots available and asks for two specific ones as interval boundaries.

It produces two types of output: text format, similar to that of the Statspack report but from the AWR repository, and the default HTML format, complete with hyperlinks to sections and subsections, providing quite a user-friendly report. Run the script and take a look at the report now to get an idea about capabilities of the AWR.

### 49.2 Implementation

AWR is an Oracle built-in tool that collects performance related statistics and derives performance metrics from them to track a potential problem. Unlike Statspack, snapshots are collected automatically every hour by a new background process called MMON and its slave processes. To save space, the collected data is automatically purged after 7 days.

The user can modify both the snapshot frequency and retention time. AWR uses several tables to store the collected statistics, all stored under the SYS schema in the new special tablespace named SYSAUX, and named in the format `WRM$_*` and `WRH$_*`. The former type stores metadata information such as the database being examined and the snapshots taken, and the latter type holds the actual collected statistics.

(H stands for "historical" and M stands for "metadata.") There are several views with the prefix `DBA_HIST_` built upon these tables, which can be used to write your own performance diagnosis tool. The names of the views directly relate to the table; for example, the view `DBA_HIST_SYSMETRIC_SUMMARY` is built upon the table `WRH$_SYSMETRIC_SUMMARY`.

The AWR history tables capture a lot more information than Statspack, including tablespace usage, filesystem usage, even operating system statistics. A complete list of these tables can be seen from the data dictionary through:

```
SQL> select view_name from user_views where view_name like 'DBA\HIST_%' escape '\';
```

The view `DBA_HIST_METRIC_NAME` defines the important metrics the AWR collects, the groups to which they belong, and the unit in which they are collected.

### 49.3 Using the Statistics

Most performance problems do not exist in isolation, but rather leave tell-tale signs that will lead to the eventual root cause of the problem. With Oracle Database 10g, changing storage parameters for a table to make it less dense or move it over to a tablespace with automatic segment space management.

Your plan of attack is generally methodical and usually based your knowledge of various events

and your experience in dealing with them. Now imagine if the same thing were done by an engine - an engine that captures metrics and deduces possible plans based on pre-determined logic. From Oracle Database 10g, this engine is known as Automatic Database Diagnostic Monitor (ADDM).

To arrive at a decision, ADDM uses the data collected by AWR. The ADDM sees the buffer busy waits are occurring and pulls the appropriate data to see the segments on which it occurs, evaluate its nature and composition, and finally offer solutions to the DBA.

After each snapshot collection by AWR, the ADDM is invoked to examine the metrics and generate recommendations. So, in effect you have a full-time robotic DBA analyzing the data and generating recommendations proactively, freeing you to attend to more strategic issues.

To see the ADDM recommendations and the AWR repository data, use the new Enterprise Manager 11g console on the page named DB Home. To see the AWR reports, you can navigate to them from Administration, then Workload Repository, and then Snapshots. We'll examine ADDM in greater detail in a future installment.

You can also specify alerts to be generated based on certain conditions. These alerts, known as Server Generated Alerts, are pushed to an Advanced Queue, from where they can be consumed by any client listening to it. One such client is Enterprise Manager 11g, where the alerts are displayed prominently.

## 49.4 Time Model

Oracle Database 11g introduces time models for identifying the time spent in various places. The overall system time spent is recorded in the view `V$SYS_TIME_MODEL`.

In addition to the database time, the `V$SYS_TIME_MODEL` view shows a whole lot of other statistics, such as time spent in different types of parsing and even PL/SQL compilation.

This view shows the overall system times as well; however, you may be interested in a more granular view: the session level times. The timing stats are captured at the session level as well, as shown in the view `V$SESS_TIME_MODEL`, where all the stats of the current connected sessions, both active and inactive, are visible. The additional column SID specifies the SID of the sessions for which the stats are shown.

In previous releases, this type of analysis was impossible to get and the user was forced to guess or derive from a variety of sources. In Oracle Database 11g, getting this information is a snap.

## 49.5 Active Session History (ASH)

The view `V$SESSION` in Oracle Database 10g has been improved; the most valuable improvement of them all is the inclusion of wait events and their duration, eliminating the need to see the view `V$SESSION_WAIT`. However, since this view merely reflects the values in real time, some of the important information is lost when it is viewed later.

For instance, if you select from this view to check if any session is waiting for any non-idle event, and if so, the event in question, you may not find anything because the wait must have been over by the time you select it.

Enter the new feature Active Session History (ASH), which, like AWR, stores the session performance statistics in a buffer for analysis later. However, unlike AWR, the storage is not persistent in a table but in memory, and is shown in the view `V$ACTIVE_SESSION_HISTORY`.

The data is polled every second and only the active sessions are polled. As time progresses, the old entries are removed to accommodate new ones in a circular buffer and shown in view.

Collects ASH statistics for all active sessions, for each second and stores them in SGA [5% shared pool.] Same statistics can be found in `V$ACTIVE_SESSION_HISTORY`.

- MMNL, this process captures ASH data to disk [AWR]. When memory is full.
- ASH data is sample of the most recent session data for all active sessions.

ASH also records parallel query server sessions, useful to diagnose the parallel query wait events. If the record is for a parallel query slave process, the SID of the coordinator server session is identified by `oc_SESSION_ID` column. The column `SQL_ID` records the ID of the SQL statement that produced the wait event, which can be joined with the `v$SQL` view to get the offending SQL statement. To facilitate the identification of the clients in a shared user environment like a web application, the `CLIENT_ID` column is also shown, which can be set by `DBMS_SESSION.SET_IDENTIFIER`.

The information is flushed to the disk by the MMON slave to the AWR table, visible through the view `DBA_HIST_ACTIVE_SESS_HISTORY`.

## 49.6 Manual Collection

Snapshots are collected automatically by default, but you can also collect them on demand. All AWR functionality has been implemented in the package `DBMS_WORKLOAD_REPOSITORY`. To take a snapshot, simply issue:

```
SQL> EXECUTE dbms_workload_repository.create_snapshot
```

It immediately takes a snapshot, recorded in the table `WRM$_SNAPSHOT`. The metrics collected are for the `TYPICAL` level. If you want to collect more detailed statistics, you can set the parameter `FLUSH_LEVEL` to `ALL` in the above procedure.

The stats are deleted automatically but can also be deleted manually by calling the procedure `drop_snapshot_range()`.

A typical performance tuning exercise starts with a capturing a baseline set of metrics, making changes, and then taking another baseline set. These two sets can be compared to examine the effect of the changes made. In AWR, the same kind of analogy can be implemented for existing snapshots taken.

## 50. Automatic Database Diagnostic Monitor (ADDM)

### 50.1 Overview

The Automatic Database Diagnostic Monitor (ADDM) analyzes data in the Automatic Workload Repository (AWR) to identify potential performance bottlenecks. For each of the identified issues it locates the root cause and provides recommendations for correcting the problem.

An ADDM analysis task is performed and its findings and recommendations stored in the database every time an AWR snapshot is taken provided the STATISTICS\_LEVEL parameter is set to TYPICAL or ALL. The ADDM analysis includes:

- CPU load
- Memory usage
- I/O usage
- Resource intensive SQL
- Resource intensive PL/SQL and Java
- RAC issues
- Application issues
- Database configuration issues
- Concurrency issues
- Object contention

The recommendations may include:

- Hardware changes
- Database configuration changes
- Schema changes
- Application changes
- Using other advisors

The analysis of I/O performance is affected by the DBIO\_EXPECTED parameter, which should be set to the average time (in microseconds) it takes to read a single database block from disk. Typical values range from 5000 to 20000 microseconds. The parameter can be set using:

```
EXECUTE DBMS_ADVISOR.set_default_task_parameter('ADDM', 'DBIO_EXPECTED', 8000);
```

### 50.2 Enterprise Manager

The obvious place to start viewing ADDM reports is Enterprise Manager. The "Performance Analysis" section on the "Home" page is a list of the top five findings from the last ADDM analysis task. Specific reports can be produced by clicking on the "Advisor Central" link, then the "ADDM" link. The resulting page allows you to select a start and end snapshot, create an ADDM task and display the resulting report by clicking on a few links.

#### addmrpt.sql Script

The addmrpt.sql script can be used to create an ADDM report from SQL\*Plus. The script is called as follows:

```
-- UNIX
@/u01/app/oracle/product/8.1.0/db_1/rdbms/admin/addmrpt.sql
-- Windows
@d:\oracle\product\8.1.0\db_1\rdbms\admin\addmrpt.sql
```

It then lists all available snapshots and prompts you to enter the start and end snapshot along with the report name.

## 50.3 DBMS\_ADVISOR

The DBMS\_ADVISOR package can be used to create and execute any advisor tasks, including ADDM tasks. The following example shows how it is used to create, execute and display a typical ADDM report:

```

BEGIN
 -- Create an ADDM task.
 DBMS_ADVISOR.create_task (
 advisor_name => 'ADDM',
 task_name => '970_1032_AWR_SNAPSHOT',
 task_desc => 'Advisor for snapshots 970 to 1031.');

 -- Set the start and end snapshots.
 DBMS_ADVISOR.set_task_parameter (
 task_name => '970_1032_AWR_SNAPSHOT',
 parameter => 'START_SNAPSHOT',
 value => 970);

 DBMS_ADVISOR.set_task_parameter (
 task_name => '970_1032_AWR_SNAPSHOT',
 parameter => 'END_SNAPSHOT',
 value => 1032);

 -- Execute the task.
 DBMS_ADVISOR.execute_task(task_name => '970_1032_AWR_SNAPSHOT');
END;
/
-- Display the report.
SQL> SET LONG 100000
SQL> SET PAGESIZE 50000
SQL> SELECT DBMS_ADVISOR.get_task_report('970_1032_AWR_SNAPSHOT') AS
report
 FROM dual;
SQL> SET PAGESIZE 24

```

The value for the SET LONG command should be adjusted to allow the whole report to be displayed.

The relevant AWR snapshots can be identified using the DBA\_HIST\_SNAPSHOT view.

## 50.4 Related Views

The following views can be used to display the ADDM output without using Enterprise Manager or the GET\_TASK\_REPORT function:

- DBA\_ADVISOR\_TASKS - Basic information about existing tasks.
- DBA\_ADVISOR\_LOG - Status information about existing tasks.
- DBA\_ADVISOR\_FINDINGS - Findings identified for an existing task.
- DBA\_ADVISOR\_RECOMMENDATIONS - Recommendations for the problems identified by an existing task.

## 51. Automatic Shared Memory Management (ASMM)

Automatic Shared Memory Management makes it possible to allocate memory where it's needed most, automatically. Sometimes the memory allocated to shared pool is insufficient for answering the user request.

In some cases the cause may not be the size of the pool itself, but rather fragmentation that results from excessive parsing due to non-usage of bind variables. The other errors derive from inadequate space in the large pool and Java pool respectively.

### 51.1 How Do You Split the Pie?

The System Global Area (SGA) of an Oracle instance comprises several memory areas, including the buffer cache, shared pool, Java pool and large pool. These pools occupy fixed amounts of memory in the operating system's memory space; their sizes are specified by the DBA in the initialization parameter file.

The four pools db block buffer cache, shared pool, Java pool, and large pool occupy almost all the space inside the SGA. (Relative to the other areas, the redo log buffer does not occupy much space and is inconsequential) A DBA must ensure that their respective memory allocations are sufficient.

The Automatic Shared Memory Management feature in Oracle Database 10g decides the total size of the SGA and then set a parameter named SGA\_TARGET that decides the total size of the SGA. The individual pools within the SGA will be dynamically configured based on the workload. A non-zero value of the parameter SGA\_TARGET is all that is needed to enable the automatic memory allocation.

### 51.2 Which Pools is not affected?

Some pools in SGA are not subject to dynamic resizing, and must be specified explicitly. Notable among them are the buffer pools for nonstandard block sizes and the non-default ones for KEEP or RECYCLE. If your database has a block size of 8K, and you want to configure 2K, 4K, 16K, and 32K block-size pools, you must set them manually. Their sizes will remain constant; they will not shrink or expand based on load.

You should consider this factor when using multiple size buffer, KEEP, and RECYCLE pools. In addition, log buffer is not subject to the memory adjustment the value set in the parameter log\_buffer is constant, regardless of the workload. (In 11g, a new type of pool can also be defined in the SGA: Streams pool, set with parameter streams\_pool\_size. This pool is also not subject to automatic memory tuning).

If you specify any of these non-auto-tunable parameters (such as db\_2k\_cache\_size), their total size is subtracted from the SGA\_TARGET value to calculate the automatically tuned parameter values so that the total size of the SGA remains constant. For instance, imagine that the values look like this:

```
sga_target = 500M
db_2k_cache_size = 50M
```

And the rest of the pool parameters are unset. The 2KB buffer pool of 50MB leaves 450MB for the auto-tuned pools such as the default block size buffer pool (db\_cache\_size), shared pool, Java pool, and large pool. When the non-tunable parameter such as the 2KB block size pool is dynamically adjusted in such a way that the tunable portion's size is affected, the tunable portion is readjusted.

The memory requirements of various pools in Oracle SGA are not static rather; they vary based on the demand on the system. Automatic Shared Memory Management in Oracle Database 10g allows DBAs to manage system memory more efficiently by dynamically reallocating resources to where they are needed most while enforcing a specified maximum to prevent paging and swapping. More efficient memory management also leads to fewer memory requirements, which can make leaner hardware more viable.

### 51.3 MMAN:

This background process responsible for ASMM to adjust components as per the request.

1. **DB buffer cache**
2. **Shared pool**
3. **Large pool**
4. **Java pool**

Other than the above-mentioned components, we still have streams pool & redo log buffer. Oracle recommends let ASMM manage the manageable components, which means we shouldn't mention the above components in init.ora file.

## 52. Row Migration & Row Chaining

### 52.1 Overview

In Oracle we sometimes come across poor performance in our databases. Though there may be several reasons for low performance, we can prevent some of them by properly designing and/or diagnosing the database.

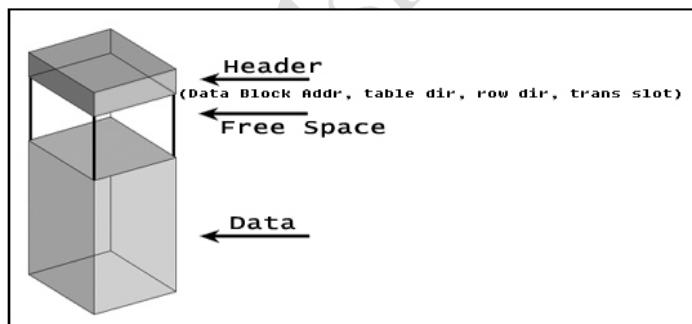
Row Migration (RM) & Row Chaining (RC) are two potential problems that can be prevented. By suitably diagnosing RM/RC, we can improve database performance. The main considerations are-

- What is RM/RC?
- How to identify RM/RC?
- How to avoid RM/RC?

To begin let's start with the basic idea of an Oracle block.

### 52.2 Oracle Block

The Operating System Block size is the minimum unit of operation (read/write) by the OS and is a property of the OS file system. While creating an Oracle DB we have to choose the 'Data Base Block Size' as a multiple of the Operating System Block size. The minimum unit of operation (read/write) by the Oracle database would be this 'Oracle block', and not the OS block. Once set, the 'Data Base Block Size' cannot be changed during the life of the database (except in case of Oracle 9i). To decide on a suitable block size for the database, we take into consideration factors like the size of the database and the concurrent number of transactions expected. Proper block sizing and its use are very important from the tuning point of view. The DB Block (dbblock) has the following structure.



3. Header: Header contains the general information about the data i.e. block address, and type of segments (table, index etc). It also contains the information about table and the actual row (address), which holds the data.
4. Free Space: Space allocated for future update/insert operations. Generally affected by the values of PCTFREE and PCTUSED parameters.
5. Data: Actual row data.

While creating/altering any table/index, Oracle used two storage parameters for space control.

- PCTFREE: The percentage of space reserved for future update of existing data.
- PCTUSED: The percentage of minimum space used for insertion of new row data. This value determines when the block gets back into Free List table.

An example would make the terms more clear. Suppose we create a table RMRC as

```
CREATE TABLE RMRC (col1 number, col2 varchar2(100))
STORAGE (PCTFREE 20 PCTUSED 55);
```

This statement will create one table RMRC having columns - col1 and col2, using PCTFREE 20 and PCTUSED 52. Now we want to insert the first record into the table

```
SQL> INSERT INTO RMRC VALUES(1, 'Hello');
```

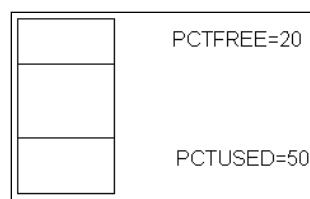
Oracle will first search for a free block in the 'free list' (this is the table where oracle maintains a list of all free available blocks) and then the data is inserted into that block

**Note:** The availability of block in the 'free list' is decided by the PCTFREE value. Initially an empty block will be listed in the free list table, and it will continue to remain there until the free space reaches the PCTFREE value.

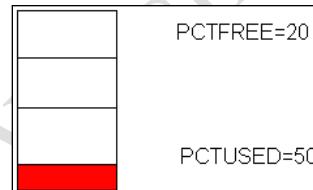
When the free space reaches the PCTFREE value the block is removed from the free list, and it is re-listed in the free list table when the volume of data in the block comes below the PCTUSED value.

Oracle uses free list to increase the performance. So for every insert operation, oracle needs to search for the free blocks only from the 'free list' table instead of searching all blocks.

Let's consider the first DB block

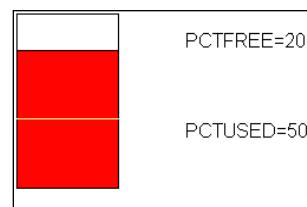


Now the first record we inserted occupies 10 units (let's consider the block size as 100 unit neglecting the header size for the sake of simplicity).



10 unit of the block is occupied by the first row we inserted. Since this is less than the available free space (80), the block would still be available for the next insertion.

We now insert 7 more rows (10 units each), which will utilize 70 more units of the block.



After 7 more rows insertion (total occupied space: 80 units) you would notice that it has occupied the available free space (80). So now this block will be removed from the free list. And if we want to insert a new row, Oracle will use the next block that is available in free list table.

Suppose we want to update our first record, which will result in the increase in the row size by another 15 units. It will take the 15 unit space required from the 20 units of PCTFREE. If we want to update the second row requiring additional 15 units of space, we would be unable to find the space in this block (we are left with just 5 more units in our block).

This is the time where Row Migration comes into the picture.

## 52.3 Row Migration (RM):

Oracle will try to shift the entire row from current block to another block that has 25 unit (10+15) free space. However, it will not remove all the relevant entries for that row from the old block. It will store the new block row ID into the old block.

Now, if we want to view that record Oracle will internally first check the old block and then from there it will get the new row id and display the row data from the new block. With this extra amount of I/O operation required you might have guessed correctly that it would degrade the performance.

Now the first question comes into mind: What is the use of maintaining the old row id if whole row data is migrating from old block to new block? This is because of oracle's internal mechanism. For the entire life span of a row data its row id will never change. That's why oracle has to maintain two-row ids. One is because of internal mechanism and one is for the current location of the data.

## 52.4 Row Chaining (RC):

What we have discussed so far is the case when we have data in a block and new insertion is not possible into that block, so Oracle will uses a new block. But what happens when a row is so large that it cannot fit into one free block?

Oracle will span this data into a number of blocks so that it can hold that data. Existence of such types of data will cause what is called 'Row Chaining'. Row Chaining is storage of data in a chain of blocks. This will mainly occur in the lob, clob, blob or big varchar2 data types.

Row chaining occurs when a row can't physically fit into an Oracle block. Another block is required to store the remainder of the row. Chaining can cause serious performance problems and is especially prevalent with those storing multimedia data or large binary objects (blobs). You should pay special attention to the DB\_BLOCK\_SIZE parameter when you create your database. Block sizes of 4 kilobytes or more are the norm, not the exception.

Migration of an Oracle row occurs when a row is updated in an Oracle block and the amount of free space in the block is not adequate to store all of the row's data. The row is migrated to another physical block in the table. The problem is that the indexes that refer to the migrated row are still pointing to the block where the row used to be, and hence the table reads are doubled. Note however that full table scans will scan blocks as they come and will perform the same number of reads whether the rows are migrated or not.

If a table has chaining problems, you can rebuild the table, specifying a larger value for the PCTFREE parameter. If the bulk of the rows currently in the table have already been updated to their full lengths, a lot of space will be wasted. The free space will be reserved for rows that will not expand any further. To eliminate this waste, you can create the table with a smaller PCTFREE parameter, load the existing data, and then run the ALTER command on the table with a larger PCTFREE.

## 52.5 How to Find RM/RC

Oracle has provided the following three methods to create/view the statistic of tables/indexes:

1. ANALYZE command
2. Dynamic views
3. Report.txt method

Execute this command to get the statistics of tab1 table-

```
SQL> ANALYZE TABLE RMRC LIST CHAINED ROWS;
```

It will populate the CHAINED\_ROWS table. (The CHAINED\_ROWS table should have been created first by executing \$ORACLE\_HOME/rdbms/utlchain.sql script.)

Query this table for head\_rowid column to get the row id of the migrated/chained row. By executing utbstat and utleststat (scripts provide by oracle and found in \$ORACLE\_HOME/rdbms) for a period of time we can create a report.txt file. Check the statistics of “table fetch continued row” in report.txt. Check the value of “table fetch continued row” in V\$SYSSTAT view. These steps can uncover the existence of Row Chaining / Migration, we now need to find a cure for it.

## 52.6 How to avoid/eliminate RM/RC

For avoiding row migration we can use a higher PCTFREE value. Since migration is typically caused by update operation. However, there is a trade off here. The space, which is allocated to PCTFREE, is not used for normal insert operation and can be wasted.

A temporary solution (it will only take care of existing RM and not future RM) is to delete the migrated row from the table and insert again. To do this, follow these steps:

1. Analyze the table.... to get the row id,
2. Copy those rows to temporary table,
3. Delete the rows from the original table,
4. Insert the rows from step 2 back to original table.

Avoiding row chaining is very difficult since generally it is caused by insert operation using large data types i.e. lob etc. A good precaution is to either use large block size (which can't be changed without creating a new database) or a large extent size.

## 53. Networking Tuning

### 53.1 Overview

Oracle Shared Server is the new name for Oracle Multi-threaded Server (MTS) and includes several new features. In MTS, clients contacts listener asking for a connection. The listener sent the address of a dispatcher to the client, who then contacted to the dispatcher directly to confirm the connection.

In Oracle Shared Server the client contacts the listener to ask for a connection resulting in a connection socket being established between the listener and the client. The listener then hands this socket to the relevant dispatcher to allow instant communication between the client and the dispatcher. This process is called a Direct Handoff.

Prior to Oracle9i, dispatchers handled network and database events separately. Network events were handled using the UNIX system call poll() via the network services, while database event were handled by the Virtual Operating System (VOS), an internal abstraction layer not visible to the DBAs. Polling and coordinating both event models wasted time. In Oracle9i the Common Event Model has poll() incorporated into the VOS layer on database so there is no need to poll the network services anymore.

Performance Manager, part of the Diagnostics Pack in Oracle Enterprise Manager (OEM), provides graphical monitoring the shared server. This can be done with an overall view, or detailed to the level of dispatchers, shared server processes and listeners. In addition Performance Manager can view the SQL statements fired for each session and provide help in tuning the Shared Server parameters.

With Oracle's importance on supporting larger user communities, the multithreaded server (MTS) option has become an important part of the Oracle Net architecture. Normally, by default Oracle works with a dedicated server processes per user.

Whenever a user establishes a valid connection he will be hooked with a server processes which is dedicated only for that user requests. In most cases the end users will be hooked up with the database but not utilizing the resources (DSP) given to him.

That is why we choose working with MTS to utilize minimum server resources. In a shared server environment a user will be connected to a dispatcher and a dispatcher can support multiple user connections.

This dispatcher will help him in database requests and replies. This dispatcher will maintain two queue's i.e., request and response queue. Shared server architecture requires Oracle Net Services.

Parameter Descriptions to be included in init.ora for MTS configuration:

```
SHARED_SERVERS=2
MAX_SHARED_SERVERS=6
DISPATCHERS="(PROTOCOL=TCP) (DISP=2)"
MAX_DISPATCHERS=4
LOCAL_LISTENER="(ADDRESS= (PROTOCOL=TCP) (HOST=192.168.0.15) (PORT=1521))"
```

#### Steps:

1. Start the listener
2. Configure the tnsnames.ora on client side giving the valid service\_name in connect\_data strings. (i.e., SQL> show parameter service\_name on server)
3. If we want a dedicated connection add one more string in connect\_data (SERVER=DEDICATED)

Format of tnsnames.ora file:

```

TNS_DEMO =
 (DESCRIPTION =
 (ADDRESS_LIST =
 (ADDRESS = (PROTOCOL = TCP) (HOST = 192.168.0.15) (PORT = 1521))
)
 (CONNECT_DATA =
 (SERVICE_NAME = DEMO.ORADBA15)
)
)
Format for tnsnames.ora file for Dedicated Connection:
TNS_DEMO =
 (DESCRIPTION =
 (ADDRESS_LIST =
 (ADDRESS = (PROTOCOL = TCP) (HOST = 192.168.0.15) (PORT =
 1521))
)
 (CONNECT_DATA =
 (SERVER = DEDICATED)
 (SERVICE_NAME = DEMO.ORADBA15)
)
)
)

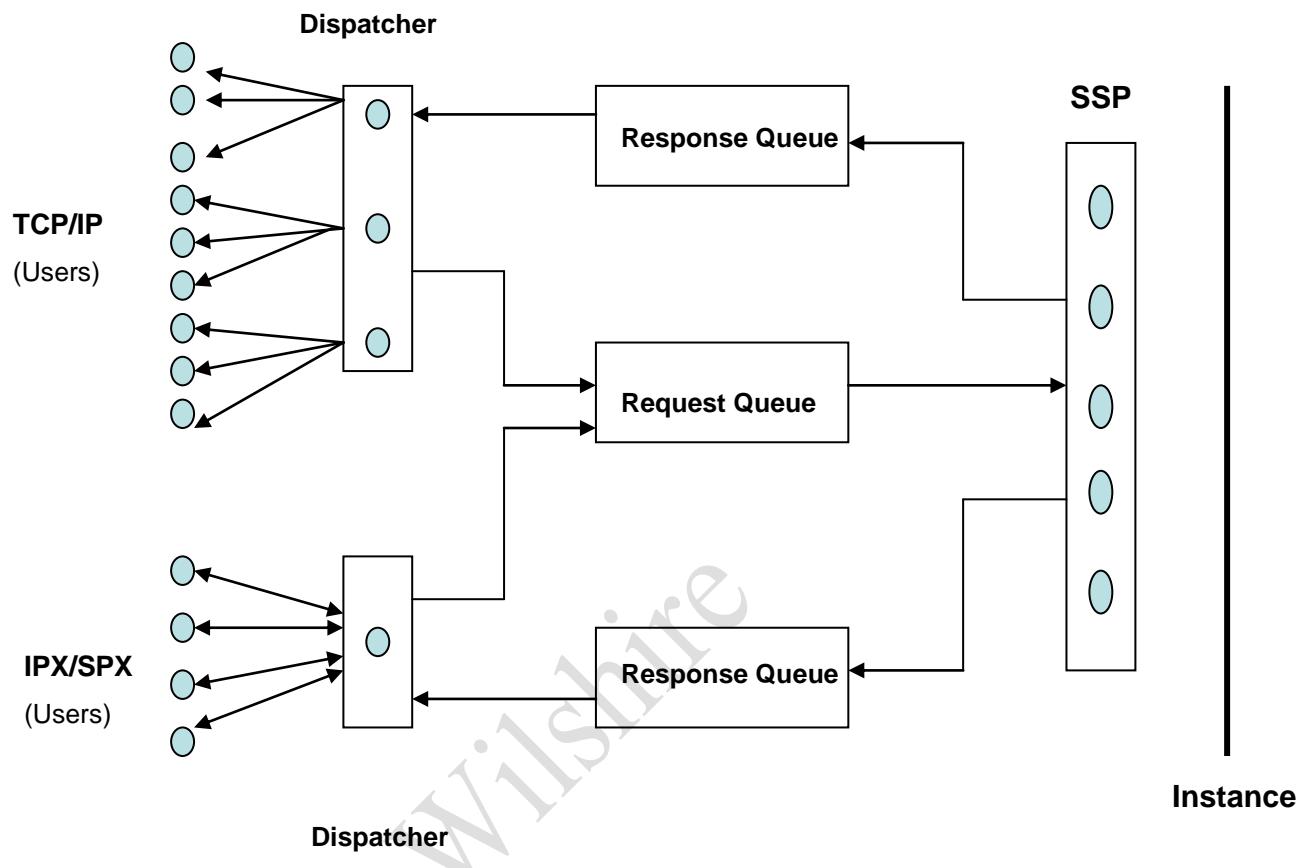
```

## 53.2 Monitoring MTS Connections

| View                     | Description                                                                                                                            |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| V\$DISPATCHER            | Provides information on the dispatcher processes, including name, network address, status, various usage statistics, and index number. |
| V\$DISPATCHER_CONFIG     | Provides configuration information about the dispatchers.                                                                              |
| V\$DISPATCHER_RATE       | Provides rate statistics for the dispatcher processes.                                                                                 |
| V\$QUEUE                 | Contains information on the shared server message queues.                                                                              |
| V\$SHARED_SERVER         | Contains information on the shared servers.                                                                                            |
| V\$CIRCUIT               | Contains information about virtual circuits, which are user connections to the database through dispatchers and servers.               |
| V\$SHARED_SERVER_MONITOR | Contains information for tuning shared server.                                                                                         |
| V\$SGA                   | Contains size information about various system global area (SGA) groups. May be useful when tuning shared server.                      |
| V\$SGASTAT               | Contains detailed statistical information about the SGA, useful for tuning.                                                            |
| V\$SHARED_POOL_RESERVED  | Lists statistics to help tune the reserved pool and space within the shared pool.                                                      |

# Multithreaded Server

## Routing Mechanism



## 54. Partitioned Tables and Indexes

Databases have grown to hundreds of gigabytes

Tables greater than 10GB are common in large data warehouse systems.

Tables need an associated index or indexes, which can also be gigabytes in size.

Tables can grow larger than a medium-sized database.

### 54.1 Why Use Partitioning?

- Continued data availability with partial failures
- Data management operations within finite maintenance windows
- Scalable performance with substantial growth in data volumes
- Simplified data disk placement

### 54.2 Partitioned Tables and Indexes

- Tables and indexes can be decomposed into smaller pieces called partitions
- Partitioning is particularly useful in VLDB environments where tables and indexes can grow to gigabytes or terabytes in size and are therefore vulnerable to failure and are not easy to manage.
- Oracle 8i supports up to 64,000 partitions per single table or index.

### 54.3 Key Features

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Core Functionality</b></p> <ul style="list-style-type: none"> <li>▪ Tables can be partitioned using five possible techniques:           <ul style="list-style-type: none"> <li>○ Range Partitioning</li> <li>○ List Partitioning</li> <li>○ Hash Partitioning</li> <li>○ Composite Range-Hash Partitioning</li> <li>○ Composite Range-List Partitioning</li> </ul> </li> <li>▪ Index-organized tables can be partitioned using two possible techniques:           <ul style="list-style-type: none"> <li>○ Range Partitioning</li> <li>○ Hash Partitioning</li> </ul> </li> <li>▪ Indexing possibilities:           <ul style="list-style-type: none"> <li>○ Local Index: Index partitioned same as base table</li> <li>○ Global Non-Partitioned Index: Index is not partitioned</li> <li>○ Global Partitioned Index: Index partitioned using different criteria from base table</li> </ul> </li> </ul> | <p><b>Key Performance Features</b></p> <ul style="list-style-type: none"> <li>▪ Partition Pruning</li> <li>▪ Partition-wise Joins</li> <li>▪ Parallel Update and Delete</li> </ul> <p><b>Manageability Features</b></p> <ul style="list-style-type: none"> <li>▪ Complete, parallelized DDL Support for adding, dropping, splitting, and merging partitions</li> <li>▪ Automatic row migration during updates</li> <li>▪ Automatic maintenance of global indexes during DDL</li> </ul> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 54.4 Tables versus Index partitioning

- A partitioned table can have partitioned and non-partitioned indexes.
- A non-partitioned table can have partitioned and non-partitioned indexes.
- Bitmap indexes on non-partitioned tables cannot be partitioned.

## 54.5 Basic Partitioning Methods

- Range Partitioning: is defined by
- The partitioning specification for a table or index
- Partitioning specifications for each partition
- Hash Partitioning
- List Partitioning (New Feature of Oracle9i). Use List partitioning when you require explicit control over how rows map to partitions. You can specify a list of discrete values for the partitioning column in the description for each partition. This is different from Range partitioning, where a range of values is associated with a partition, and from Hash partitioning, where the user has no control of the row to partition mapping. List partitioning method is specifically designed for modeling data distributions that follow discrete values. This cannot be easily done by range or hash partitioning. List partitioning allows unordered and unrelated sets of data to be grouped and organized together.

### A partitioned table does not support:

- LONG and LONG RAW datatypes
- Being a part of a cluster table

However partitioning support is provided for all LOB types.

## 54.6 Updatable Partition Keys

- A partitioned table can be created or altered to allow row movement between partitions.  
SQL> ALTER TABLE SALES ENABLE ROW MOVEMENT;
- Can only be applied to partitioned tables.
- Disabled by default.

## 54.7 Equipartitioned Tables and Indexes

- Tables and Indexes are equipartitioned when
- Tables and Indexes have the same partitioning method (range or hash).
- Tables and Indexes must have same number of partitions.
- Range partitions must have same VALUES LESS THAN boundaries.
- Tables and Indexes must have the same number, type, and scale in partition key columns.
- Consider equipartition for table/index and table/table primary key-foreign key (PK-FK) whenever possible.

## 54.8 Partitioned Indexes

- Indexes can be either partitioned or non-partitioned. Choice of indexing strategy allows greater flexibility to suit database and application requirements.
- An index can be partitioned, except you cannot partition a cluster index.

## 54.9 Types of Indexes

- Non-partitioned indexes
- Partitioned indexes
- Global indexes
- Global prefixed indexes
- Local indexes
- Local prefixed indexes
- Local non-prefixed indexes

A global index can only be range partitioned.

A global index can be created by specifying the keyword “GLOBAL”.

In a global index, one index partition may refer to more than one underlying table partition.

An index is prefixed if the leftmost columns of the index are based on exactly the same columns as the partition key of the index.

An index is non-prefixed if the leftmost columns of the index are not based on the same columns as the partition key of the index.

## Global Prefixed Indexes

- Global prefixed indexes are normally not equipartitioned with the table.
- Oracle maintains only the index structures not the partitions themselves.
- Highest partition must be able to hold maximum value for the date; therefore MAXVALUE must be specified.
- Global prefixed indexes can be unique or non-unique.
- There is no index as global non-prefixed. A non-partitioned index would perform better than global non-prefixed index.

## 54.10 Prefixed Versus Non-prefixed Local Indexes

Use local prefixed indexes whenever possible

4. Both (prefixed and non-prefixed local indexes) support partition independence.
5. May be more expensive to scan a non-prefixed index because more index probes may be used

### Data Dictionary Views for Partitioned Tables and Indexes

- |                                                                                                                                                                                |                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>▪ DBA_PART_TABLES</li> <li>▪ DBA_TAB_PARTITIONS</li> <li>▪ DBA_PART_KEY_COLUMNS</li> <li>▪ DBA_TABLES</li> <li>▪ DBA_OBJECTS</li> </ul> | <ul style="list-style-type: none"> <li>▪ DBA_PART_INDEXES</li> <li>▪ DBA_IND_PARTITIONS</li> <li>▪ DBA_INDEXES</li> <li>▪ DBA_OBJECTS</li> <li>▪ DBA_SEGMENTS</li> </ul> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## 54.11 Benefits of Range Partitioning

### High Availability

Partitions can be independently managed

Backup and restore operations can be done on individual partitions

Partitions that may be unavailable

### Ease of Administration

A partition can be moved from one tablespace to another

A partition can be divided at a user-defined value

A partition can be dropped, added, or truncated

Select, Update, Insert, and Delete can be applied on partition level instead of table level

### Improved Performance

The optimizer eliminates partitions that do not need to be scanned

Partitions can be scanned in parallel. Partitions can be updated, inserted deleted in parallel

The size of temporary tables used for sorts can be reduced

### Example to Create a List Partitioned Table:

```
SQL> CREATE TABLE SALES_HISTORY (PNO NUMBER PRIMARY KEY,
 PNAME VARCHAR2 (50) NOT NULL, QTY NUMBER DEFAULT 0,
 SALES NUMBER DEFAULT 0, STATE VARCHAR2 (20) NOT NULL,
 COUNTRY VARCHAR2 (20))
PARTITION BY LIST (COUNTRY) (
 PARTITION EUROPE VALUES ('UK', 'Germany', 'France'),
 PARTITION North_America VALUES ('US', 'Canada', 'Mexico'),
 PARTITION South_America values ('Brazil', 'Argentina'),
 PARTITION ASIA VALUES ('japan', 'singapore', 'malyasia');
```

## 55. Partitioning Tables Enhancements in Oracle 11g

Partitioning addresses key issues in supporting very large tables and indexes by letting you decompose them into smaller and more manageable pieces called partitions. SQL queries and DML statements do not need to be modified in order to access partitioned tables.

However, after partitions are defined, DDL statements can access and manipulate individual's partitions rather than entire tables or indexes. This is how partitioning can simplify the manageability of large database objects. Also, partitioning is entirely transparent to applications.

Each partition of a table or index must have the same logical attributes, such as column names, datatypes, and constraints, but each partition can have separate physical attributes such as pctfree, pctused, and tablespaces.

Partitioning is useful for many different types of applications, particularly applications that manage large volumes of data. OLTP systems often benefit from improvements in manageability and availability, while data warehousing systems benefit from performance and manageability. Partitioning offers these advantages:

- Partitioning enables data management operations such data loads, index creation and rebuilding, and backup/recovery at the partition level, rather than on the entire table.
- This results in significantly reduce times for these operations.
- Partitioning improves query performance. In many cases, the results of a query can be achieved by accessing a subset of partitions, rather than the entire table. For some queries, this technique (called partition pruning) can provide order-of-magnitude gains in performance.
- Partitioning can significantly reduce the impact of scheduled downtime for maintenance operations.
- Partition independence for partition maintenance operations lets you perform concurrent maintenance operations on different partitions of the same table or index. You can also run concurrent SELECT and DML operations against partitions that are unaffected by maintenance operations.
- Partitioning increases the availability of mission-critical databases if critical tables and indexes are divided into partitions to reduce the maintenance windows, recovery times, and impact of failures.
- Partitioning can be implemented without requiring any modifications to your applications. For example, you could convert a non-partitioned table to a partitioned table without needing to modify any of the SELECT statements or DML statements which access that table. You do not need to rewrite your application code to take advantage of partitioning.

This section includes the following topics:

- Partition Key
- Partitioned Tables
- Partitioned Index-Organized Tables
- Partitioning Methods

### 55.1 Partition Key

Each row in a partitioned table is unambiguously assigned to a single partition. The partition key is a set of one or more columns that determines the partition for each row. Oracle Database automatically directs insert, update, and delete operations to the appropriate partition through the use of the partition key. A partition key:

- Consists of an ordered list of 1 to 16 columns
- Cannot contain a LEVEL, ROWID, or MLSLABEL pseudo column or a column of type ROWID
- Can contain columns that are NULL able

### 55.2 Partitioned Tables

Tables can be partitioned into up to 1024K-1 separate partitions. Any table can be partitioned except those tables containing columns with LONG or LONG RAW datatypes. You can, however, use tables containing columns with CLOB or BLOB datatypes.

**Note:** To reduce disk use and memory use (specifically, the buffer cache), you can store tables and partitioned tables in a compressed format inside the database. This often leads to a better

scale up for read-only operations. Table compression can also speed up query execution. There is, however, a slight cost in CPU overhead.

### 55.3 Partitioned Index-Organized Tables

You can partition index-organized tables by range, list, or hash. Partitioned index-organized tables are very useful for providing improved manageability, availability, and performance for index-organized tables. In addition, data cartridges that use index-organized tables can take advantage of the ability to partition their stored data. For partitioning an index-organized table:

- Partition columns must be a subset of primary key columns
- Secondary indexes can be partitioned—locally and globally
- OVERFLOW data segments are always equipartitioned with the table partitions

### 55.4 Partitioning Methods

Oracle provides the following partitioning methods:

1. Range partitioning maps data to partitions based on ranges of partition key values that you establish for each partition.
2. Interval partitioning is an extension of range partitioning which instructs the database to automatically create partitions of a specified interval when data inserted into the table exceeds all of the range partitions.
3. Hash partitioning maps data to partitions based on a hashing algorithm that evenly distributes rows among partitions, giving partitions approximately the same size.
4. List partitioning enables you to explicitly control how rows map to partitions by specifying a list of discrete values in the description for each partition.
5. Reference partitioning enables you to partition a table based on the partitioning scheme of the table referenced in its referential constraint.
6. Composite partitioning is a combination of two partitioning methods to further divide the data into subpartitions:
7. Composite range-range partitioning partitions data using the range method, and within each partition, subpartitions it using the range method.
8. Composite range-hash partitioning partitions data using the range method, and within each partition, subpartitions it using the hash method.
9. Composite range-list partitioning partitions data using the range method, and within each partition, subpartitions it using the list method.
10. Till oracle11g we have only range-list, range-hash partitions, but from oracle 11g major enhancements in composite partitioning occurred which lead to convenience of partitioning from rang-{any partition type} and list-{any partition type}.
11. Composite list-range partitioning partitions data using the list method, and within each partition, subpartitions it using the range method.
12. Composite list-hash partitioning partitions data using the list method, and within each partition, subpartitions it using the hash method.
13. Composite list-list partitioning partitions data using the list method, and within each partition, subpartitions it using the list method.
14. System partitioning enables application-controlled partitioning for arbitrary tables.

### 55.5 Interval Partitioning

This new partitioning strategy fully automates the partition creation for range. In other words, new partitions will be created when they are needed. By defining the interval criteria, the database knows when to create new partitions for new or modified data.

Managing the creation of new partitions can be a cumbersome and highly repetitive task. This is especially true for predictable additions of partitions covering small ranges, such as adding new daily partitions. Interval partitioning automates this operation by creating partitions on-demand.

## 55.6 Reference Partitioning

The partitioning key is resolved through an existing parent-child relationship, enforced by active primary key or foreign key constraints. The benefit of this feature is that tables with a parent-child relationship can be logically equi-partitioned by inheriting the partition key from the parent table without duplicating the key columns.

The logical dependency will also automatically cascade partition maintenance operations, thus making application development easier and less error-prone.

## 55.7 System Partitioning

System partitioning enables application-controlled partitioning for arbitrary tables and indexes. The database simply provides the ability to break down an object into partitions without any data placement rules.

All aspects of data placement and retrieval are controlled by the application. System partitioning provides the well-known benefits of partitioning (scalability, availability, and manageability), but the partitioning and actual data placement are controlled by the application

## 55.8 Virtual Column-Based Partitioning

In Oracle Database 11g, you can now partition key columns defined on virtual columns of a table. Frequently, business requirements to logically partition objects do not match existing columns in a one-to-one manner. Oracle partitioning has been enhanced to allow a partitioning strategy being defined on virtual columns, thus enabling a more comprehensive match of the business requirements.

## 55.9 Partitioning Tables Enhancements in oracle 11g R2

### 55.15.1 LOCAL PARTITION

Use this clause to indicate that the indextype can be used to create local domain indexes on range- and list-partitioned tables. You use this clause in combination with the storage\_table in several ways like storage\_table\_clause

The recommended method is to specify WITH LOCAL [RANGE] PARTITION WITH SYSTEM MANAGED STORAGE TABLES. This combination uses system-managed storage tables, which are the preferred storage management, and lets you create local domain indexes on both range- and list-partitioned tables.

In this case the RANGE keyword is optional and ignored, because it is no longer needed if you specify WITH LOCAL PARTITION WITH SYSTEM MANAGED STORAGE TABLES.

- You can specify WITH LOCAL RANGE PARTITION (including the RANGE keyword) and omit the storage\_table clause. Local domain indexes on range-partitioned tables are supported with user-managed storage tables for backward compatibility. Oracle does not recommend this combination because it uses the less efficient user-managed storage tables.

If you omit this clause entirely, then you cannot subsequently use this indextype to create a local domain index on a range- or list-partitioned table.

### 55.15.1.1 Storage Table Clause (storage\_table\_clause)

Use this clause to specify how storage tables and partition maintenance operations for indexes built on this indextype are managed:

- Specify WITH SYSTEM MANAGED STORAGE TABLES to indicate that the storage of statistics data is to be managed by the system. The type you specify in statistics\_level should be storing the statistics related information in tables that are maintained by the system. Also, the indextype you specify must already have been created or altered to support the WITH SYSTEM MANAGED STORAGE TABLES clause.
- Specify WITH USER MANAGED STORAGE TABLES to indicate that the tables that store the user-defined statistics will be managed by the user. This is the default behavior.

### 55.11.2 Allow Virtual Columns in the Primary Key or Foreign Key for Reference Partitioning

Virtual columns can be used as the primary or the foreign key column of a reference partition table. Allowing the use of virtual columns for reference partitioned tables enables an easier implementation of various business scenarios using Oracle Partitioning.

### 55.15.3 System-Managed Indexes for List Partitioning

System-managed domain indexes are now supported for list partitioned tables.

This feature provides enhanced completeness of domain-specific indexing support for partitioning to meet user requirements including Oracle XML DB. Performance of local domain indexes on list partitioned tables is improved in this release

## 56. Partitioning Enhancements in Oracle 12c

Oracle Database 12c offers several enhancements for partitioning. In a previous blog posting, I talked about reference partitioning enhancements. This post will present other enhancements in relation to general partitioning.

### 56.1 Asynchronous index maintenance

With Oracle database, a global index becomes UNUSABLE when dropping or truncating the partition on which this index points, until the clause UPDATE GLOBAL INDEXES is used.

With Oracle 11g, dropping or truncating a partition involves the index maintenance, consisting on the deletion of orphaned entries. With Oracle 12c, the index maintenance can be deferred, automatically or manually.

To demonstrate this new feature, create a table TB\_SALARIES on an Oracle 11g R2 and an Oracle 12c databases in a schema MSC, containing a list of salaries with their name, first name and hire date. The table uses range partitioning by year, based on the hire date.

### 56.2 Multiple partition maintenance

Oracle 12c allows managing multiple partitions at the same time. This is not the case with Oracle 11g, where partitions must be added, deleted or truncated one by one.

### 56.3 ONLINE move partition – Enhancement in 12c

Oracle 12c offers the new clause **ONLINE** for move operation on partitions and sub partitions. This clause allows DML during the move operation, and global indexes remain in a valid state after a move partition operation. Not only DML are immediately executed, but also the partition move operation is completely transparent for users.

Note that if a transaction is active on the concerned partition before the move partition statement is ordered, two scenarios may occur:

**Without ONLINE clause**, the error "ORA-00056.: resource busy and acquire with NOWAIT specified or timeout expired" is displayed. The transaction must be finished (commit or rollback) and the move statement must be re executed. This is common to Oracle 11g and Oracle 12c releases.

**With ONLINE clause**, the move operation waits the transaction to be finished (commit or rollback) and is then executed.

In Oracle 11g, it was already possible to perform DML operation when moving a partition. But the DML stayed pending until the move was finished, causing a relative waiting time depending on the move operation duration.

Move a partition also made indexes UNUSABLE in Oracle 11g, if the moved partition was not empty. So, even if a DML was finally executed after moving the partition, the error "ORA-01502 index '%s.%s' or partition of such index is in unusable state" was displayed in many cases and the statement was rolled back.

## 57. Indexes

Indexes are totally optional structures that are intended to speed up the execution of SQL statements against table data and cluster data. Indexes are used for direct access to a particular row or set of rows in a table. Indexes are most typically organized as some type of tree structure.

### 57.1 Concepts and Facts

An index can be composed of a single column for a table, or it may be comprised of more than one column for a table. An index based on more than one column is termed a concatenated (or composite) index.

Examples:

1. The SSN key is used to track individual students at a university.
2. The concatenated primary key index for an ENROLL table (SSN + SectionID + Term + Year) that is used to track the enrollment of a student in a particular course section.
3. The maximum number of columns for a concatenated index is 32; but the combined size of the columns cannot exceed about one-half of a data block size.

A unique index allows no two rows to have the same index entry. An example would be an index on student SSN.

A non-unique index allows more than one row to have the same index entry (this is also called a secondary key index). An example would be an index on U.S. Mail zip codes.

A function-based index is created when using functions or expressions that involve one or more columns in the table that is being indexed. A function-based index pre-computes the value of the function or expression and stores it in the index. Function-based indexes can be created as either a B-tree or a bitmap index.

A partitioned index allows an index to be spread across several tablespaces - the index would have more than one segment and typically access a table that is also partitioned to improve scalability of a system. This type of index decreases contention for index lookup and increases manageability.

- To create an index in our own schema:
  - The table to be indexed must be in our schema, OR
  - We have the INDEX privilege on the table to be indexed, OR
  - We have the CREATE ANY INDEX system privilege.
- To create an index in another schema:
  - We have the CREATE ANY INDEX system privilege, AND
  - The owner of the other schema has a quota for the tablespace that will store the index segment (or UNLIMITED TABLESPACE privilege).

### 57.2 Loading Data

Data initially loaded into a table will load more efficiently if the index is created after the table is created. This is because the index must be updated after each row insertion if the index is created before loading data.

Creating an index on an existing table requires sort space – typically memory values that are paged in and out of segments in the TEMP tablespace allocated to a user. Users are also allocated memory for index creation based on the SORT\_AREA\_SIZE parameter – if memory is insufficient, then swapping takes place.

### 57.3 B-Tree Index Structure

The below figure illustrates in a very simple way the structure of a B-Tree index. A B-tree index usually stores a list of primary key values and a list of associated ROWID values that point to the

row location of the record with a given primary key value. In this figure the ROWID values are represented by the pointers.

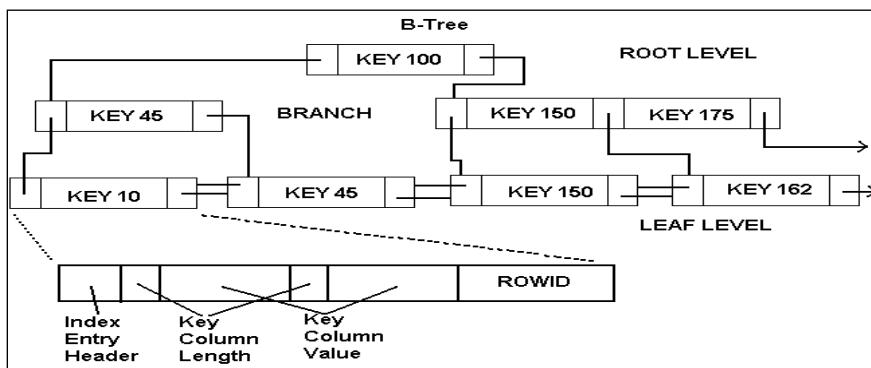


Figure 1-1. B-Tree Index

The top level of the index is called the Root. The Root points to entries at lower levels of the index - these lower levels are termed Branch Blocks. A node in the index may store multiple (more than one) key values - in fact, almost all B-Trees have nodes with multiple values - the tree structure grows upward and nodes split when they reach a specified size. At the Leaf level the index entries point to rows in the table.

At the Leaf level the index entries are linked in a bi-directional chain that allows scanning rows in both ascending and descending order of key values - this supports sequential processing as well as direct access processing. In a non-partitioned table, Key values are repeated if multiple rows have the same key value – this would be a non-unique index (unless the index is compressed). Index entries are not made for rows that have all of the key columns NULL.

**Leaf Index Format:** The index entry at the Leaf level is made up of three components.

**Entry Header** - stores number of columns in the index and locking information about the row to which the index points.

**Key Column Length-Value Pairs** - defines the size of the column in the key followed by the actual column value. These pairs are repeated for each column in a composite index.

**ROWID** - This is the ROWID of a row in a table that contains the key value associated with this index entry.

**Data Manipulation Language Effects:** Any DML on a table also causes the Oracle Server to maintain the associated indexes.

When a row is inserted into a table, the index must also be updated. This requires the physical insertion of an index entry into the index tree structure.

When a row is deleted from a table, the index only has the entry "logically" deleted (turn a delete bit from off to on). The space for the deleted row is not available for new entries until all rows in the block are deleted.

When a row key column is updated for a table, index has both logical deletion and physical insertion into the index.

**PCTFREE** has no effect on an index except when the index is created. New entries to an index may be added to an index block even if the free space in the block is less than the PCTFREE setting.

- If an indexed table has lots of rows to be inserted, set PCTFREE high to accommodate new index values.
- If the table is static, set PCTFREE low.
- PCTUSED cannot be specified for indexes.

### 57.3.1 Creating B-Tree Indexes

An example CREATE INDEX command for a normal B-tree index is given here.

```
SQL> CREATE [UNIQUE] INDEX User345.Orders_Index ON
User345.Orders(OrderId)
PCTFREE 20 INITTRANS 6 MAXTRANS 10
STORAGE (INITIAL 48K NEXT 48K PCTINCREASE 0 MAXEXTENTS 100)
LOGGING TABLESPACE Index01;
```

The UNIQUE clause specifies unique entries - the default is NONUNIQUE. Note that the owner's schema (User350) is specified – this is optional. The PCTFREE parameter is only effective when the index is created - after that, new index block entries are made and PCTFREE is ignored. PCTFREE is ignored because entries are not updated - instead a logical delete and physical insert of a new index entry is made. PCTUSED cannot be specified for an index because updates are not made to index entries.

Use a low PCTFREE when the indexed column is system generated as would be the case with a sequence (sequence indexes tend to increase in an ascending fashion) because new entries tend to be made to new data blocks - there are no or few insertions into data blocks that already contain index entries. Use a high PCTFREE when the indexed column or set of columns can take on random values that are not predictable.

Such is the case when a new Orderline row is inserted - the ProductID column may be a non-unique foreign key index and the product to be sold on an Orderline is not predictable for any given order.

The Default and Minimum for INITTRANS is 2. The limit on MAXTRANS is 255 - this number would be inordinately large. By default, LOGGING is on so that the index creation is logged into the redo log file.

Specifying NOLOGGING would increase the speed of index creation initially, but would not enable recovery at the time the index is created. Interestingly, Oracle will use existing indexes to create new indexes whenever the key for the new index corresponds to the leading part of the key of an existing index.

### 57.3.2 Indexes and Constraints

The UNIQUE and PRIMARY KEY constraints on tables are enforced by creating indexes on the unique key or primary key – creation of the index is automatic when such a constraint is enabled. We can specify a USING INDEX clause to control the creation process. The index created takes the name of the constraint unless otherwise specified.

Example: Creating a PRIMARY KEY constraint while creating a table.

```
SQL> CREATE TABLE District (
District_Number NUMBER(5) CONSTRAINT District_PK PRIMARY KEY,
District_Name VARCHAR2(50))
ENABLE PRIMARY KEY USING INDEX TABLESPACE Index01 PCTFREE 0;
```

### 57.4 Key-Compressed Index

This is a B-tree with compression – compression eliminates duplicate occurrences of a key in an index leaf block.

```
SQL> CREATE INDEX Emp_Name ON Emp (Last_Name, First_Name)
TABLESPACE Index01 COMPRESS 1;
```

This approach breaks an index key into a prefix and suffix entry in the index block. Compression causes sharing of the prefix entries among all suffix entries and save lots of space allowing the storage of more keys in a block.

Use key compression when:

The index is non-unique where the ROWID column is appended to the key to make the index key unique.

The index is a unique multicolumn index → example: Zip\_Code + Last\_Name.

## 57.5 Other General Topics

This section covers altering, rebuilding, coalescing, and dropping indexes. It also covers validating indexes and identifying unused indexes. Guidelines are provided for when to index columns.

### 57.5.1 Altering Index Storage Parameters

The ALTER INDEX command is used to alter storage parameters. Use of the command is straight-forward as illustrated in this example.

```
SQL> ALTER INDEX User345.Products_Region_Idx STORAGE(NEXT 200K MAXEXTENTS 200) ;
```

One of the most common changes is to increase MAXEXTENTS for an index as a system grows in size.

**Allocating and De-allocating Index Space:** If a DBA is going to insert a lot of rows into a table, performance can be improved by adding extents to the indexes associated with the table prior to doing the inserts.

This improves performance by avoiding the dynamic addition of index extents. This can be accomplished by using the ALTER INDEX command with the ALLOCATE EXTENT option.

```
SQL> ALTER INDEX User345.Products_Region_Idx
 ALLOCATE EXTENT (SIZE 400K
 DATAFILE '/a01/student/user350/oradata/user350index01.dbf') ;
```

We can DEALLOCATE unused index space after the insertions are complete. This will free up space that is not in use within the tablespace.

```
SQL> ALTER INDEX User345.Products_Region_Idx DEALLOCATE UNUSED ;
```

### 57.5.2 Creating an Index Online

Creating an Online Index allows Data Manipulation Language (DML) operations against the table while the index build is being completed. DDL operations on the table are not allowed during the index creation process.

```
SQL> CREATE INDEX Manager_Employee_Idx ON Employee(Manager_ID, Emp_SSN)
 ONLINE;
```

There are some restrictions:

- Temporary tables cannot be created/rebuilt online.
- Partitioned indexes must be created/rebuilt one partition at a time.
- We cannot de-allocate unused space during an online rebuild.
- We cannot change the PCTFREE parameter for the whole index.

### 57.5.3 Rebuilding Indexes

We may improve system performance by rebuilding an index that is highly fragmented due to lots of physical insertions and logical deletions. Such may be the case for a SALES\_ORDER table where filled orders are deleted over time.

```
SQL> ALTER INDEX User345.Products_Region_Idx REBUILD TABLESPACE Index01;
```

If we rebuild an index from an existing index, the rebuild operation is more efficient because the index information does not need to be sorted - it is already sorted. We can specify a new tablespace if necessary for the rebuilt index. The older index is deleted after the new index is rebuilt. We must have sufficient space in the tablespace for the old/new index during the rebuild operation.

Queries use the old index while the rebuild operation is under way.

We can also specify the ONLINE option when rebuilding an index.

```
SQL> ALTER INDEX User345.Products_Region_Idx REBUILD ONLINE;
```

#### 57.5.4 Coalescing Indexes

This is an alternative to the ALTER INDEX...REBUILD command – it eliminates index fragmentation. Coalesce on an index is a block rebuild that is done online. In situations where a B-tree index has leaf blocks that can be freed up for reuse, we can merge leaf blocks using the ALTER INDEX...COALESCE statement.

```
SQL> ALTER INDEX User345.Products_Region_Idx COALESCE;
```

In this figure, the first two leaf node blocks are about 50% full – fragmentation is evident. After coalescing, blocks are merged freeing up a block for reuse.

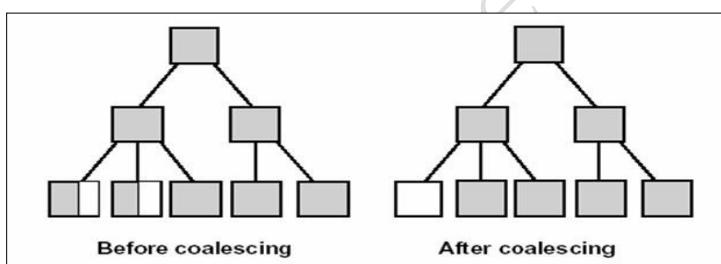


Figure 3-1. Coalescing Index

##### 57.5.4.1 When to Rebuild or Coalesce

| Rebuild Index                                       | Coalesce Index                               |
|-----------------------------------------------------|----------------------------------------------|
| To quickly move an index to another tablespace.     | Cannot move the index to another tablespace. |
| Requires more disk space.                           | Operation does not require more disk space.  |
| Creates new index tree – height of tree may shrink. | Only coalesces leaf blocks.                  |
| Enables changing storage and tablespace parameters. | Frees up index leaf blocks for use.          |

#### 57.5.5 Checking Index Validity

The ANALYZE INDEX command with the VALIDATE STRUCTURE option can be used to check for block corruption in an index.

```
SQL> ANALYZE INDEX dbock.pk_course VALIDATE STRUCTURE;
```

We can query the INDEX\_STATS view and obtain information about the index.

SQL> ANALYZE INDEX Course\_PK VALIDATE STRUCTURE;

Index analyzed.

```
SQL> SELECT blocks, pct_used, distinct_keys, lf_rows, del_lf_rows
 FROM index_stats;
```

| BLOCKS | PCT_USED | DISTINCT_KEYS | LF_ROWS | DEL_LF_ROWS |
|--------|----------|---------------|---------|-------------|
| 15     | 1        | 2             | 2       | 0           |

An index needs to be reorganized when the proportion of deleted rows (DEL\_LF\_ROWS) to existing rows (LF\_ROWS – short for Leaf) is greater than 30%.

### 57.5.6 Dropping an Index

DBAs drop indexes if they are no longer needed by an application, i.e., the application is discontinued. We also drop indexes prior to large data loads - this improves performance and tends to use index space more efficiently. We can also drop indexes that are marked by the system as INVALID because of some type of instance failure, or if the index is corrupt.

```
SQL> DROP INDEX index_name;
```

We cannot drop an index used to implement an integrity constraint. We would first need to remove the integrity constraint by altering the table, and then drop the index.

### 57.5.7 Identifying Unused Indexes

Beginning with Oracle9i, statistics about the usage of an index can be gathered and displayed in `V$OBJECT_USAGE`.

Start usage monitoring of an index:

```
SQL> ALTER INDEX user345.dept_id_idx MONITORING USAGE;
```

Stop usage monitoring of an index:

```
SQL> ALTER INDEX user345.dept_id_idx NOMONITORING USAGE;
```

If the information gathered indicates that an index is never used, the index can be dropped.

In addition, eliminating unused indexes cuts down on overhead that the Oracle server has to do for DML, thus performance is improved.

Each time the MONITORING USAGE clause is specified, `V$OBJECT_USAGE` will be reset for the specified index. The previous information is cleared or reset, and a new start time is recorded.

### 57.5.8 Guidelines for Creating Indexes

A table can have any number of indexes, but the more indexes, the more overhead that is associated with DML operations:

- A single row insertion or deletion requires updating all indexes on the table.
- An update operation may or may not affect indexes depending on the column updated.

The use of indexes involves a tradeoff - query performance tends to speed up, but data manipulation language operations tend to slow down.

- Query performance improves because the index speeds up row retrieval.
- DML operations slow down because along with row insertions, deletions, and updates, the indexes associated with a table must also have insertions and deletions completed.
- For very volatile tables, minimize the number of indexes used.

When possible, store indexes to a tablespace that does not have rollback segments, temporary segments, and user/data tables – DBAs usually create a separate tablespace just used for index segments.

We can minimize fragmentation by using extent sizes that are at least multiples of 5 times the DB\_BLOCK\_SIZE for the database. Create an index when row retrieval involves less than 15% of a large table's rows – retrieval of more rows is generally more efficient with a full table scan. We may want to use NOLOGGING when creating large indexes - we can improve performance by avoiding redo generation.

Indexes created with NOLOGGING requires a backup as their creation is not archived.

Example:

```
SQL> CREATE BITMAP INDEX User345.Products_Region_Idx
 ON User345.Products_Region (RegionId ASC) NOLOGGING;
```

Usually index entries are smaller than the rows they index. Data blocks that store index entries tend to store more entries in each block, so the INITTRANS parameter should be higher on indexes than on their related tables.

Columns that are strong candidates for indexing include:

Take into consideration the typical queries that will access the table. A concatenated index is only used by Oracle in retrieving data when the leading column of the index is used in a query's WHERE clause.

- The order of the columns in the CREATE INDEX statement affect query performance – specify the most frequently used columns first.
- An index on column1, column2, and column3 can improve performance for queries with WHERE clauses on column1 or on column1+column2, but will not be used for queries with a WHERE clause that just uses column2 or column3.

Columns used in join operations – join operations perform better if the columns used in the join are indexed.

Columns where the values stored in the columns are relatively unique (but not number columns that store currency values).

Columns with a wide range of values (use regular B-tree indexes).

Columns with a small range of values (use bitmap indexes).

Columns with few NULL values and queries often search for rows having some value.

Small tables don't need indexes – but it is a good idea to still index the Primary Key.

## 57.7 Special Indexes

### 57.7.1 Bitmap Index Structure

Bitmap indexes are alternatives to B-tree indexes and are only used to create secondary key indexes. They are used in certain situations:

If a table has millions of rows and there are very few distinct values for index entries, for example, indexes on zip codes could have many, many rows for a single zip code value, and then a bitmap index may perform well.

- A bitmap index may best support WHERE conditions involving the OR operator.
  - A bitmap index may work best for low update activity tables or read-only key columns.

A bitmap index is also organized as a B-tree structure; however, the leaf nodes in a bitmap index stores a bitmap for each key value - it does NOT store the ROWIDs. Instead, each bit corresponds to a possible ROWID. If the bit is set on, the row with the corresponding ROWID contains the key value. This figure illustrates this concept. In the figure, the key consists of only one column, and the first entry has a logical key value of Blue, the second is Green, the third Red, and the fourth Yellow.

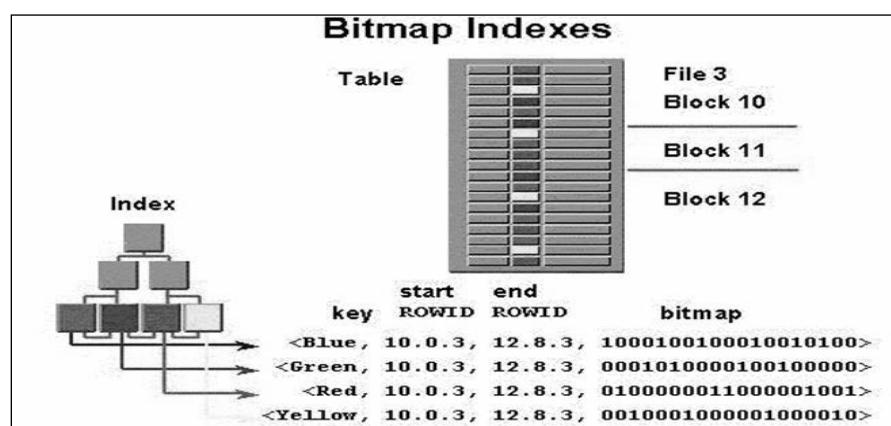


Figure 2-1. Bitmap Index

**Comparisons and Limitations:** Updates to key columns for bitmap indexes require locking a significant portion of the index in order to perform the update. This table compares B-tree and bitmap indexes.

|                                                   |                                                 |
|---------------------------------------------------|-------------------------------------------------|
| B-Tree                                            | Bitmap                                          |
| Suitable for high-cardinality columns             | Suitable for low-cardinality columns            |
| Updates on keys relatively inexpensive            | Updates to key columns very expensive           |
| Inefficient for queries using OR logical operator | Efficient for queries using OR logical operator |
| Used mostly for On-Line Transaction Processing    | Very useful for Data Warehousing                |

### 57.7.1.1 Creating Bitmap Indexes

Example CREATE BITMAP INDEX command:

```
SQL> CREATE BITMAP INDEX User345.Products_Region_Idx
 ON User345.Products_Region (RegionId ASC)
 PCTFREE 40 INITTRANS 6 MAXTRANS 10
 STORAGE (INITIAL 48K NEXT 48K PCTINCREASE 0 MAXEXTENTS 100)
 LOGGING TABLESPACE Index01;
```

A BITMAP index cannot be specified as UNIQUE since they are only used for secondary key indexes. We can specify an INIT.ORA parameter CREATE\_BITMAP\_AREA\_SIZE to specify the amount of memory allocated for storing bitmap segments. The default is 8MB. If the memory allocated for bitmap index segment creation is larger, then indexes are created faster. If there are only a few unique values for the index key field, then you can allocate much less memory to the creation of bitmap index segments, perhaps only a few kilobytes.

## 57.8 Reverse Key Index:

- It "reverses" the bytes of each column indexed (except the ROWID), while keeping the column order.
- By reversing the keys of index, the insertions become distributed across all leaf keys in the index.
- Should be used in a situation where user inserts ascending values and deletes lower values from a Table, thereby preventing "skewed" indexes.

### 57.8.1 Limitations of Reverse-Key Indexes:

- A reverse-key index cannot be used if the query requires a range scan.
- A bitmapped index cannot be reversed.
- An index-organized table cannot be reversed.

```
SQL> CREATE INDEX <indexname> ON <TABLE (column, COLUMN) > REVERSE;
```

## 57.9 Function-Based Indexes

These indexes support queries that use a value returned by a function or expression in a WHERE clause.

- The index computes and stores the value from the function or expression in the index.
- The COMPATIBLE parameter in the init.ora file must be 8.1.0.0.0 or higher.
- If user-based functions are used, the function must be marked DETERMINISTIC (stochastic functions won't work).
- If the function is owned by another user, you must have the EXECUTE privilege on the function object.
- The table must be analyzed after the index is created.
- The query must be guaranteed to not need NULL values from the indexed expression because NULL values are not stored in indexes.

Example: This defines a function-based index (Last\_Name\_Caps\_Idx) based on the UPPER function – this facilitates use of the UPPER function when specifying a WHERE clause condition and ensures use of the INDEX to return values efficiently.

```
SQL> CREATE TABLE Employee (
 Emp_ID NUMBER(5) CONSTRAINT Employee_PK PRIMARY KEY,
 Last_Name VARCHAR2(20), First_Name VARCHAR2(20)) TABLESPACE
Data01;
 INSERT INTO Employee VALUES (1, 'Bock', 'Douglas');
 INSERT INTO Employee VALUES (2, 'Bordoloi', 'Bijoy');
 INSERT INTO Employee VALUES (3, 'Sumner', 'Mary');
 INSERT INTO Employee Values (4, 'BOCK', 'Ronald');
Now create the (non-unique) index, then select from the table.
SQL> CREATE INDEX Last_Name_Caps_Idx ON Employee (UPPER(Last_Name))
 TABLESPACE Index01;
```

```
SQL> SELECT Emp_ID, Last_Name, First_Name FROM Employee
 WHERE UPPER(Last_Name) LIKE 'BOCK';

 EMP_ID LAST_NAME FIRST_NAME
 ----- -----
 1 Bock Douglas
 4 BOCK Ronald
```

## 57.10 Enhancements In Oracle 12c

You can create multiple indexes on the same set of columns as long as some characteristic is different like

**Unique versus nonunique**

**B-tree versus bitmap**

**Different partitioning strategies**

**Indexes that are not partitioned and indexes that are partitioned**

**Indexes that are locally partitioned and indexes that are globally partitioned**

**Indexes that differ in partitioning type (range or hash)**

It's very helpful feature if you want to quickly migrate to different type of index or use them during different period of time.

NOTE – to create such indexes you need to use INVISIBLE clause.

```
SQL>> create table test_tbl (id1 number, id2 number, id3 number, id4 number);
```

Not using INVISIBLE clause steal leads to standard oracle error like in previous version before 12C:

ORA-01408: such column list already indexed

```
SQL>> create unique index test_tbl_idx1 on test_tbl(id1);
SQL>> create index test_tbl_idx2 on test_tbl(id1);
```

**SQL Error: ORA-01408: such column list already indexed**

```
SQL>> drop index test_tbl_idx1;
```

Let's make indexes on the same set of columns with different characteristic

Unique versus non-unique

```
SQL>> create unique index test_tbl_idx1 on test_tbl(id1);
SQL>> create index test_tbl_idx2 on test_tbl(id1) invisible;
```

**B\*tree versus bitmap(MULTIPLE INDEXES ON SAME COLUMN ORACLE 12C)**

```
SQL>> create index test_tbl_idx3 on test_tbl(id2);
SQL>> create bitmap index test_tbl_idx4 on test_tbl(id2) invisible;
```

## 57.11 Descending Indexes

Oracle treats descending indexes as if they were function-based indexes. You do not need the QUERY REWRITE or GLOBAL QUERY REWRITE privileges to create them, as you do with other function-based indexes. However, as with other function-based indexes, Oracle does not use descending indexes until you first analyze the index and the table on which the index is defined. Oracle ignores DESC if index is bitmapped or if the COMPATIBLE initialization parameter is set to a value less than 8.1.0.

```
SQL> CREATE INDEX index_gender_state ON index_demo (person_id, state
DESC)
 PCTFREE 0 TABLESPACE indx_sml;
SQL> SELECT index_name, index_type FROM user_indexes;
```

## 57.12 Index Organized Tables (IOT):

Index-organized tables store data in B\*-tree structures that are similar to the indexes on regular tables. However, these tables minimize overall storage, because there is no need for a separate index structure that holds columns already defined in the table.

- IOT structure is transparent to users
- No syntax changes for DML on IOTs.
- Used for information retrieval applications that is content-based, such as text, images, and sound storage and retrieval.
- If index blocks contain long rows, full index scans may be slow.
- We can store non-key columns in an overflow area.

We can specify:

- The overflow tablespace
- A threshold size for overflow rows
- The column name where the row is split

### 57.11.1 Row Overflow Area

The row overflow area preserves dense clustering of a B\*-tree index.

If the row size becomes greater than PCTTHRESHOLD, the non-key column values are stored in the overflow tablespace

The index entry then contains only the key value and a pointer to the rest of the row.

```
SQL> CREATE TABLE test (token VARCHAR2 (20),
 doc_id number, token_frequency number,
 token_occurrence_data VARCHAR2 (1000), constraint pk_test
 primary key (token, doc_id)
 organization index tablespace text_indx pctthreshold 20
 overflow tablespace users_overflow;
```

### 57.11.2 Advantages:

- Faster key-based access to table data
- Reduced storage requirements

### 57.11.3 Restrictions:

- Must have a primary key
- Cannot use unique constraints
- Cannot contain LONG columns
- Distribution and replication not supported
- Cannot create an IOT of object types

## 58. Flashback Query

Oracle's **FlashBack Query** feature enables us to see a consistent version of DB as of a specified time in the past. We can execute queries, or even applications, as of a previous time in DB. The retention period for undo information is an important factor in execution of Flashback queries. Oracle uses DBMS\_FLASHBACK package to implement this. We must establish an undo retention interval that is long enough that it enables us to construct a snapshot of DB for oldest version of database we are interested in. This is done through UNDO\_RETENTION initialization parameter.

**Assumptions:** An employee table that has an on delete cascade constraint on manager column. One of the employees is deleted from the table and all the subordinates belonging to that employee are deleted, using flashback query, we try to retain the information back into the employee table. We maintain a table called keep\_scn to store the SCN number for flashback query.

### Steps to implement Flashback Query

Creating tables employee and keep\_scn and insert records into employee table.

```
SQL> create table keep_scn (scn number);
SQL> create table employee (employee_no NUMBER(5) primary key,
 employee_name VARCHAR2 (20), employee_MGR NUMBER (5),
 salary number, hiredate date);
```

Viewing the information of all the employees using connect by clause

```
SQL> SELECT LPAD (' ', 2*(level-1)) || employee_name Name
 from employee connect by prior employee_no = employee_mgr
 start with employee_no = 1 order by level;
Generating the SCN number using DBMS_FLASHBACK and storing in keep_scn table
SQL> DECLARE
 I NUMBER;
 begin
 I:= dbms_flashback.get_system_change_number;
 Insert into keep_scn values (I);
 commit;
```

Now delete SMITH from employee table

```
SQL> delete from employee where employee_name = 'Roger Smith';
sql> commit;
```

Restoring the table to point before deleting smith's info using DBMS\_FLASHBACK

```
SQL> declare
 restore_scn number;
 begin
 SELECT SCN INTO restore_scn from keep_scn;
 dbms_flashback.enable_at_system_change_number (restore_scn);
 end;
After performing step 5, we will be able to view the deleted records from the table, write an explicit cursor and get the data back into the table.
declare
 s_emp number;
 s_mgr number;
 cursor c2 is
 c2_rec c2%ROWTYPE;
 begin
 for c2_rec in c2 loop
 dbms_flashback.disable;
 begin
 dbms_output.put_line(c2_rec.eno);
 insert into employee values(c2_rec.eno,c2_rec.ename,c2_rec.mgr,
 c2_rec.sal,c2_rec.hiredate);end;end loop;end;
```

## 59. Flashback Version Query

### 59.1 Overview

In Oracle9i DB, we have seen the introduction of the "time machine" manifested in the form of Flashback Query. This feature allows DBA to see the value of a column as of a specific time, as long as the before image copy of the block is available in undo segment.

However, Flashback Query only provides a fixed snapshot of the data as of a time, not a running representation of changed data between two time points. Some applications, such as those involving the management of foreign currency, may need to see the value data changes in a period, not just at two points of time.

### 59.2 Querying Changes to a Table

In this example, We have used a bank's foreign currency management application. The database has a table called RATES to record exchange rate on specific times.

```
SQL> desc rates
```

| Name     | Null? | Type           |
|----------|-------|----------------|
| CURRENCY |       | VARCHAR2 (4)   |
| RATE     |       | NUMBER (15,10) |

This table shows the exchange rate of US\$ against various other currencies as shown in the CURRENCY column. In the financial services industry, exchange rates are not merely updated when changed, rather they are recorded in a history. This approach is required because bank transactions can occur as applicable to a "past time," to accommodate the loss in time because of remittances. For example, for a transaction that occurs at 10:12AM but is effective as of 9:12AM, the applicable rate is that at 9:12AM, not now.

Up until now, the only option was to create a rate history table to store the rate changes, and then query that table to see if a history is available. Another option was to record the start and end times of the applicability of the particular exchange rate in the RATES table itself. When the change occurred, the END\_TIME column in the existing row was updated to SYSDATE and a new row was inserted with the new rate with the END\_TIME as NULL.

In Oracle Database 11g, however, the Flashback Versions Query feature may obviate the need to maintain a history table or store start and end times. Rather, using this feature, you can get the value of a row as of a specific time in the past with no additional setup. Bear in mind, however, that it depends on the availability of the undo information in the database, so if the undo information has been aged out, this approach will fail.

For example, say that the DBA, in the course of normal business, updates the rate several times or even deletes a row and reinserts it:

```
SQL> INSERT INTO rates VALUES ('EURO',1.1012);
SQL> COMMIT;
SQL> UPDATE rates SET rate = 1.1014;
SQL> COMMIT;
SQL> UPDATE rates SET rate = 1.1013;
SQL> COMMIT;
SQL> DELETE rates;
SQL> COMMIT;
SQL> INSERT INTO rates VALUES ('EURO',1.1016);
SQL> COMMIT;
SQL> UPDATE rates SET rate = 1.1011;
SQL> COMMIT;
```

```
SQL> SELECT * FROM rates;

CURREN RATE
----- -----
EURO 1.1011
```

This output shows the current value of the RATE, not all the changes that have occurred since the first time the row was created. Thus using Flashback Query, you can find out the value at a given point in time; but we are more interested in building an audit trail of the changes somewhat like recording changes through a camcorder, not just as a series of snapshots taken at a certain point.

The following query shows the changes made to the table:

```
SQL> SELECT versions_starttime, versions_endtime, versions_xid,
 versions_operation, rate
 FROM rates versions BETWEEN timestamp minvalue AND maxvalue
 ORDER BY VERSIONS_STARTTIME

VERSIONS_STARTTIME VERSIONS_ENDTIME VERSIONS_XID V
RATE

--
01-DEC-03 03.57.12 PM 01-DEC-03 03.57.30 PM 0002002800000C61 I
1.1012
01-DEC-03 03.57.30 PM 01-DEC-03 03.57.39 PM 000A000A00000029 U
1.1014
01-DEC-03 03.57.39 PM 01-DEC-03 03.57.55 PM 000A000B00000029 U
1.1013
01-DEC-03 03.57.55 PM
1.1013
01-DEC-03 03.58.07 PM 01-DEC-03 03.58.17 PM 000A000D00000029 I
1.1016
01-DEC-03 03.58.17 PM
1.1011
```

Note that all the changes to the row are shown here, even when the row was deleted and reinserted. The VERSION\_OPERATION column shows what operation (Insert/Update/Delete) was performed on the row. This was done without any need of a history table or additional columns.

The above query, the columns versions\_starttime, versions\_endtime, versions\_xid, versions\_operation are pseudo-columns, similar to other familiar ones such as ROWNUM, LEVEL. Other pseudo-columns such as VERSIONS\_STARTSCN and VERSIONS\_ENDSCN show the System Change Numbers at that time. The column versions\_xid shows the identifier of the transaction that changed the row. More details about transaction can be found from the view FLASHBACK\_TRANSACTION\_QUERY, where the column XID shows transaction id.

For instance, using VERSIONS\_XID value 000A000D00000029 from above, the UNDO\_SQL value shows the actual statement.

```
SQL> SELECT UNDO_SQL FROM FLASHBACK_TRANSACTION_QUERY
 WHERE XID = '000A000D00000029';
```

```
UNDO_SQL
```

```

insert into "ANANDA"."RATES"("CURRENCY","RATE") values ('EURO','1.1013');
```

In addition to the actual statement, this view also shows the timestamp and SCN of commit and the SCN and timestamp at the start of the query, among other information.

## 59.3 Finding Out Changes During a Period

How we can find out the value of the RATE column at 3:57:57. PM.

```
SQL> SELECT rate, versions_starttime, versions_endtime FROM rates
 versions
 BETWEEN timestamp TO_DATE('12/1/2003 15:57:57.', 'mm/dd/yyyy
hh24:mi:ss')
 AND TO_DATE('12/1/2003 16:57:55', 'mm/dd/yyyy hh24:mi:ss')

 RATE VERSIONS_STARTTIME VERSIONS_ENDTIME
 ----- -----
 1.1011
```

This query is similar to the flashback queries. In the above example, the start and end times are null, indicating that the rate did not change during the time period; rather, it includes a time period. You could also use the SCN to find the value of a version in the past. The SCN numbers can be obtained from the pseudo-columns VERSIONS\_STARTSCN and VERSIONS\_ENDSCN. Here is an example:

```
SQL> SELECT rate, versions_starttime, versions_endtime
 FROM rates versions BETWEEN scn 1000 AND 1001
```

Using the keywords MINVALUE and MAXVALUE, all the changes that are available from the undo segments is displayed. You can even give a specific date or SCN value as one of the end points of the ranges and the other as the literal MAXVALUE or MINVALUE. For instance, here is a query that tells us the changes from 3:57:52 PM only; not the complete range:

```
SQL> SELECT versions_starttime, versions_endtime, versions_xid,
 versions_operation, rate
 FROM rates versions BETWEEN timestamp
 TO_DATE('12/11/2003 15:57:52', 'mm/dd/yyyy hh24:mi:ss') AND
 maxvalue
 ORDER BY VERSIONS_STARTTIME

 VERSIONS_STARTTIME VERSIONS_ENDTIME VERSIONS_XID V
 RATE

 --
 01-DEC-03 03.57.55 PM 000A000C00000029 D
 1.1013
 01-DEC-03 03.58.07 PM 01-DEC-03 03.58.17 000A000D00000029 I 1.1016
 01-DEC-03 03.58.17 PM 000A000E00000029 U
 1.1011
```

Flashback Versions Query replicates the short-term volatile value auditing of table changes out of the box. This advantage enables the DBA to get not a specific value in the past, but all the changes in between, going as far back as the available data in undo segments. Therefore, the maximum available versions are dependent on the UNDO\_RETENTION parameter.

## 60. Flashback Table

Oracle9i Database introduced the concept of a Flashback Query option to retrieve data from a point in time in the past, but it can't flash back DDL operations such as dropping a table. The only recourse is to use tablespace point-in-time recovery in a different database and then recreate the table in the current database using export/import or some other method. This procedure demands significant DBA effort as well as precious time, not to mention the use of a different database for cloning. Flashback Table feature in Oracle Database 10g, which makes the revival of a dropped table as easy as the execution of a few statements. Let's see how this feature works.

### Drop Table

```
First, let's see the tables in the present schema.
SQL> SELECT * FROM tab;
TNAME TABTYPE CLUSTERID

EMP TABLE
Now, we accidentally drop the table:
SQL> DROP TABLE EMP;
Table dropped.
Let's check the status of the table now.
SQL> SELECT * FROM TAB;
TNAME TABTYPE CLUSTERID

BIN$04LhcpndanfgMAAAAAANPw==$0 TABLE
```

The table EMP is gone but note the presence of the new table BIN\$04LhcpndanfgMAAAAAANPw==\$0. The dropped table EMP, instead of completely disappearing, was renamed to a system-defined name. It stays in the same tablespace, with the same structure as that of the original table. If there are indexes or triggers defined on the table, they are renamed too, using the same naming convention used by the table. Any dependent sources such as procedures are invalidated; the triggers and indexes of the original table are instead placed on the renamed table BIN\$04LhcpndanfgMAAAAAANPw==\$0, preserving the complete object structure of the dropped table.

The table and its associated objects are placed in a logical container known as the "**Recycle Bin**," which is similar to the one in your PC. However, the objects are not moved from the tablespace they were in earlier; they still occupy the space there. The recycle bin is merely a logical structure that catalogs the dropped objects. Use the following command from the SQL\*Plus prompt to see its content.

```
SQL> SHOW RECYCLEBIN
ORIGINAL NAME RECYCLEBIN NAME OBJECT TYPE DROP TIME

EMP BIN$04LhcpndanfgMAAAAAANPw==$0 TABLE 2004-02-
16:21:13:31
```

This shows the original name of the table, EMP, as well as the new name in the recycle bin, which has the same name as the new table we saw created after the drop table command. (Note: the exact name may differ by platform.) To reinstate the table, all you have to do is use the **FLASHBACK TABLE** command:

```
SQL> FLASHBACK TABLE EMP TO BEFORE DROP;
FLASHBACK COMPLETE.
```

```
SQL> SELECT * FROM TAB;
TNAME TABTYPE CLUSTERID

EMP TABLE
```

The table is reinstated effortlessly. If you check the recycle bin now, it will be empty. Remember, placing tables in the recycle bin does not free up space in the original tablespace. To free the space, you need to purge the bin using:

```
SQL> PURGE RECYCLEBIN;
```

If you want to drop the table permanently using: **SQL> DROP TABLE EMP PURGE;**

## 61. Flashback Database

### 61.1 Overview

Oracle 9i provided the capability to "flash back" to a prior view of the DB via queries performed against specific logical entities. For Example, if a user had accidentally added, modified or deleted a large number of rows incorrectly, it was now possible to view state of logical entity just before the operation had taken place.

This capability was limited, of course, by the amount of UNDO data retained in DBs UNDO segments and bounded by time frame specified by the UNDO\_RETENTION initialization parameter. Oracle 10g expands these logical flashback capabilities significantly.

However, when a DBA needed to return an entire database back to a prior state to recover from a serious logical error. For Example, when multiple invalid transactions within the same logical unit of work have affected the contents of several DB tables, a logical option was to perform an incomplete database recovery.

Since an incomplete recovery requires that all datafiles are first restored from the latest backup, and then a careful "roll forward" recovery through the appropriate archived and online redo logs until the appropriate point in time was reached, the database would be unavailable until this process was completed. With the addition of Flashback Database, Oracle 10g has significantly improved the availability of a database while it's restored and recovered to the desired point in time. These new features, however, do take some additional effort to plan for and set up configuration of Flash Recovery Area.

### 61.2 Enabling The Flash Recovery Area

Before any Flash Backup and Recovery activity can take place, the Flash Recovery Area must be set up. The Flash Recovery Area is a specific area of disk storage that is set aside exclusively for retention of backup components such as datafile image copies, archived redo logs, and control file autobackup copies. These features include:

**Unified Backup Files Storage:** All backup components can be stored in one consolidated spot. The Flash Recovery Area is managed via Oracle Managed Files (OMF), and it can utilize disk resources managed by Automated Storage Management (ASM). The Flash Recovery Area can be configured for use by multiple DB instances if so desired.

**Automated Disk-Based Backup and Recovery:** Once the Flash Recovery Area is configured, all backup components (datafile image copies, archived redo logs, and so on) are managed automatically by Oracle.

**Automatic Deletion of Backup Components:** Once backup components have been successfully created, RMAN can be configured to automatically clean up files that no longer needed (thus reducing risk of insufficient disk space for backups).

**Disk Cache for Tape Copies:** Finally, if our disaster recovery plan involves backing up to alternate media, the Flash Recovery Area can act as a disk cache area for those backup components that are eventually copied to tape.

**Flashback Logs:** The Flash Recovery Area is also used to store and manage flashback logs, which are used during Flashback Backup operations to quickly restore a database to a prior desired state.

**Sizing the Flash Recovery Area:** Oracle recommends that Flash Recovery Area must be sized large enough to include all files required for backup and recovery. If insufficient disk space is available, Oracle recommends that it be sized at least large enough to contain any archived redo logs that have not yet been backed up to alternate media.

The below table shows minimum and recommended sizes for the Flash Recovery Area based on the sizes of database files

| Database Element                               | Estimated Size (MB) |
|------------------------------------------------|---------------------|
| Image copies of all datafiles                  | 1200                |
| Incremental backups                            | 256                 |
| Online Redo Logs                               | 48                  |
| Archived Redo Logs retained for backup to tape | 96                  |
| Control Files                                  | 6                   |
| Control File Auto backups                      | 6                   |
| Flash Recovery Logs                            | 96                  |
| Recommended Size:                              | 1708                |
| Minimum Size:                                  | 96                  |

Based on these estimates, we will dedicate 2GB of available disk space so we can demonstrate a complete implementation of the Flash Recovery Area. Setting Up the Flash Recovery Area. Activation of the Flash Recovery Area specifying values for two additional initialization parameters:

- DB\_RECOVERY\_FILE\_DEST\_SIZE specifies the total size of all files that can be stored in the Flash Recovery Area. Note that Oracle recommends setting this value first.
- DB\_RECOVERY\_FILE\_DEST specifies the physical disk location where the Flashback Recovery Area will be stored. Oracle recommends that this be a separate location from the database's datafiles, control files, and redo logs. Also, note that if the database is using Oracle's new Automatic Storage Management (ASM) feature, then the shared disk area that ASM manages can be targeted for the Flashback Recovery Area.

Activating the Flash Recovery Area. It is obviously preferable to set up the Flash Recovery Area when a database is being set up for the first time, as all that needs to be done is to make the changes to the database's initialization parameters. However, if the Flash Recovery Area is being set up for an existing database, all that's required to do is issue the appropriate ALTER SYSTEM commands. This shows the changes we have made to the database's initialization parameter file, including an example of how to insure that an additional copy of the database's archived redo logs is created in the Flash Recovery area.

### 61.2.1 Setting up the Flash Recovery Area - closed database

```
Entries to add to database's INIT.ORA:
Flashback Backup and Recovery settings
db_recovery_file_dest_size = 2G
db_recovery_file_dest = '/disk3/oradata/fbrdata/wildb' # should be a
separate
area of disk
db_flashback_retention_target = 2880 # Will hold two days (2880 minutes)
worth
of Flashback
Activate this to transmit an extra copy of archived redo logs to Flash
Recovery Area
log_archive_dest_2 = 'location=use_db_recovery_file_dest'
log_archive_dest_state_2 = enable
```

This shows the commands to issue to set up the Flash Recovery Area when the database is already open before flashback logging has been activated.

### Setting up the Flash Recovery Area - open database

Be sure to set DB\_RECOVERY\_FILE\_DEST\_SIZE first...

```
SQL> ALTER SYSTEM SET db_recovery_file_dest_size = '5G' SCOPE=BOTH
 SID='*';
And then set DB_FILE_RECOVERY_DEST and DB_FLASHBACK_RETENTION_TARGET
SQL> ALTER SYSTEM SET db_recovery_file_dest =
 '/disk3/oradata/fbrdata/wildb'
 SCOPE=BOTH SID='*';
SQL> ALTER SYSTEM SET db_flashback_retention_target = 2880;
```

## 61.3 Enabling Flashback Database

As its name implies, Flashback Database offers the capability to quickly "flash" a database back to its prior state as of a specified point in time. Oracle does this by retaining a copy of any modified database blocks in flashback logs in the Flash Recovery Area.

A new flashback log is written to the Flash Recovery Area on a regular basis (usually hourly, even if nothing has changed in the database), and these logs are typically smaller in size than an archived redo log. Flashback logs have a file extension of .FLB.

When a Flashback Database request is received, Oracle then reconstructs the state of the DB just prior to the point in time requested using the contents of the appropriate flashback logs. Then the database's archived redo logs are used to fill in the remaining gaps between the last backup of the datafile and the point in time desired for recovery.

The beauty of this approach is that no datafiles need to be restored from backups; further, only the few changes required to fill in the gaps are automatically applied from archived redo logs. This means that recovery is much quicker than traditional incomplete recovery methods, with much higher database availability. It is worth noting the few prerequisites that must be met before a database may utilize Flashback Database features:

- The database must have flashback logging enabled, and therefore a Flash Recovery Area must have been configured. (For a RAC environment, the Flash Recovery Area must also be stored in either ASM or in a clustered file system.)
- Since archived redo logs are used to "fill in the gaps" during Flashback Database recovery, the database must be running in ARCHIVELOG mode.

**Activating Flashback Database.** Once the Flash Recovery Area has been configured, the next step is to enable Flashback Database by issuing the ALTER DATABASE FLASHBACK ON; command while the database is in MOUNT EXCLUSIVE mode, similar to activating a database in ARCHIVELOG mode.

**Setting Flashback Retention Target.** Once Flashback Database has been enabled, the DB\_FLASHBACK\_RETENTION\_TARGET initialization parameter determines exactly how far a database can be flashed back. The default value is 1440 minutes (one full day), but this can be modified to suit the needs of our database. For purposes of illustration, we have set our demonstration database's setting to 2880 minutes (two full days).

**Deactivating Flashback Database.** Likewise, issuing the ALTER DATABASE FLASHBACK OFF; command deactivates Flashback Backup and Recovery. Just as in the activation process, note that this command must be issued while the database is in MOUNT EXCLUSIVE mode.

This queries that display the status of the Flash Recovery Area, status of the related initialization parameters, and whether the database has been successfully configured for flashback.

### 61.3.1 Flash Recovery status queries

This will display what Flashback options are currently enabled for this database

```
SQL> SELECT name,value FROM v$parameter
 WHERE NAME LIKE '%flash%' OR NAME LIKE '%recovery%' ORDER BY NAME;
```

This will display what's the status of the Flash Recovery Area

```
SQL> SELECT name,space_limit /(1024*1024) spc_lmt_mb,space_used
 /(1024*1024)
 spc_usd_mb,space_reclaimable
 /(1024*1024) spc_rcl_mb,number_of_files
 FROM v$recovery_file_dest;
```

This will display is Flashback Database currently activated for this database

```
SQL> SELECT name,current_scn,flashback_on FROM v$database;
```

This will display what's the earliest point to which this database can be flashed back

```
SQL> SELECT oldest_flashback_scn,oldest_flashback_time,retention_target,
 flashback_size,estimated_flashback_size FROM
 v$flashback_database_log;
```

## 61.4 Storing Backups In Flash Recovery Area

Now that we have enabled the Flash Recovery Area and enabled flashback logging, we can next turn our attention to preparing the database to use flashback logs during a Flashback Database recovery operation.

### Configuring RMAN to use Flash Recovery Area

```
RUN {
 # Configure RMAN specifically to use Flash Recovery Area features
 CONFIGURE RETENTION POLICY TO REDUNDANCY 1;
 CONFIGURE BACKUP OPTIMIZATION ON;
 CONFIGURE CONTROLFILE AUTOBACKUP ON;}
```

The above example lists the RMAN commands we will need to issue to configure the database for Flash Recovery Area and Flashback Database use. Notice that we have not CONFIGURED a FORMAT directive for the RMAN channels used to create database backups; for these examples, we are going to let RMAN place all backup components directly in the Flash Recovery Area.

### 61.4.1 RMAN Daily Backup Scheme Using Image Copies

```
RUN {
 # RMAN Script: DailyImageCopyBackup.rcv
 # Creates a daily image copy of all datafiles and Level 1 incremental
 # backups
 # for use by the daily image copies

 # Roll forward any available changes to image copy files
 # from the previous set of incremental Level 1 backups
 RECOVER COPY OF DATABASE WITH TAG 'img_cpy_upd';

 # Create incremental level 1 backup of all datafiles in the database
 # for roll-forward application against image copies
 BACKUP INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG 'img_cpy_upd'
 DATABASE;
}
```

The above example implements Oracle's recommended daily RMAN backup scheme using datafile image copies and incrementally updated backups.

## Results of First Daily Backup

## List of Datafile Copies

| Key | File                                      | S  | Completion Time | Ckp | SCN      | Ckp       | Time            | Name |
|-----|-------------------------------------------|----|-----------------|-----|----------|-----------|-----------------|------|
| 41  | 1                                         | A  | 07-DEC-04       |     | 2119100  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | SYSTEM          |     | 0VDM2NP9 |           | .DBF            |      |
| 1   | 1                                         | A  | 20-NOV-04       |     | 2006057  | 20-NOV-04 | /disk3/RMANBKUP |      |
| 43  | 2                                         | A  | 07-DEC-04       |     | 2119143  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | UNDOTBS1        |     | 0VDM6MRV |           | .DBF            |      |
| 48  | 3                                         | A  | 07-DEC-04       |     | 2119180  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | DRSYS           |     | 0VDM9OP2 |           | .DBF            |      |
| 44  | 4                                         | A  | 07-DEC-04       |     | 2119156  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | EXAMPLE         |     | 0VDM7S0X |           | .DBF            |      |
| 46  | 5                                         | A  | 07-DEC-04       |     | 2119173  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | INDX            |     | 0VDM94ON |           | .DBF            |      |
| 60  | 6                                         | A  | 07-DEC-04       |     | 2119186  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | TOOLS           |     | 0VDMB270 |           | .DBF            |      |
| 47  | 7                                         | A  | 07-DEC-04       |     | 2119176  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | USERS           |     | 0VDM9F8W |           | .DBF            |      |
| 45  | 8                                         | A  | 07-DEC-04       |     | 2119166  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | XDB             |     | 0VDM8N66 |           | .DBF            |      |
| 51  | 9                                         | A  | 07-DEC-04       |     | 2119189  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | LMPT1           |     | 0VDMB6CL |           | .DBF            |      |
| 49  | 10                                        | A  | 07-DEC-04       |     | 2119184  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | LMPT3           |     | 0VDM9Y6J |           | .DBF            |      |
| 53  | 11                                        | A  | 07-DEC-04       |     | 2119193  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | LMPT2           |     | 0VDMBGJN |           | .DBF            |      |
| 52  | 12                                        | A  | 07-DEC-04       |     | 2119191  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | LMPT4           |     | 0VDMBBGW |           | .DBF            |      |
| 42  | 13                                        | A  | 07-DEC-04       |     | 2119127  | 07-DEC-04 |                 |      |
|     | /disk3/oradata/fbrdata/wilddb/datafile/01 | MF | SYSAUX          |     | 0VDM53DD |           | .DBF            |      |

## List of Archived Log Copies

| Key  | Thrd                                                        | Seq | S              | Low Time  | Name |
|------|-------------------------------------------------------------|-----|----------------|-----------|------|
| 148  | 1                                                           | 203 | A              | 05-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002030010493846699 | ARC |                |           |      |
| 149  | 1                                                           | 204 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002040010493846699 | ARC |                |           |      |
| 160  | 1                                                           | 205 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002060010493846699 | ARC |                |           |      |
| 151  | 1                                                           | 206 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002060010493846699 | ARC |                |           |      |
| 152  | 1                                                           | 207 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002070010493846699 | ARC |                |           |      |
| 153  | 1                                                           | 208 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002080010493846699 | ARC |                |           |      |
| 157. | 1                                                           | 209 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002090010493846699 | ARC |                |           |      |
| 155  | 1                                                           | 209 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archivelog/2004_12_06/01      | MF  | 1_209_0V9Q1HHJ |           | .ARC |
| 160  | 1                                                           | 210 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002100010493846699 | ARC |                |           |      |
| 161  | 1                                                           | 210 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archivelog/2004_12_08/01      | MF  | 1_210_0VH53GGG |           | .ARC |
| 156  | 1                                                           | 210 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archive/ZDC002100010493846699 | ARC |                |           |      |
| 157  | 1                                                           | 210 | A              | 06-DEC-04 |      |
|      | /disk3/oradata/fbrdata/wilddb/archivelog/2004_12_07/01      | MF  | 1_210_0VDOMOGQ |           | .ARC |

```

162 1 211 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archive/ZDC002110010493846699.ARC
163 1 211 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archivelog/2004_12_08/01_MF_1_211_0VH53NS2_.
ARC
158 1 211 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archive/ZDC002110010493846699.ARC
159 1 211 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archivelog/2004_12_07/01_MF_1_211_0VDOPPDT_.
ARC
164 1 212 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archive/ZDC002120010493846699.ARC
166 1 212 A 07-DEC-04
/disk3/oradata/fbrdata/wilddb/archivelog/2004_12_08/01_MF_1_212_0VH53V2V_.
ARC

```

Finally, this shows the abbreviated results of the first cycle's run of this backup scheme. Note that Oracle uses OMF naming standards for each backup component file - in this example, datafiles, the "extra copy" of the archived redo logs, and control file autobackups - stored in the Flash Recovery Area.

## 61.5 Flashback Database: An Example

Now that we have enabled flashback logging and have created sufficient backup components that are being managed in the Flash Recovery Area, it is time to demonstrate a Flashback Database operation.

Let's assume a worst-case scenario: One of my junior developers has been enthusiastically experimenting with logical units of work on what he thought was his personal development database, but instead mistakenly applied a transaction against the production database.

He has just accidentally deleted several thousand entries in the SH.SALES and SH.COSTS tables - just in time to endanger our end-of-quarter sales reporting schedule, of course! Here is the DML statements issued, along with the number of records removed:

```
SQL> DELETE FROM sh.sales WHERE prod_id BETWEEN 20 AND 80;
```

```
10455 rows deleted
```

```
Executed in 89.408 seconds
```

```

SQL> DELETE FROM sh.costs WHERE prod_id BETWEEN 20 AND 80;
6728 rows deleted
Executed in 18.086 seconds
SQL> COMMIT;
Commit complete
Executed in 0.881 seconds
Flashback Database to the rescue! Since we know the approximate date and
time that this transaction was committed to the database, we will issue
an appropriate FLASHBACK DATABASE command from within an RMAN session to
return the database to that approximate point in time. Here is a more
complete listing of the FLASHBACK DATABASE command set:
SQL> FLASHBACK [DEVICE TYPE = <device type>] DATABASE
 TO [BEFORE] SCN = <scn>
 TO [BEFORE] SEQUENCE = <sequence> [THREAD = <thread id>]
 TO [BEFORE] TIME = '<date_string>'
```

Note that we can return the database to any prior point in time based on a specific System Change Number (SCN), a specific redo log sequence number (SEQUENCE), or to a specific date and time (TIME).

If we specify the BEFORE directive, we can tell RMAN to flash the database back to the point in time just prior to the specified SCN, redo log, or time, whereas if the BEFORE directive is not specified, the database will be flashed back to the specified SCN, redo log, or time as of that specified point in time, i.e., inclusively.

First, we queried our database's Flashback Logs to determine which ones are available, found the log just prior to the user error and decided to flash back the database based on that log's starting SCN. The below example contains the query we ran against V\$FLASHBACK\_DATABASE\_LOGFILE to obtain this information.

#### Flashback Log Query

```
This displays what Flashback Logs are available
SQL> SELECT LOG#,bytes,first_change#,first_time
 FROM v$flashback_database_logfile;
```

Just as we would do during a normal point-in-time incomplete recovery, we then shut down the database by issuing the SHUTDOWN IMMEDIATE command, and then restarted the database and brought it into MOUNT mode via the STARTUP MOUNT command.

Instead of having to perform a restoration of datafiles as in a normal incomplete recovery, we instead simply issue the appropriate FLASHBACK DATABASE command to take the database back to the SCN we desired.

Once the flashback is completed, we could have continued to roll forward additional changes from the archived redo logs available; however, we simply chose to open the database at this point in time via the ALTER DATABASE OPEN RESETLOGS; command. Here are the actual results from the RMAN session:

```
C:>rman nocatalog target sys/@zdcdb
Recovery Manager: Release 15.1.0.2.0 - Production
Copyright (c) 1995, 2004, Oracle. All rights reserved.
connected to target database: ZDCDB (DBID=186357.1959)
using target database controlfile instead of recovery catalog
RMAN> FLASHBACK DEVICE TYPE = DISK DATABASE TO SCN = 2127725;
Starting flashback at 08-DEC-04
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=158 devtype=DISK
starting media recovery
media recovery complete
Finished flashback at 08-DEC-04
RMAN> ALTER DATABASE OPEN RESETLOGS;
database opened
```

To see what is really going on during the flashback and recovery process, we have also included a portion of the database's alert log. Note that Oracle automatically cleaned up after itself: Since they are of no use any longer after the RESETLOGS operation, Oracle even deleted the outmoded Flashback Logs from the Flashback Recovery Area.

## 61.6 Flashback Data Archive

The Flashback Archive Data feature in Oracle Database 11g – part of the Oracle Total Recall option - overcomes most of the above limitations. It automatically tracks every single change made to the data stored inside the database and maintains a secure, efficient and easily accessible archive of historical data.

The captured historical data can be retained for as long as the business demands and is easily accessible using standard SQL constructs. Historical data tracking can be enabled on both existing and new tables instantaneously and more importantly, in a completely application transparent manner.

Implemented natively inside the database, Flashback Data Archive presents a high-performance, storage optimized solution with a centralized management interface for satisfying data retention and change control requirements for organizations. The primary advantages of using Flashback Data Archive for historical data tracking include:

- 1. Application Transparent** – Enabling historical data capture on one or more tables can be done instantaneously with no or minimal application changes. Customers can therefore use this feature to capture historical data for both packaged as well as home grown applications.
- 2. Seamless Access** – Historical data can be easily accessed using familiar Flashback SQL constructs. Flashback Data Archive includes support for Flashback Queries. Applications can seamlessly query the history of table data, as it existed in different points in time. No special snapshots need to be taken to take advantage of this feature.
- 3. Security** - Historical data, once generated, is immutable to all users. This is enabled out-of-the-box and no special or extra setup is required. Access to the internal history tables is restricted to reads only. No DML operations are allowed to users, including administrators. Applications need not query the internal history tables directly as seamless access is provided through the Flashback Query mechanism.
- 4. Minimal performance overhead** – Regular user transactions will see negligible impact. Flashback Data Archive employs a lightweight mechanism to mark DML operations on tracked tables for archiving. The actual history generation and archiving is done asynchronously through a background process as explained later.
- 5. Storage optimized** – The history data is internally partitioned and highly compressed to reduce the storage footprint. Flashback Data Archive employs a highly efficient compression scheme to compress the internal history tables. In addition, it automatically partitions the internal history tables based on a range-partitioning scheme. Both compression and partitioning in flashback data archive are managed automatically and require no special administration.
- 6. Centralized Management** – Flashback Data Archive provides a centralized and policy-based management interface to automate a number of ongoing administrative tasks. With Flashback Data Archive, you can easily group tables and set a common retention policy. New tables will automatically inherit the retention parameter from the flashback data archive it is a part of. Oracle will automatically purge aged-out history data for all tracked tables based on the specified retention. This frees up the administrator from the repetitive management of history data and avoids costly errors associated with manual maintenance such as purging wrong history.

### 61.6.1 Oracle Flashback Technology

Oracle's Flashback technology provides a set of functionality to access data as of a time in the past and recover from human errors. Flashback technology is unique to the Oracle Database and supports recovery at any level of granularity including the row, transaction, table, and database wide.

Using flashback features you can query past versions of data as well as perform change analysis and self-service repair to recover from logical corruption while the database is online. Flashback technology includes the following features:

- Flashback query allows the user to query data at some point-in-time in the past to reconstruct lost data that may have been deleted or changed by accident.
- Flashback version query provides a mechanism to view changes made to the database over time at the row level.
- Flashback transaction query provides a mechanism to view changes made to the database at the transaction level.
- Flashback database is a new strategy for doing point-in-time recovery. It quickly rewinds an Oracle database to a previous time to correct any problems caused by logical data corruption or user error.
- Flashback table provides the ability to recover a table, or a set of tables, to a specified point in time in the past very quickly and easily.
- Flashback drop provides a safety net when dropping objects as you can very quickly and easily undrop a table and its dependent objects.

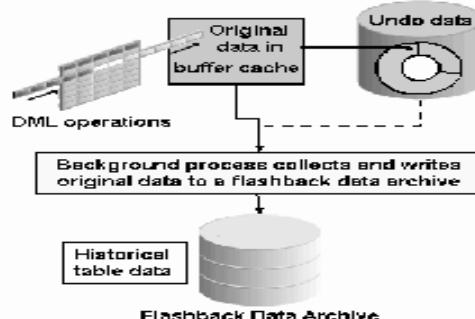
Flashback Data Archive is the newest addition to the Flashback family. In Oracle Database 11g Release 1, Flashback Data Archive provides support for querying the history data using the familiar Flashback Query SQL construct 'AS OF'.

An important distinction that distinguishes Flashback Data Archive from the other Flashback features is that history data can be accessed for any specified duration, far and beyond the amount of retained undo or flashback log data. Historical data can be retained for any length of time as required for compliance or other business reasons. With Flashback Data Archive, history data is always available and readily queryable.

## 61.7 Architecture

Oracle implements a multi-versioning mechanism that ensures read-consistency while maintaining high degree of concurrency. When DML operations such as insert, update or delete happen on data, Oracle writes the data into an undo tablespace that is used not only for transaction rollbacks but also guaranteeing read consistency in a concurrent environment.

Flashback Data Archive works with the undo data as with many other Flashback features. As stated above, however, the historical data retain in the flashback data archive is no limited by the size of undo tablespace. History generation is implemented in a non-intrusive manner through a new background process called 'FBDA'.



After a table has been enabled for history tracking with Flashback Data Archive, all transactions and the corresponding undo records are marked for archival. In order to guarantee that every transaction is archived, the undo records are not recycled until the history is generated and stored in the database.

The process sleeps and wakes up at system-managed intervals and processes the undo data marked for archival. After 'FBDA' generates the history, the transactions and undo records are candidates for recycling. This asynchronous FBDA process ensures negligible impact on the overall transaction or foreground performance.

The 'FBDA' process intelligently adjusts its sleep interval based on the system undo generation rate. As transaction activity increases, 'FBDA' automatically reduces the sleep interval from the default sleep time of 5 minutes.

For better performance, 'FBDA' process also adjusts its sleep time in order to maximize undo data reads from the buffer cache. Flashback Data Archive also uses an internal partitioning scheme for all the historical data for better performance. It also employs table compression to reduce the storage footprint of the historical data that could easily grow into hundreds of terabytes for longer retention times.

## 61.8 Understanding Flashback Data Archive

A Flashback Data Archive is a logical container for managing historical information for related tables. It is a new data dictionary object in Oracle Database 11g that defines archive storage and data purging policies. A Flashback Data Archive can span multiple tablespaces. Administrators can define the amount of space a flashback data archive can use in each tablespace using the 'QUOTA' parameter.

Multiple flashback data archives can be created as needed to implement different archiving policies. Each flashback data archive contains a 'RETENTION' parameter that specifies the duration for retaining the historical changes. Flashback Data Archive guarantees that historical

data will be retained for the duration specified by 'RETENTION' and automatically purges out aged historical data. Additionally, administrators, with the necessary privileges, can purge historical data in an ad-hoc manner.

Flashback Data Archive creates an internal history table for every tracked table. The internal history table is a replica of the tracked table with additional timestamp columns. When one or more columns in the tracked table are updated, a new row is inserted into the history table that is the before-image of the row before the transaction.

It is important to note that UPDATE and DELETE operations generate a new record in the history table. Flashback Data Archive does not create a new history record for INSERT operations. The internal history table is partitioned for better performance.

No modifications are allowed to internal partitions. The internal history tables are compressed to reduce the disk space requirements. Applications or users need not access the internal history tables directly. The 'AS OF' SQL construct can be used to seamlessly query the historical data.

### 61.8.1 Viewing Flashback Data Archive Data

Table lists and briefly describes the static data dictionary views that you can query for information about Flashback Data Archives.

Table: Static Data Dictionary Views for Flashback Data Archives

| View                       | Description                                                                 |
|----------------------------|-----------------------------------------------------------------------------|
| *_FLASHBACK_ARCHIVE        | Displays information about Flashback Data Archives.                         |
| *_FLASHBACK_ARCHIVE_TS     | Displays tablespaces of Flashback Data Archives.                            |
| *_FLASHBACK_ARCHIVE_TABLES | Displays information about tables that are enabled for flashback archiving. |

### 61.9 Flashback Data Archive Enhancement in Oracle 11g R2

Oracle Database 11g Release 2 (11.2) users can now use most DDL commands on tables that are being tracked with Flashback Data Archive. This includes:

- Add, Rename, Modify Column
- Drop, Truncate Partition
- Rename, Truncate Table
- Add, Drop, Rename, Modify Constraint

For more complex DDL (for example, upgrades and split table), the Disassociate and Associate PL/SQL procedures can be used to temporarily disable Total Recall on specified tables. The Associate procedure enforces schema integrity after association; the base table and history table schemas must be the same.

This feature makes it much easier to use the Total Recall option with complex applications that require the ability to modify the schema.

## 61.10 Flashback Data Archive Enhancement in Oracle 12C R1

### 61.10.1 User context tracking

This new feature allows tracking the user context and makes it easier to know which user made which changes in a table.

To track user-contexts, Oracle has created two new subprograms in the DBMS\_FLASHBACK\_ARCHIVE package as well as a new table.

#### "set\_sys\_context" procedure

This procedure allows to set the level of user-context to return. It has only one parameter (LEVEL).

The possible values for the LEVEL parameter are:

- **ALL:** Obtain all information from sys\_context
- **TYPICAL:** Provides the user id, the global user id and the hostname
- **NONE:** Nothing

To enable the database to capture user-context, the procedure must be run once with "ALL" or "TYPICAL" value. Example for ALL:

```
SQL> exec dbms_flashback_archive.set_context_level(level=>'ALL');
"get_sys_context" function
```

This function returns the user-context only if the procedure set\_sys\_context was previously run with a LEVEL different of NONE.

The function has 3 parameters:

- **XID:** Transaction identifier provided by the archive history table
- **NAMESPACE:** Provided by SYS\_FBA\_CONTEXT\_LIST
- **PARAMETER:** Parameters of namespace

### 61.11 Database Hardening

With Oracle 11g, FDA had to be managed at table level. Starting with Oracle 12c, it is possible to create logical groups of tables for an application: Enabling or disabling of FDA is performed at application level and it prevents the listing of all application tables to enable or disable FDA for each one.

To make the best out of this new feature, the first step is to create and enable an application group. In my example, the application is named "PRODUCTS":

```
SQL> exec dbms_flashback_archive.register_application
(application_name=>'PRODUCTS',
flashback_archive_name->'FBA1');
```

## 62. SQL\*Loader

### 62.1 Overview

SQL\*Loader is the primary method for quickly populating Oracle tables with data from external files. It has a powerful data parsing engine that puts little limitation on the format of the data in the datafile. SQL\*Loader is invoked when we specify sqldr command or use the Enterprise Manager interface. SQL\*Loader is an integral feature of Oracle databases and is available in all configurations.

### 62.2 Key Features

- Load data across a network. This means that a SQL\*Loader client can be run on a different system from the one that is running the SQL\*Loader server.
- Load data from multiple datafiles during the same load session
- Load data into multiple tables during the same load session
- Specify the character set of the data
- Selectively load data
- Load data from disk, tape, or named pipe
- Specify the character set of the data
- Generate sophisticated error reports, which greatly aid troubleshooting
- Load arbitrarily complex object-relational data
- Use either conventional or direct path loading.

### 62.3 File Types

#### SQL\*Loader Control File

The control file is a text file written in a language that SQL\*Loader understands. The control file tells SQL\*Loader where to find the data, how to parse and interpret the data, and where to insert the data.

#### Input Data and Datafiles

SQL\*Loader reads data from one or more files specified in the control file. From SQL\*Loader's perspective, data in datafile is organized as records. A particular datafile can be in fixed record format, variable record format, or stream record format. The chosen format depends on data and depends on flexibility and performance necessary for the job.

### 62.4 Load Methods

- Conventional Path
- Direct Path
- External Table

#### Conventional Path Load

Conventional path load builds an array of rows to be inserted and uses SQL INSERT statement to load data. During conventional path loads, input records are parsed according to field specifications, and each data field is copied to its corresponding bind array. When the bind array is full (or no more data is left to read), an array insert is executed.

#### Direct Path Load

A direct path load builds blocks of data in memory and saves these blocks directly into extents allocated for table being loaded. A direct path load uses field specifications to build whole Oracle blocks of data, and write blocks directly to Oracle datafiles, bypassing much of data processing that normally takes place.

Direct path load is much faster than conventional load, but entails some restrictions. A parallel direct path load allows multiple direct path load sessions to concurrently load the same data segments. Parallel direct path is more restrictive than direct path.

## External Table Load

An external table load creates an external table for data in a datafile and executes INSERT statements to insert the data from the datafile into the target table.

There are two advantages of using external table loads over conventional path and direct path loads:

- An external table load attempts to load datafiles in parallel. If a datafile is big enough, it will attempt to load that file in parallel.
- An external table load allows modification of the data being loaded by using SQL functions and PL/SQL functions as part of the INSERT statement that is used to create the external table.

## 62.5 Important Points

- Sqldr is a utility to load data from ASCII files into Oracle Database.
- \$ sqldr userid=scott/tiger control=test1.ctl
- (In this e.g., "test1.ctl" is the controlfile, which dictates what should be done by the "sqldr" utility)
- The file, which is treated as INFILE, should be consistent (e.g. a comma delimited or white space or fixed length), but the file should be consistent all the way through.
- We specify all rules & regulations in another file, which is known as Controlfile and passed to "sqldr" utility.
- In the controlfile we talk about (usually named like test1.ctl)
  - Where the data is coming from → INFILE
  - Where the data should be loaded → INTO a Oracle table
  - Whether the data should be appended or replaced or truncate-table before we load this new data.
- Rows can be loaded in 3 ways:
  - By deleting existing records in the table → TRUNCATE
  - Can Append the table with new data → APPEND
  - Insert & Update (update will take place if the row is already there) → REPLACE
- Replace is not a suggested option. Anyway most of the time we load the data into a temporary table, we try to do 'Truncate' and load fresh data from the file.
  - Describe file delimiter (e.g. comma or semi-colon or tilde (~))
  - Incase if there is an errors in the file (e.g. having char for number fields), where these error lines should be written. This is very important since at the end of the load we are going to look for the cause for all these bad lines.
    - This file is called as → "bad" file → test1.bad
    - If we want to purposefully suppress some lines (e.g. want to load only "Male" lines but not "Female" lines since they go to another table), in that case we can use "WHEN" clause in controlfile. So that all the un-wanted lines will go to "disc" file instead of going to the table. Usually this file (→test1.disc) is called as discard file and this is very different from "bad" file
      - Bad means → line is not fine to load,
      - Disc means → line is not qualified to load

We take this discard (test1.disc) file as "input" file while loading our "Female Table". Doing this we are saving time, since if we go back to original file to perform loading for "Female" lines, we need to omit "Male Lines", which is slower compared to loading lines from "disc" file (created from original load).

- In some cases we may want to skip top lines since they may be headings. This also can be specified as an option (SKIP).
- In some cases though we have 1000+ lines in our input file, we are only interested in loading 600 lines. This can be specified as number of lines to be loaded from "input" file (LINES TO LOAD).

- We can also opt for "Direct" load, which means don't go through Oracle's conventional path, but just load data directly into the Table. By doing so, in case if we are violating any "FK" rules, then the appropriate "Constraint" will be "Disabled" before loading of the data. This constraint you have to "Re-Enable" manually after fixing any missing Master Tables content. E.g. my boss asked me to load EMP Table from a File, which contains some DEPTNO, which are not in DEPT Table. He knows about it and says it's OK and load. (Idea is, he has a different file for DEPT Table, which he planned to give me later). So if I perform regular load (by choosing Conventional-Path), then all lines coming from this ASCII File with DEPTNO that are not there in DEPT Table, will not be loaded. But he also wants to see data since he knows that after loading DEPT, that FK problem won't be there. In these situations, we choose "DIRECT" load option, but keep in our mind that, this'll "Disable" the "FK-Constraint", which should be "RE-ENABLED" by us.
- We can also perform this load in "Parallel" when we have Millions of records to load in Table. For e.g. if we have a Table with 3 extents and all these 3 extents are in 3 different datafiles (belonging to same TS), and say we have 3 CPUs on this Server, in this case Loading data in "Parallel" will make much more sense, since there are 3 CPUs which can split the loading Job into 3 pieces and they can work with 3 Disks on which we have 3 extents (for the same table). Usually we use this for loading high-end Data Warehousing Tables with huge data.
- True power for SQL\*Loader is in handing SQL-Functions on top of ASCII File. This is really amazing since, we can load data according to our needs, which means we can do data massaging before loading itself. For e.g. let's say we are interested in making COMM as 0, in case if it is NULL. Then we can simply say decode (: comm, null, 0,comm) what it does, it checks if COMM is NULL or not, if it is, it returns 0 otherwise whatever is COMM value; it is loaded into our Table. Similarly we can use all SQL-Functions like SUBSTR, RPAD etc.  
E.g. If SSN is 333-33-3333 this can be altered without dashes (-), since our column in Oracle Table is defined as NUMBER, by using SQL Function "Replace".
- In our regular SQL we have "Where" clause. This is so powerful since we have power in doing whatever we want. SQL\*Loader is also giving a similar functionality by giving "WHEN". But "WHEN" is not that powerful compared to "WHERE" Clause. In "WHERE" clause we can say SAL > 1000 (which basically means where salary is more than \$1000) this we can't get using "WHEN", since using "WHEN", either it should be "Equal to" or "Not-Equal-To". SAL = 1000 (this can be done, but we can't use < or > signs). We use this "WHEN" clause to qualify or disqualify each row before loading that row into Table. All unqualified rows will be written to "test1.disc" file. This "discard" file may become "infile" for loading into another table. So try not to lose it.
- All the above rules & regulations will be enforced and the total action will be written to a file called "LOG" file. (In this case test1.log). After each load DBA must visit this LOG file, since here you can find out
  - How many lines are there in "input" file
  - " " " Loaded successfully into Table.
  - " " " Written to "disc" file (Omitted lines)
  - " " " Written to "bad" file (not qualified for the load)
  - " " Constraints are disabled (so that you can plan enabling those constraints after you load the master table corresponding records) and finally how much time we have spent in loading this data.
- Most of the time, these files may be coming from your Mainframe or may be from another Server. So basically a Programmer at the other-end is giving you this data. As a DBA, after performing the load, you should see how many "bad" lines have taken place, and what the cause is for it. May be you should talk to that Programmer about these "bad" entries, so that he may fix his "Program" which may stop sending these "bad" lines as part of the "input" file for you from the next time onwards.
- Most of the companies we use SQL\*Loader as part of daily routine. Always try to create a Unix-Shell program to do this loading. But keep one thing in our mind, whichever is "bad", "disc" & "log" files, should better have dates with them. So that if your boss wants to know what happened on Tuesday's data-load, we can always show him since files are saved like this. For E.g. **test1\_Mar\_13.log, test1\_Mar\_13.bad, test1\_Mar\_13.disc** which means Shell program will automatically generate these file names depending on System Date.
- Always try to mention huge-buffer size while performing big loading. Otherwise SQL\*Loader is going to use only 256Kb buffer, which means SQL\*Loader reads 256Kb worth of data from the ASCII File and writes to the Database and performs "Commit". Since 256Kb is very small amount of data, the entire load may take very long time with so much commits in between. To avoid so much "Commits" and to have big buffers, use "bindsize, rows" as the parameters for your "sqlldr"

- command. This will ensure in using huge Buffer size so that "Commit-Frequency" is reduced, so that performance is increased.
- Go to \$ORACLE\_HOME/rdbms/demo directory. Here you will find some \*.ctl files & \*.dat files, which may give you more details with examples.
  - Being a DBA, it's necessary to understand why some rows have failed and should be processed again after fixing the bugs. Bugs can be of different types:
    - Primary Key/UK errors (Duplicate records)
    - FK Problems.
    - Unable to get next Extent (need to add another datafile to TS)
    - Rollback segment problems
    - Column size mismatching
    - Destination Table is missing
    - Userid and/or password is/are wrong.
  - We can even do remote loading using SQL\*Net Aliases. So if the file is on one server and the Database is on another, still without copying the file from Server 1 to Server 2, we can use SQL\*Net Alias and dump the data into the destination table.  
`$ sqldr userid=scott/tiger@devl2prod control=test1.ctl`

## 62.6 Oracle 12C SQL Loader Express Mode Basic Example

### 62.6.1 SQL Loader - Oracle 12C - Express Mode

I got introduced to the new capabilities of SQL Loader in Oracle 12C today. One such feature is the express mode that is available in 12C and in this post let us experiment this new approach with a basic example. To get this example working, you would require a working version of Oracle 12C available at your end. Some theory to start with, that explains what all this express mode is about;

### 62.6.2 Express Mode - Basic Definition

Express mode as the name suggests simplifies the SQL loader operation by offering a mode of loading, that requires no control files. All it warrants is a data file with the name which is same as the name of the table that you are loading. For this basic example, it would be enough to know this much (we will see some advanced usage in subsequent posts). Let us see how this mode works now.

### Working Example - Express Mode

We will create a test table in HR schema, which we will use for this post. The DDL of the table is shown below:

```
Sql> create table test (text1 varchar2(10), counter number)
```

Tablecreated.

Next, you will have to create a simple dat file the contents of which we will load into this table via SQL Loader.

As we have two columns you have to specify the column values in the same order in the file. The separator between two columns in the file should be a comma. A sample file is shown below:

```
[oracle@localhost sqldrtest]$ more test.dat
time,1
test,2
trim,3
twin,4
```

That is all you need. You are ready to use SQL Loader to load this data into the table - through the express mode. Run the command as shown below;

```
[oracle@localhost sqldrtest]$ sqldr hr table=test
```

Note that we are specifying the schema where the load has to happen, followed by the table name. You have to keep the test.dat file in the same directory where you are triggering the load. When you enter this command, Oracle will prompt you for the password for hr schema. Enter the password and viola - you have completed the load. A sample output that comes on the screen is provided below:

```
[oracle@localhost sqldrtest]$ sqldr hr table=test
```

Password:

```
SQL*Loader: Release 15.1.0.1.0 - Production on Fri Jul 4 07:42:41 2014
```

```
Copyright (c) 1982, 2013, Oracle and/or its affiliates. All rights reserved.
```

```
Express Mode Load, Table: TEST
```

```
Path used:External Table, DEGREE_OF_PARALLELISM=AUTO
```

Table TEST:

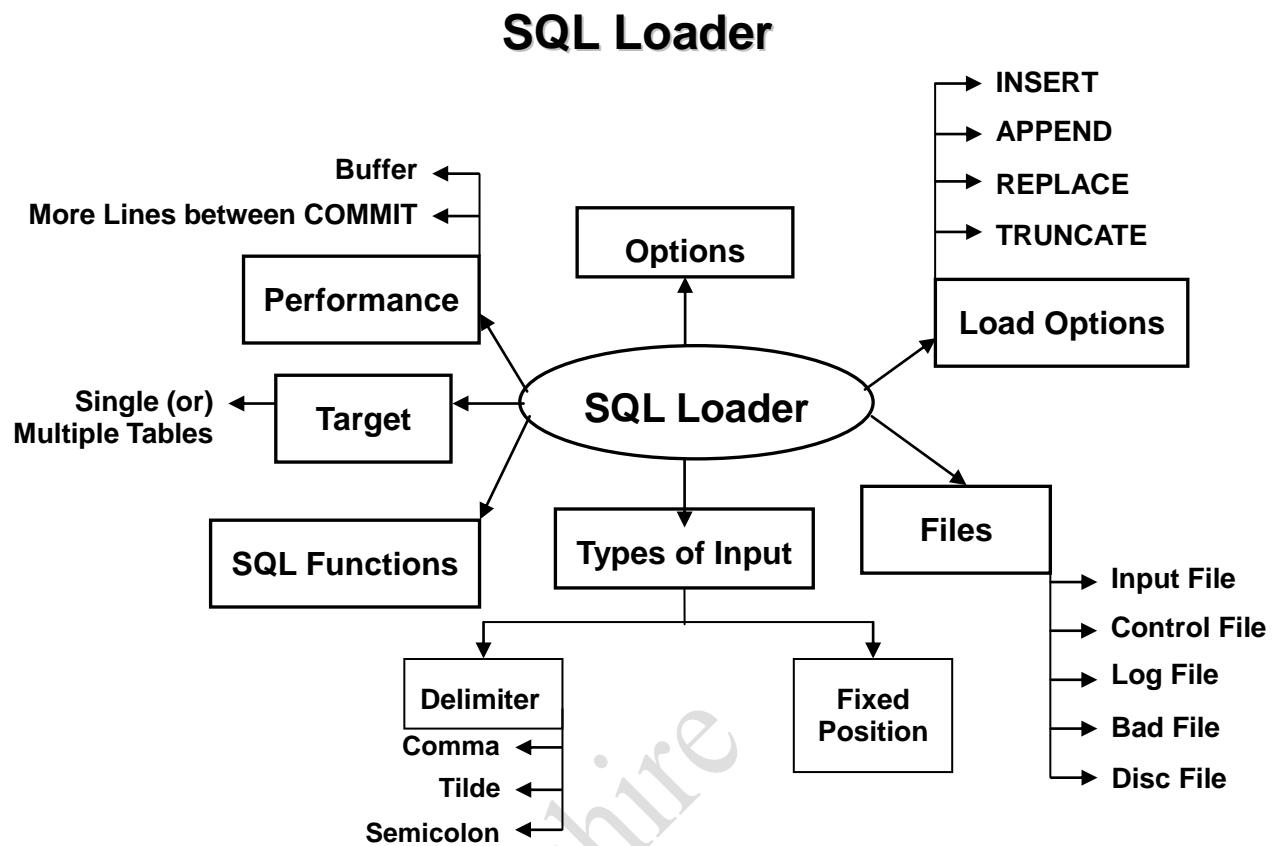
4 Rows successfully loaded.

Check the log files:

test.log

test\_%p.log\_xt

For more information about the load



## 63. Database Auditing

### 63.1 Overview

Auditing is done to check regular and suspicious activity on the database. When your auditing purpose is to monitor for suspicious database activity, consider the following guidelines:

- Audit generally, then specifically: When starting to audit for suspicious database activity, it is common that not much information is available to target specific users or schema objects. Therefore, audit options must be set more generally at first. Once preliminary audit information is recorded and analyzed the general audit options should be turned off and more specific audit options enabled.
- Protect the audit trail: When auditing for suspicious database activity, protect the audit trail so that audit information cannot be added, changed or deleted without being audited.

When auditing normal database activity, consider the following guidelines:

- Audit only pertinent actions: To avoid cluttering meaningful information with useless audit records and reduce the amount of audit trail administration, only audit the targeted database activities.
- Archive audit records and purge audit trail: Once you have collected the required information, archive the audit records of interest and purge audit trail of this information.

In order to invoke auditing you have to set these parameters in INIT.ORA.

- audit\_trail=TRUE /DB /OS
- audit\_file\_dest= /users/DEMO/audit

Audit\_trail can be set to three parameters:

|      |                                                      |
|------|------------------------------------------------------|
| DB   | It stores the audit information in the database only |
| OS   | Stores the audit information in physical files       |
| None | Auditing is disabled                                 |

AUDIT\_TRAIL enables or disables the writing of rows to the audit trail. Audited records are not written if the value is NONE or if the parameter is not present. The OS option enables system-wide auditing and causes audited records to be written to the operating system's audit trail.

The DB option enables system-wide auditing and causes audited records to be written to the database audit trail (the SYS.AUD\$ table). The values TRUE and FALSE are also supported for backward compatibility. TRUE is equivalent to DB, and FALSE is equivalent to NONE.

**Creating and Deleting the Database trail views:** The database audit trail (sys.AUD\$) is a single table in each Oracle's Data Dictionary. To help you view meaningful auditing information in this table, several predefined views are provided. You have to run CATAUDIT.SQL as sys to create audit trail views. Auditing can be done on all types of commands.

### 63.2 Types of Auditing:

- Privilege level of Auditing
- Object level of Auditing
- Statement level of Auditing

#### Focusing Statement, Privilege and Object Auditing:

Oracle allows you to focus statement, privilege and object auditing in three areas

Successful and unsuccessful executions of the audited SQL statement  
BY SESSION and BY ACCESS auditing

For specific users or for all users in the database (statement and privilege auditing only)

**Auditing Successful and unsuccessful statement Execution:** For a statement, privilege or object auditing oracle allows the selective auditing of successful executions of statements, unsuccessful attempts to execute statements, or both.

Using either form of the AUDIT command, you can include

- The WHENEVER SUCCESSFUL option, to audit only successful executions of the audited statement.
- The WHENEVER NOT SUCCESSFUL option, to audit only unsuccessful executions of the audited statement.
- Neither of the previous options, to audit both successful and unsuccessful executions of the audited statement

**Auditing BY SESSION versus BY ACCESS:** Most auditing options can be set to indicate how audit records should be generated if the audited statement is issued multiple times in a single user session.

- BY SESSION: For any type of audit, BY SESSION inserts only one audit record in the audit trail, per user and schema object, during the session that includes an audited action.
- BY ACCESS: Setting audit BY ACCESS inserts one audit record into the audit trail for each execution of an auditable operation.

Ex: the single audit trail contains 4 records for four select statements if the audit is on select.

**Privilege level of AUDITING:** Monitoring on suspicious activities done at the system privileges granted to any user.

- To audit sessions of STEEVE and LORI:

```
SQL> AUDIT CREATE SESSION BY steeve, lori;
```

- To audit create tables by Scott:

```
SQL> AUDIT CREATE TABLE BY scott;
```

To see the specifications that were there we need to query DBA\_PRIV\_AUDIT\_OPTS

```
SQL> SELECT * FROM DBA_PRIV_AUDIT_OPTS;
```

If any of the users who are under auditing has performed the audited operation then he can be queried from dba\_audit\_trail.

```
SQL> SELECT username, action_name, returncode FROM dba_audit_Trail;
```

**Object level of Auditing:** Object auditing is the selective auditing of specific DML statements. Object auditing audits the operations permitted by schema object privileges, such as SELECT or DELETE statements on a given table.

Consider the following series of SQL statements:

Auditing on select by all.

```
SQL> AUDIT SELECT ON emp;
```

- To audit select and delete only whenever any user access the object.

```
SQL> AUDIT SELECT, DELETE ON scott.emp BY ACCESS;
```

- To audit insert by all only when they are successful

```
SQL> AUDIT INSERT ON scott.emp WHENEVER SUCCESSFUL;
Auditing on all types of DML statements by all users
SQL> AUDIT INSERT, UPDATE, DELETE ON SCOTT.EMP;
```

To see the specifications that were there we need to query dba\_obj\_audit\_opts

```
SQL> SELECT * FROM dba_obj_audit_opts WHERE object_name='emp' ;
```

If any of the users who are under auditing has performed the audited operation then he can be queried from dba\_audit\_trail.

```
SQL> SELECT username, action_name, returncode FROM dba_audit_Trail;
```

**STATEMENT AUDITING:** Statement auditing is the selective auditing of related groups of statements that fall into two categories.

- DDL statements, regarding a particular type of database structure or schema object, but not a specifically named structure or schema object (for example, audit table audits all create and drop table statements)
- DML statements, regarding a particular type of database structure or schema object, but not a specifically named structure or schema object (for example, audit select table audits all select .. from TABLE/VIEW statements, regardless of the table or view.
- Queries, which work with statement level of auditing:
- audit on any statement which has table key word in it  

```
SQL> AUDIT TABLE BY SCOTT;
```
- To audit at statement by all users when ever not successful  

```
SQL> AUDIT SELECT TABLE WHENEVER NOT SUCCESSFUL;
```
- To audit all users by session  

```
SQL> AUDIT SESSION;
```
- To audit Scott by access  

```
SQL> AUDIT VIEW BY SCOTT BY ACCESS;
```

To see the specifications that were there we need to query dba\_stmt\_audit\_opts

```
SQL> SELECT * FROM dba_STMT_audit_OPTS;
```

If any of the users who are under auditing has performed the audited operation then he can be queried from dba\_audit\_trail.

```
SQL> SELECT username, action_name, returncode FROM dba_audit_Trail;
```

### Exercises

```
To audit Insert, Update, Delete on EMP table of SCOTT:
SQL> AUDIT INSERT, UPDATE, DELETE ON scott.emp;
```

To audit all unsuccessful select, insert and delete statements on all tables and unsuccessful uses of the execute any procedure system privilege, by all database users, by access:

```
SQL> AUDIT SELECT ANY TABLE, INSERT ANY TABLE, DELETE ANY TABLE,
EXECUTE ANY PROCEDURE BY ACCESS WHENEVER NOT SUCCESSFUL;
```

To disable audit:

```
SQL> NOAUDIT;
To disable audit for a session:
SQL> NOAUDIT SESSION;
To disable select table, insert table, delete table:
SQL> NOAUDIT INSERT TABLE, SELECT TABLE, DELETE TABLE;
To view the information of auditing you have to query:
SQL> SELECT * FROM all_def_audit_OPTS;
SQL> DELETE sys.aud$;
SQL> SELECT username, obj_name, action_name, ses_actions
 FROM sys.dba_audit_objects;
SQL> SELECT username, logoff_time, logoff_lread, logoff_pread,
 logoff_lwrite,
 logoff_dlock FROM sys.dba_audit_session;
```

## 63.3 Unified and Conditional Auditing

Oracle Database 12c Unified Auditing enables selective and effective auditing inside the Oracle database using policies and conditions. The new policy based syntax simplifies management of auditing within the database and provides the ability to accelerate auditing based on conditions. For example, audit policies can be configured to audit based on specific IP addresses, programs, time periods, or connection types such as proxy authentication. In addition, specific schemas can be easily exempted from auditing when the audit policy is enabled.

New roles have been introduced for management of policies and the viewing of audit data. The AUDIT\_ADMIN and AUDIT\_VIEWER roles provide separation of duty and flexibility to organizations who wish to designate specific users to manage audit settings and view audit activity. The new architecture unifies the existing audit trails into a single audit trail, enabling simplified management and increasing the security of audit data generated by the database. Audit data can only be managed using the built-in audit data management package within the database and not directly updated or removed using SQL commands. Three default policies are configured and shipped out of the box. Oracle Audit Vault and Database Firewall 15.1.1 is integrated with the new Oracle Database 12c Unified Auditing for audit consolidation.

## What is Unified Auditing in Oracle Database 12c?

Auditing is not new in 12c. It has been there with database server for quite sometime now. But it was a pain for DBA to manage auditing and auditing related data. There were many types of auditing for example Standard auditing, Fine Grained Auditing, Privileged user access etc. Not only you were required to enable or disable all of them individually but also all of them had their own location and format to store audit records. Unified auditing is introduced to mitigate all of these problems and also make the whole audit process more effective.

Starting from 12c you can now use Unified auditing to keep track of all audit data in a single audit trail. The unified auditing trail can have auditing data from following audit sources.

Audit records (including SYS audit records) from unified audit policies and AUDIT settings

- Fine-grained audit records from the DBMS\_FGAPI/SQL package
- Oracle Database Real Application Security audit records
- Oracle Recovery Manager audit records
- Oracle Database Vault audit records
- Oracle Label Security audit records
- Oracle Data Mining records
- Oracle Data Pump
- Oracle SQL\*Loader Direct Load

All auditing data is stored inside the SYSAUX tablespace and made available in the read only format via a view named UNIFIED\_AUDIT\_TRAIL. Oracle recommends that you store the audit data in a separate tablespace especially created for auditing purposes. You can change the default location of auditing using the **DBMS\_AUDIT\_MGMT.SET\_AUDIT\_TRAIL\_LOCATION** procedure.

The audit records are recorded in audit trail if the database is open for read write. However if for some reason the database is in read only mode then all the audit records are stored in a new format at the operating system level. The location for audit records under such situations is \$ORACLE\_BASE/audit/\$ORACLE\_SID.

Auditing is controlled by defining the audit policies. You can combine the audit settings to form a complete audit policy or you can use the predefined policies supplied by Oracle itself. You can also go ahead and create fine grained auditing policies for individual objects and users.

## Mixed Mode Auditing

Auditing is enabled by default in 12c, unlike the previous versions where you had to turn it on manually. The default auditing mode is the mixed mode auditing. Mixed mode auditing allows you to use the traditional auditing alongside with the new Unified auditing. This serves as a good mediator for an easy and hassle free switch to the preferred Unified auditing.

The previous parameters used for different types of auditing are only valid if you are using mixed mode auditing.

## Auditing for CDB and PDB

Auditing policies can be applied at the CDB level or they can be applied to the individual PDB's databases. The only point worth noting is that you cannot have fine grained auditing policies for the root database. Fine grained policies are only applicable to the individual PDB's.

## Benefits of Unified Auditing

Below are the key benefits attached to Unified auditing.

Once enabled, Unified auditing frees you from setting the different initialization parameters. It is always on.

- As there is only one audit trail, managing the auditing data is whole a lot easier.
- The actual audit process actually gets a lot simpler. Now you have only one audit trail and only one view to query for all types of audit records. Whether it is the record for an insert statement or some action from the SYS user, all are in one location making the audit process simpler. The AUDIT\_TYPE column shows the type of the audit record.
- The writing of the audit records to disk was also caused a problem for database performance as well. But Unified auditing works in a Queued Write mode which means that all the records are initially recorded in the SGA instead of the immediate write to disk. Although you can change this behavior but the former one is recommended and has huge performance gains.
- The new auditing policies has a lot of flexibility. It allow you to fine grain an audit policy and also to point out exclusions and exceptions from that particular policy.

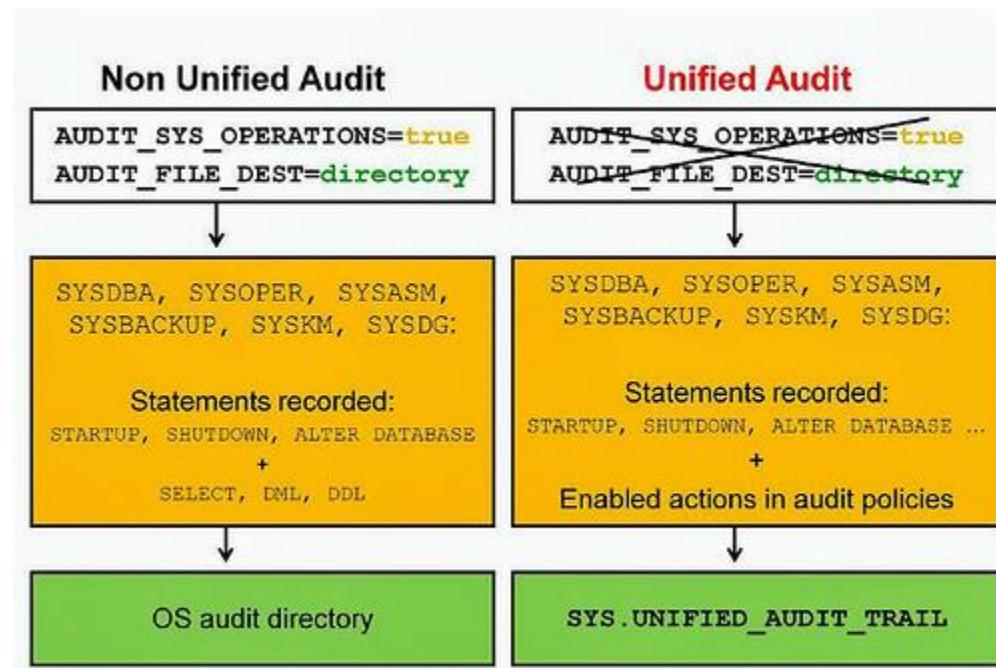
## Audit Roles

There are two new audit roles in 12c.

**AUDIT\_ADMIN** can create the new audit policies, able to define fine grained audit policies, view and manage all the audit data and perform other administrative actions related to auditing. The role should be granted to trusted persons only.

**AUDIT\_VIEWER** role is suited for external auditors. The role is only allowed to view audit data.

**NOTE:** In the previous versions of the database it was possible for every user to enable/disable auditing on his own objects. This is not possible now.



### Step 1: Creating the Audit Policy

Create audit policies based on **systemwide** audit options.

- System privilege

```
SQL> CREATE AUDIT POLICY audit_syspriv_pol1
 2 PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY;
```

- Actions

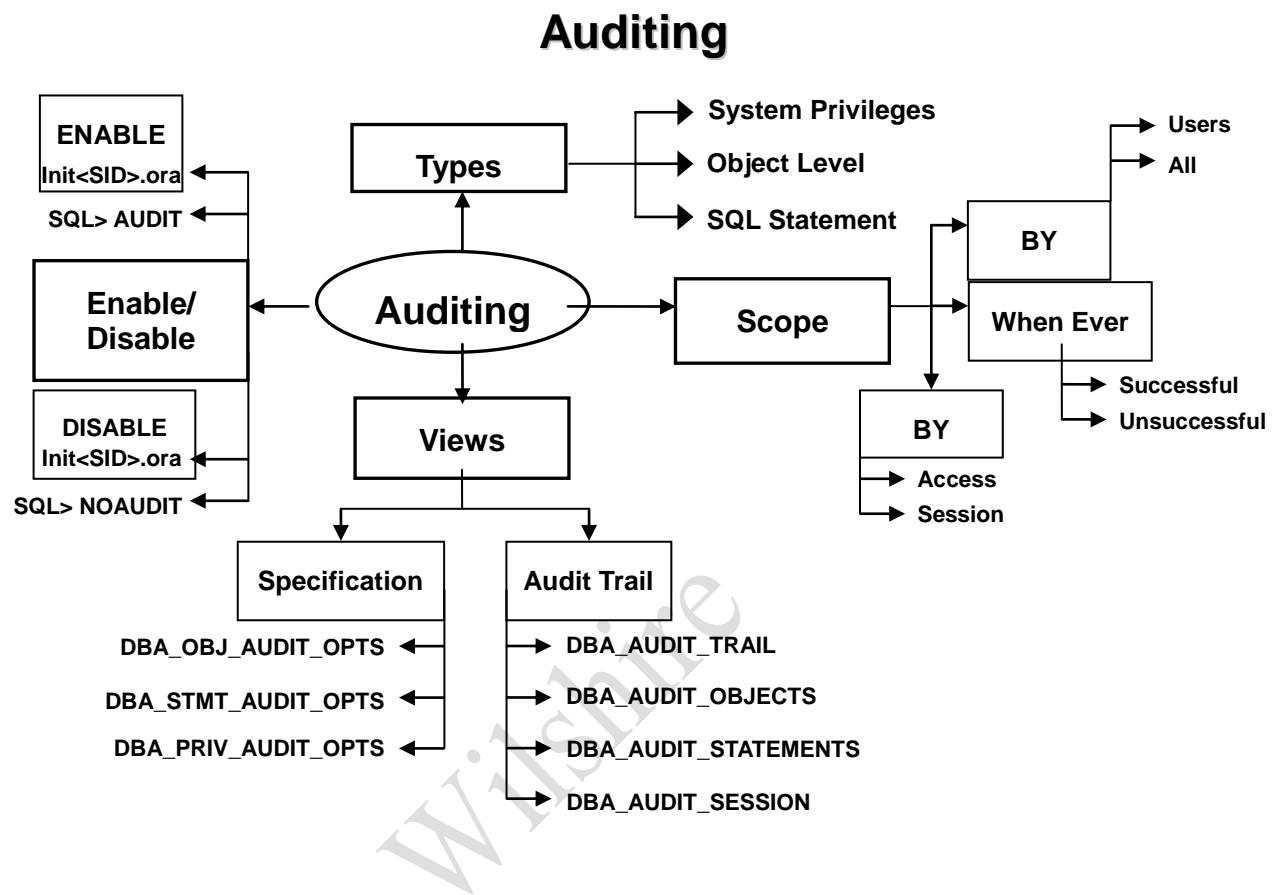
```
SQL> CREATE AUDIT POLICY audit_actions_pol2
 2 ACTIONS AUDIT, ALTER TRIGGER;
```

- Role

```
SQL> CREATE AUDIT POLICY audit_role_pol3
 2 ROLES mgr_role;
```

- System privilege, actions, and roles

```
SQL> CREATE AUDIT POLICY audit_mixed_pol4
 2 PRIVILEGES DROP ANY TABLE
 3 ACTIONS CREATE TABLE, DROP TABLE, TRUNCATE TABLE
 4 ROLES emp_role;
```



## 64. Log Miner

LogMiner tool suite lets a DBA scan through online redo logs or archived redo logs to obtain actual DML SQL statements that have been issued to the database server to create the redo change entries. LogMiner can also return the SQL statements needed to undo the DML that has been issued.

However, LogMiner did have a few drawbacks: Even with the Oracle Enterprise Manager User Interface, it could take some power struggle to get LogMiner to return the information needed for recovery. In addition, it did not support retrieval of data from columns with Large Object (LOB) datatypes. The good news is that Oracle 10g has enhanced the LogMiner tool suite to overcome many of these issues:

**Automated Determination of Needed Log Files:** Prior to Oracle 10g, one of the more tedious tasks before initiating a LogMiner operation was to determine which archived redo logs were appropriate targets for mining.

This is handled by querying the V\$ARCHIVED\_LOG view to determine which archived redo log files might fulfill LogMiner query based on their start and end time periods, and then used the DBMS\_LOGMNR.ADD\_LOGFILE procedure to query against just those log files. Oracle 10g has greatly simplified this by scanning the control file of the target database to determine which redo logs will fulfill the requested timeframe or SCN range.

**Example:** Letting the database's control file establish which redo logs LogMiner needs to complete its work

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI';
SQL> SPOOL C:\Listing_45.log
```

Start LogMiner, running from the database's online data dictionary and preparing for several mining attempts

```
BEGIN
 DBMS_LOGMNR.START_LOGMNR(STARTTIME => '02/20/2005 06:00',
 ENDTIME => '02/20/2005 12:00',
 OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
 DBMS_LOGMNR.CONTINUOUS_MINE
);
END;
/

-- Find the desired data
SQL> SELECT seg_owner,seg_name,operation,sql_redo FROM v$logmnr_contents
 WHERE operation = 'INSERT' AND seg_owner = 'HR';
```

Reissue the START\_LOGMNR directive for a new starting and ending period, but this time based on specified starting and ending SCNs

```
BEGIN
 DBMS_LOGMNR.START_LOGMNR(STARTSCN => 2265600,ENDSCN => 2261629,
 OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
 DBMS_LOGMNR.CONTINUOUS_MINE
);
END;
/

-- Find the desired data
SQL> SELECT seg_owner,seg_name,operation,sql_redo FROM v$logmnr_contents
 WHERE operation = 'INSERT' AND seg_owner = 'HR';

-- End the LogMiner session
SQL> EXEC DBMS_LOGMNR.END_LOGMNR;
SQL> SPOOL OFF
```

The above shown example of the new CONTINUOUS\_MINE directive of procedure [www.wilshiresoft.com](http://www.wilshiresoft.com)

DBMS\_LOGMNR.START that directs Oracle to determine what log files are needed based on the ranges specified. It also illustrates that the DBMS\_LOGMNR.START procedure can be executed multiple times within a LogMiner session to effectively limit the range of log files required for the mining request.

In addition to the existing directive, NO\_SQL\_DELIMITER that removes semicolons from the final display, Oracle 10g also adds a new directive, PRINT\_PRETTY\_SQL that formats the SQL into a more legible format. Another new directive, NO\_ROWID\_IN\_STMT, will omit the ROWID clause from the reconstructed SQL when the DBA intends to reissue the generated SQL - especially when it is going to be executed against a different database with different ROWIDs. See the below examples for these directives.

Example: Making LogMiner SQL output "prettier"

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'MM/DD/YYYY HH24:MI';
SQL> SPOOL C:\Listing_45.log
BEGIN
 Start LogMiner, running from the database's online data dictionary and
 preparing for several mining attempts
 DBMS_LOGMNR.START_LOGMNR(STARTTIME => '02/20/2005 06:05',
 ENDTIME => '02/20/2005 06:15',
 OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG +
 DBMS_LOGMNR.CONTINUOUS_MINE + DBMS_LOGMNR.NO_ROWID_IN_STMT +
 DBMS_LOGMNR.PRINT_PRETTY_SQL
);
END;
/
-- Find the desired data
SELECT sql_redo FROM v$logmnr_contents WHERE seg_owner = 'HR';

-- End the LogMiner session
BEGIN
 DBMS_LOGMNR.END_LOGMNR;
END;
/
SPOOL OFF
```

Expanded Support for Additional Datatypes. LogMiner now supports retrieval of SQL Redo and Undo information for Large Objects (LOBs) including multibyte CLOBs and NCLOBs. Data stored in Index-Organized Tables (IOTs) is now also retrievable, so long as the IOT does not contain a LOB.

Storing the LogMiner Data Dictionary in Redo Logs. LogMiner needs to have access to the database's data dictionary so that it can make sense of the redo entries stored in the log files. Prior to Oracle 10g, only two options were available. The database's data dictionary can be used as long as the database instance is accessible. Another option is to store the LogMiner data dictionary in a flat file created by the DBMS\_LOGMNR.D.BUILD procedure. This offers the advantage of being able to transport the data dictionary flat file and copies of the database's log files to another, possibly more powerful or more available server for LogMiner analysis. However, this option does take some extra time and consumes a lot of resources while the flat file is created.

Oracle 10g now offers a melding of these two options: The capability to store the LogMiner data dictionary within the active database's redo log files. The advantage to this approach is that the data dictionary listing is guaranteed to be consistent, and it is faster than creating the flat file version of the data dictionary. The resulting log files can then be specified as the source of the LogMiner data dictionary during mining operations. See the below example to implement this option.

Example: Creating a LogMiner data dictionary and storing it within the online redo log files

```
BEGIN
 DBMS_LOGMNR_D.BUILD(OPTIONS => DBMS_LOGMNR_D.STORE_IN_REDO_LOGS);
END;
/
```

#### **DDL\_DICT\_TRACKING:**

This new feature of Oracle9i allows logmnrd dictionary to use either a flat file or the redo logs, to ensure that its internal dictionary is updated if a DDL event is found in the redo log files. This ensures that SQL\_REDO and SQL\_UNDO information is correct for objects that are modified in the redo log files after the LogMiner internal dictionary was built. By default this option is disabled.

Wilshire

## 65. Automatic Storage Management (ASM)

Setting up storage takes a significant amount of time during most database installations. Zeroing on a specific disk configuration from among the multiple possibilities requires careful planning and analysis, and, most important, intimate knowledge of storage technology, volume managers, and filesystems. The design tasks at this stage can be loosely described as follows (note that this list is merely representative; tasks will vary by configuration):

- Confirm that storage is recognized at the OS level and determine the level of redundancy protection that might already be provided (hardware RAID).
- Assemble and build logical volume groups and determine if striping or mirroring is also necessary.
- Build a file system on the logical volumes created by the logical volume manager.
- Set the ownership and privileges so that the Oracle process can open, read, and write to the devices.
- Create a database on that filesystem while taking care to create special files such as redo logs, temporary tablespaces, and undo tablespaces in non-RAID locations, if possible.

Oracle Database 10g offers a new and exciting feature, Automatic Storage Management (ASM), lets DBAs execute many of the above tasks completely within the Oracle framework. Using ASM you can transform a bunch of disks to a highly scalable (and the stress is on the word scalable) and performant filesystem/volume manager using nothing more than what comes with Oracle Database 10g software at no extra cost.

### 65.1 What is ASM?

Let's say that you have 10 disks to be used in the database. With ASM, you don't have to create anything on the OS side; the feature will group a set of physical disks to a logical entity known as a diskgroup.

A diskgroup is analogous to a striped (and optionally mirrored) filesystem, with important differences: it's not a general-purpose filesystem for storing user files and it's not buffered. Because of the latter, a diskgroup offers the advantage of direct access to this space as a raw device yet provides the convenience and flexibility of a filesystem.

Logical volume managers typically use a function, such as hashing to map the logical address of the blocks to the physical blocks. This computation uses CPU cycles. Furthermore, when a new disk (or RAID-5 set of disks) is added, this typical striping function requires each bit of the entire data set to be relocated.

In contrast, ASM uses a special Oracle Instance to address the mapping of the file extents to the physical disk blocks. This design, in addition to being fast in locating the file extents, helps while adding or removing disks because the locations of file extents need not be coordinated. This special ASM instance is similar to other filesystems in that it must be running for ASM to work and can't be modified by the user.

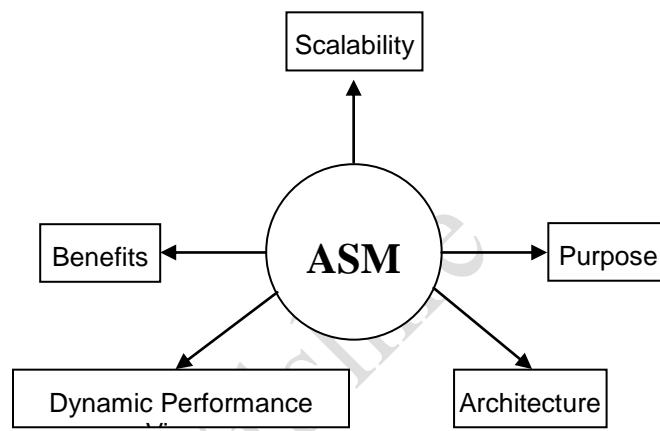
One ASM instance can service a number of Oracle databases instances on the same server. This special instance is just that: an instance, not a database where users can create objects. All the metadata about the disks are stored in the diskgroups themselves, making them as self-describing as possible. The Advantages are :

- Disk Addition-Adding a disk becomes very easy. No downtime is required and file extents are redistributed automatically.
- I/O Distribution-I/O is spread over all the available disks automatically, without manual intervention, reducing chances of a hot spot.
- Stripe Width-Striping can be fine grained as in Redo Log Files (128K for faster transfer rate) and coarse for datafiles (1MB for transfer of a large number of blocks at one time).

- Buffering-The ASM filesystem is not buffered, making it direct I/O capable by design.
- Kernelized Asynch I/O-There is no special setup necessary to enable kernelized asynchronous I/O, without using raw or third party filesystems such as Veritas Quick I/O.
- Mirroring-Software mirroring can be set up easily, if hardware mirroring is not available.

Even archived log destinations can also be set to a diskgroup. Pretty much everything related to Oracle Database can be created in an ASM-based diskgroup. For example, backup is another great use of ASM. You can set up a bunch of inexpensive disks to create the recovery area of a database, which can be used by RMAN to create backup datafiles and archived log files. However ASM supports files created by and read by the Oracle Database only; it is not a replacement for a general-purpose filesystem and cannot store binaries or flat files.

The introduction of ASM provides a significant value in making it much easier to manage files in an Oracle database. Using this bundled feature, you can easily create a very scalable and performant storage solution from a set of disks. Any dynamic database environment requires the addition, shifting, and removal of disks, and ASM provides the necessary toolset to free the DBA from those mundane tasks.



### Purpose

1. Stripping and Mirroring.
2. Dynamic Storage Configuration.
3. No need for 3rd Party Volume Manager Software.
4. Better Performance thru Stripping
5. Fault Tolerance (Protection using Mirroring).
6. Scalability

### Architecture

1. ASM Instance
2. Disk Groups
3. ASM Files

### Benefits

1. I/O is spread evenly across all available disks
2. Automatic Online redistribution
3. Maintain redundant copies of data to provide Fault Tolerance
4. ASM Software is available free of cost, It is part of Oracle 10G
5. Mirroring is done at extent level (Not at disk Level).

### Scalability

ASM imposes the following limits

1. 57. disk groups in a storage system.
2. 10,000 ASM disks in a storage system.
3. 4 Petabyte Max storage for each ASM disk.
4. 1 million files for each disk groups
5. Max file sizes are shown in the below table.

6. Prevents disk fragmentation (No need for db.Reorg)
7. Can support multiple nodes in RAC Environment.
8. Can add new drives without shutting down ASM Instance.

| Disk Group Type     | Max. File Size |
|---------------------|----------------|
| External Redundancy | 35 TB          |
| Normal Redundancy   | 5.8 TB         |
| High Redundancy     | 3.9 TB         |

### 65.1.1 Dynamic Performance Views

- v\$asm\_diskgroup
- v\$asm\_client
- v\$asm\_disk
- v\$asm\_template
- v\$asm\_alias
- v\$asm\_operation

### 65.2 Important Points

1. ASM is built on top of OMF
2. Use RMAN to backup ASM Files
3. One database can consist of ASM/OMF/OS Files together
4. ASM automatically balances I/O load
5. ASM provides Fault Tolerance
6. We access ASM files through ASM-DG
7. ASM - DG is a collection of Disks
8. Better performance – Disk Striping
9. Better redundancy – Disk mirroring
10. ASM mirrors extents but not entire disk
11. ASM redundancy Levels are – External, Normal, High
12. Number of Failure Groups determine the degree of redundancy
13. ASM automatically deletes any unnecessary files
14. When creating Tablespace mention DG (no need to specify datafile name)
15. Can see FQF name (Fully qualified File) for v\$logfile, v\$datafile views
16. ASM uses FQF to refer to a file
17. ASM derives numeric names for a FQF to refer an existing file
18. We can use alias ASM filenames to create/refer to ASM files
19. We can use Incomplete file names only to create but not to refer
20. We must create DG-dir to use alias filenames
21. ASM file-templates specifies File-Attributes
22. Can't change file Attributes after file got created
23. When a new DG is created, set of definition templates are copied.
24. Each ASM-Disk has a disk name by
25. Assigned by DBA
26. Assigned by ASM when added to DG
27. Coarse striping spreads files in units of 1MB each across all disks – suitable for OLTP-DBs
28. Fine striping spreads files in units of 128KB each across all disks – suitable for OLAP-DBs
29. Definition-templates (cf/dbf/log/arch/tmpfile/backupset/fb)
30. Use RMAN to migrate Non-ASM-DB to ASM-DB

**Only when referring to an existing ASM file**

- Fully qualify names → +DG2/db/FileType/tag.File.incornation  
Example: +DG1/ACCT/datafile/user\_data.315.1
- Numeric Name → +DG2.315.1

**When referring to an existing file [OR] for creating ASM File**

- Alias Name → Alter DG ADD Alias  
`SQL> ALTER DG DG2 ADD Dir '+DG2/Acct';  
SQL> ALTER DG DG2 ADD Alias '+DG2/Acct/user.dbf'  
FOR 'DG2/Acct/datafile/users3.315.1';`

**Only for creating ASM File**

- Alias with Template `SQL> CREATE TABLESPACE User4 DATAFILE '+DG2/user_spare(datafile)';`
- Incomplete Names  
For creating Single / Multiple Files.  
Only Disk Group is specified

`SQL> CREATE TABLESPACE User5 DATAFILE '+DG2';`

- Incomplete with Template (Template Determines character of File)  
`SQL> CREATE TABLESPACE User6 DATAFILE '+DG(tempfile)';`

## 66. ASM Enhancements in 11G R1

Automatic Storage Management (ASM) in Oracle Database 11g is an evolution in file system and volume management functionality for Oracle database files. ASM further enhances automation and simplicity in storage management that is critical to the success of the Oracle grid architecture. ASM also improves file system scalability and performance, manageability, and database availability for single-instance Oracle databases as well as for Oracle Real Application Clusters (Oracle RAC) environments.

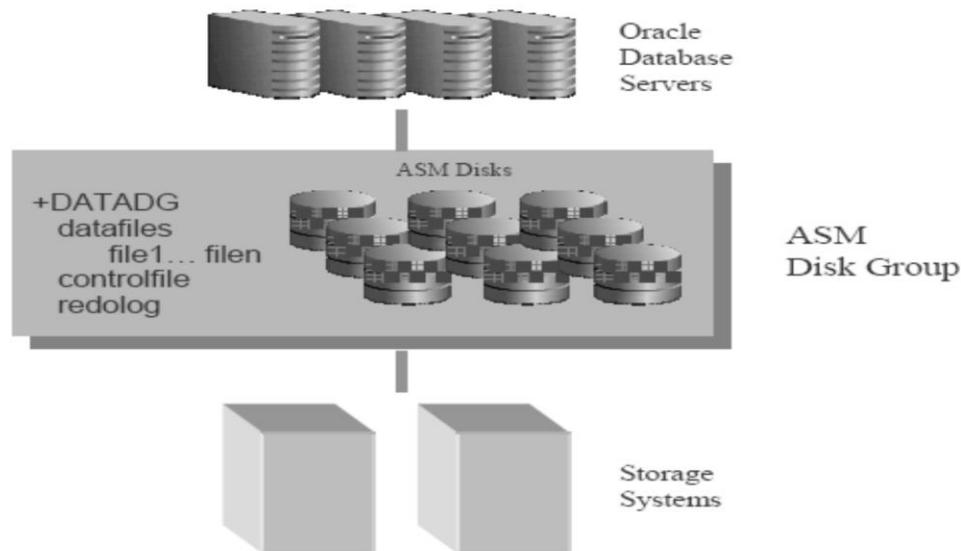
This document is targeted at a technical audience mainly comprising:

- Database, system and storage administrators
- Architects
- Consultants
- System engineers
- Technical managers

### 66.1 Automatic Storage Management Overview

ASM is a feature of Oracle Database 11g that provides an integrated cluster file system and volume management capabilities at no additional cost. ASM lowers the Oracle database storage total cost of ownership and increases storage utilization while improving performance and availability over traditional file system and volume management solutions. With ASM, a fraction of the time is needed to manage your database storage environment and datafiles.

ASM is easier to manage than conventional file systems, and it has the performance of raw volumes and is tightly integrated with the Oracle Database forming the foundation of a storage grid for Oracle. Additionally, ASM eliminates the need for 3rd-party volume managers and file systems for managing the Oracle database files.



ASM brings significant key values to Oracle Database platforms. ASM improves manageability by simplifying storage provisioning, storage array migration, and storage consolidation. ASM provides flexible easy to manage interfaces including the SQL\*Plus, Oracle Enterprise Manager GUIs and a UNIX-like command line interfaces. ASM provides sustained best in class performance because of its innovative rebalancing feature that distributes data evenly across all storage resources,

(VLDB) efficiently without compromising functionality or performance.

ASM is built to maximize database availability. ASM provides self-healing automatic mirror reconstruction and resynchronization, rolling upgrades and patching. ASM also supports dynamic and on-line storage reconfigurations in both single instance and Oracle RAC database configurations.

ASM customers realize significant cost savings and achieve lower total cost of ownership because of features such as just-in-time provisioning, and clustered pool of storage that is ideal for database consolidation. ASM provides all of this without an additional license or licensing fees.

ASM is the preferred file system and volume manager for Oracle database files for the following reasons:

- Simplifies and automates storage management
- Increases storage utilization, uptime, and agility
- Delivers predictable performance and availability service level agreements

## 66.2 Automatic Storage Management New Features

ASM in Oracle Database 11g provides major functionality enhancements in the following areas:

- Scalability and performance
- Improved scalability and performance for VLDB
- Fast mirror resynchronization
- Preferred mirror read in a clusters
- Support for larger allocation units (AU)
- Fast rebalance
- Rolling upgrade and patching
- Manageability enhancements
- Table level migration wizard in EM
- Disk group compatibility attributes
- New ASMCMD commands
- New SYSASM privilege separate from the SYSDBA privilege
- More flexible FORCE options to MOUNT and DROP disks group

## 66.3 Improved Scalability and Performance

### Fast Mirror Resync

ASM Fast Mirror Resync significantly reduces the time required to recover from the ability to resync mirrors transient disk failures in normal and high redundancy disk groups. Proactive maintenance of ASM disks and/or transient ASM disk failures can both highly benefit from this feature. When a disk goes offline following a transient failure, ASM tracks the extents that are modified during the outage.

When the transient failure is repaired, ASM can quickly resynchronize only the ASM mirror extents that have been updated during the outage. A transient failure assumes that the contents of the mirrored disks are intact and the reasons for the disk unavailability are because of failures such as a loose cable, a power failure or a recycle, a host bus adapter, or a disk controller.

The new DISK\_REPAIR\_TIME attribute defines a time window in which the transient failure can be repaired and the mirror disk can be brought back on-line. ASM only resynchronizes the changed blocks (extents), which results in much faster recovery time. This can mean a difference between minutes to hours or days depending on the amount of data on the disks being brought on-line.

The DISK\_REPAIR\_TIME attribute is set to 3.6 hours by default. You can use

```
ALTER DISKGROUP <disk group name> SET ATTRIBUTE "DISK_REPAIR_TIME"=" 24H"
```

to, as in this example, extend this value to a longer time period. After you repair the disk, run the SQL statement ALTER DISKGROUP DISK ONLINE. This statement brings a repaired disk group back online and starts the resynchronization process. The V\$ASM\_ATTRIBUTE view shows the current disk repair time attribute value.

You can see the time remaining value in the REPAIR\_TIME column of the V\$ASM\_DISK or V\$ASMDISK\_STAT views. You can also manually OFFLINE a disk or failure group, or force drop offline disks that cannot be repaired. In this case, a full resynchronization is necessary after replacing the failed disks and adding them to the disk group.

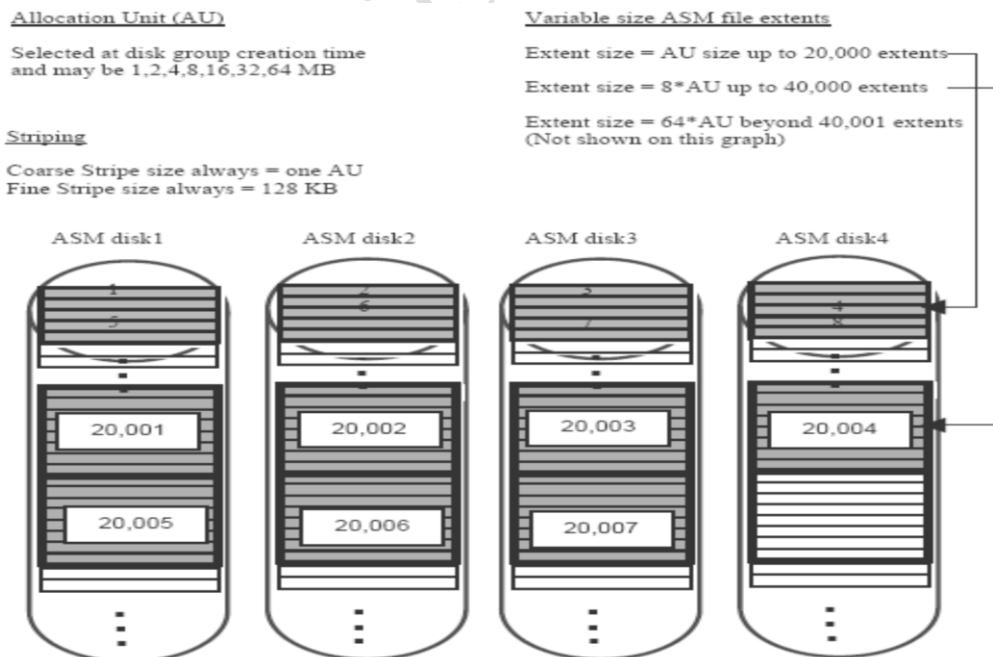
This feature applies to all cases where ASM mirroring is used and further improves database availability. Applications such as extended clusters, where you create two failure groups (one on each site), derive additional benefits because a network or connectivity outage can be recovered very quickly without requiring a full mirror resync.

Proactive preventative maintenance is also now feasible because you can offline failure groups, perform maintenance, and then online the failure group to update your disk contents in relatively much shorter time without the need to take the database down.

## 66.4 VLDB Support

### Variable size extents

The variable size extent feature in Oracle Database 11g enables support for much larger ASM files, reduces SGA memory requirements for very large databases, and improves performance for file create and open functions.



An ASM file can begin with 1 MB extents (assuming 1MB AU) and as the file size increases, the extent size also increases to 8MB and 55MB at a predefined number of extents. Therefore, the size of the extent map defining a file can be smaller by a factor of 8 and 55 depending on the size of the file.

The initial extent size is equal to the allocation unit size and it increases by the 8 and 55 factor at predefined thresholds. This feature is completely automatic for newly created files (or the extended part of files) once compatible.asm and compatible.rdbms has been advance to 15.1. ASM VLDB support is extended to:

- 140 Peta Byte (PB) in External Redundancy (no ASM mirroring)
- 42PB in Normal Redundancy (2-way ASM mirroring)
- 15PB in High Redundancy (3-way ASM mirroring)
- 40 ExaBytes of storage
- 4PB for each ASM disk
- 1 million files per disk group
- 10,000 ASM disks
- 57. disk groups per ASM instance

Please note that Oracle database limits file sizes to 128TB with big file tablespaces and 32k block size which is considerably lower than ASM file size limits.

## 66.5 Multiple Allocation Unit size

You can now set the ASM Allocation Unit (AU) size to be 1/2/4/8/16/32/55MB at disk group creation time. An allocation unit is the minimum segment of a disk that can be assigned to a file. Each extent is an integral number of allocation units. Each file contains an integral number of extents. Our research has shown that significant performance can be gained for some workloads and storage system types deployed by using larger AU sizes.

The default AU size is set to 1MB for ASM to match the MAXIO default for Oracle Database IO buffer size. The larger AU sizes are likely to benefit large sequential reads and writes. It may be beneficial to increase the MAXIO buffer size to 4MB and create the ASM disk group with a 4MB allocation unit for optimal performance for applications performing large sequential IO.

Please note that the FINE-grained striping remains 128KB as before, but the COARSE-grained stripe is equal to the AU size selected.

## 66.6 Faster Rebalance with Restricted Mount Option

A disk group mounted in RESTRICTED mode is mounted exclusively by only one node. Clients of ASM on that node cannot access that disk group while it is mounted in RESTRICTED mode. The RESTRICTED mode enables you to perform all maintenance tasks on a disk group in the ASM instance without any external interaction.

Rebalance operations performed while the diskgroup is in RESTRICTED mode eliminate the lock/unlock extent maps messaging between ASM instances in an Oracle RAC environment, thus improving overall rebalance throughput.

At the end of the maintenance cycle you have to explicitly dismount the disk group and remount it in normal mode.

The ALTER DISKGROUP diskgroupname MOUNT command is extended to allow for ASM to mount the diskgroup in restricted mode. Here is an example of the sequence of commands:

```
ALTER DISKGROUP data DISMOUNT;
```

ALTER DISKGROUP data MOUNT RESTRICT; Maintenance task: You can add/drop online/offline disks, drop files, create directories, etc. ALTER DISKGROUP data DISMOUNT;

```
ALTER DISKGROUP data MOUNT;
```

When you use the RESTRICTED option to startup an ASM instance, all of the disk groups defined in the ASM\_DISKGROUPS parameter are mounted in RESTRICTED mode.

## Preferred Mirror Read

When ASM is managing redundancy, you can configure an ASM instance on a node to read from a preferred mirror copy (i.e. a preferred read failure group). The default behavior is to always read from the primary copy.

This feature is beneficial when you have an extended Oracle RAC cluster, where the nodes and the failure groups are separated by a long distance to enable disaster recovery. In this case, the Oracle RAC nodes on each site can be configured to read from their local storage mirror copies instead of going through a network with potentially high latencies.

Set the ASM\_PREFERRED\_READ\_FAILURE\_GROUPS initialization parameter to specify a list of failure group names as preferred read disks. This parameter is a multi-valued parameter and should contain a string with a list of <disk group name>.<failure group name> separated by a comma.

The new column PREFERRED\_READ has been added to the V\$ASM\_DISK and V\$ASM\_DISK\_IOSTAT views. If the disk group that the disk is in belongs to a preferred read failure group, then the value of this column is Y.

To set up preferred mirror read between sites A and B, the failure group name SITEA or SITEB should be preceded by the disk group name:

```
ASM_PREFERRED_READ_FAILURE_GROUP=DATA.SITEA or
ASM_PREFERRED_READ_FAILURE_GROUP=DATA.SITEB
```

## 66.7 ASM Rolling Upgrades and Patching

In an Oracle RAC environment, the ASM rolling upgrade feature enables you to independently upgrade or patch ASM nodes in a cluster, thus providing greater a huge win and significantly flexibility and uptime. This feature is available starting from Oracle Database 11g forward.

Before patching or upgrading ASM, you must place the ASM cluster in a "rolling migration" mode. This enables an ASM instance to operate in a multi-version environment. Rolling migration mode is expected to be a short-term state.

You should have the appropriate software installed on all nodes of the cluster prior to entering rolling migration mode. Oracle Clusterware must be upgraded to the appropriate version on all nodes prior to upgrading ASM. All normal database file operations are allowed, but most disk group configuration changes are prevented while in rolling migration mode.

```
ALTER SYSTEM START ROLLING MIGRATION TO number;
```

Now you can shut down each ASM instance individually and perform a software upgrade on that instance. The upgraded ASM instance can then rejoin the cluster after the upgrade because it is in rolling migration mode. After all the nodes are successfully upgraded, you can end the rolling migration mode to return to full functionality normal operation.

```
ALTER SYSTEM STOP ROLLING MIGRATION;
```

You can also use the same procedure to roll back (downgrade) nodes if you encounter problems during the migration. You cannot enter rolling migration mode if there are rebalancing operations occurring anywhere in the cluster.

You must wait until the rebalance completes before initiating a rolling migration. New instances that join the cluster during a migration, switch to a rolling migration state immediately upon startup. You can use the following SQL function to identify the state of a clustered ASM environment:

```
select SYS_CONTEXT('sys_cluster_properties', 'cluster_state') from dual;
```

Show output from this command

## 66.8 Manageability Enhancements

### 66.8.1 Disk Group Compatibility Attributes

Two new disk group compatibility attributes are introduced in Oracle Database 11g ASM. The new compatible.asm and compatible.rdbms disk group compatibility attributes determine the minimum version of ASM and database instances that can connect to an ASM disk group.

If you upgrade your ASM and database version from 8.1 to 15.1, you have the option to keep the ASM disk compatibility attribute at 8.1. In this case, your ASM and database 15.1 will function but not be able to take advantage of all the 15.1 new features yet.

This is valuable since this allows you to upgrade your software and make sure you are happy with the release before committing. You can advance the disk group compatibility attribute from 8.1 to 15.1 to enable all the features in 15.1. You can revert back to the previous release (8.1) if you have not advanced the disk group compatibility attributes. Care must be taken to advance the attributes since this action is irreversible.

You can set the disk group compatibility by using either the CREATE DISKGROUP or ALTER DISKGROUP commands.

CREATE DISKGROUP DATA NORMAL REDUNDANCY

```
DISK '/dev/raw/raw1', '/dev/raw/raw2'
```

```
ATTRIBUTE 'compatible.asm'=15.1' compatible.rdbms'=15.1'
```

The following features will be enabled when you advance asm and rdbms disk group compatibility attributes:

Advancing compatible.asm or compatible.rdbms attributes from 8.1 to 15.1 will enable the following key features:

- Preferred mirror read
- Variable size extents
- Fast mirror resync

## 66.9 New ASMCMD commands

We have introduced three new functionalities to the ASMCMD utility to improve node recovery after a crash, repair bad blocks on a disk, copy files and simplify the listing of ASM disks in a disk group.

ASMCMD cp

The ASMCMD cp option allows you to copy files between ASM disk groups and OS file systems and between two ASM servers. The following file copy permutations are supported:

```
ASM disk group -- OS file system
OS file system -- ASM disk group
ASM disk group -- Another ASM disk group on the same server
ASM disk group -- ASM disk group on a remote server
```

This example shows an ASM file being copied to a file system directory /backups:

```
asmcmd cp +DG1/VDB.CTF1 /backups
ASMCMD> cp +DG1/vdb.ctf1 /backups/vdb.ctf1
copying file(s) ...
copying file(s) ...
copying file(s) ...
file, /backups/vdb.ctf1, copy committed.
```

ASMCMD disk group metadata backup and restore

ASMCMD is extended to include ASM disk group metadata backup and restore functionality. This provides the ability to recreate a pre-existing ASM disk group with the same disk paths, disk names, failure groups, attributes, templates, and alias directory structure.

Currently if an ASM disk group is lost, then it is possible to restore the lost files using RMAN but you have to manually recreate the ASM disk group and any required user directories/templates. There is no need to backup and restore ASM metadata itself.

ASM metadata backup and restore works in two modes. In backup mode it gathers information about existing disks and failure group configurations, attributes, templates, and alias directory structures and dumps it to a backup file.. In restore mode, it reads the previously generated file to reconstruct the disk group and its metadata. You have the possibility to control the behavior in restore mode to do a full, nodg, or newdg restore. The "full" mode restores the disk group exactly as it was at the time of backup.

The "nodg" mode restores the attributes, templates, and alias directory structure specified in the backup file to an existing disk group. The "newdg" mode allows the user to override the disk group name, disk, and failure group specifications as part of a disk group creation, but retains the attribute, template, and alias directory structure from the backup. This example describes how to backup ASM metadata using the md\_backup command, and how to restore them using the md\_restore command.

```
ASMCMD> md_backup <file> backup_file -g data
```

Disk group to be backed up: DATA#

```
ASMCMD>
```

The above statement specifies the <file> option and the <group> option of the command. This is to define the name of the generated file containing the backup information as well as the disk group that needs to be backed up.

Before you can restore the database files it contained, you have to restore the disk group itself. Let's assume that a storage array failure lost all of the disks in disk group DATA. In this case, you initiate the disk group recreation as well as restoring its metadata using the md\_restore command. Here, you specify the name of the backup file generated at step one, as well as the name of the disk group you want to restore, and also the type of restore you want to do. Here a full restore of the disk group is done because it no longer exists.

```
ASMCMD> md_restore -b jfv_backup_file -t full -g data
Disk group to be restored: DATA#
ASMCMDAMBR-09358, Option -t newdg specified without any override options.

Current Diskgroup being restored: DATA
Diskgroup DATA created!
User Alias directory +DATA/jfv
created!
ASMCMD>
Once the disk group is recreated, you can restore its database files
using RMAN for example.
ASMCMD lsdsk
```

The lsdsk command lists ASM disk information. This command can run in two modes: connected and non-connected. In connected mode, ASMCMD uses the V\$ and GV\$ views to retrieve disk information. In non-connected mode, ASMCMD scans disk headers to retrieve disk information, using an ASM disk string to restrict the discovery set. The connected mode is always attempted, first. This is very useful for system or storage administrators who want a consolidated list of disks with ownerships.

```
ASMCMD> lsdsk &ekspt "/dev/sdb6
ASMCMD remap
```

When a normal read from an ASM disk group fails with an I/O error ASM satisfies the write from a mirrored extent in Normal and High redundancy mode. ASM attempts to remap the extent where the read failed by relocating it to another allocation unit on the same disk with the contents from the good mirror.

If the remap succeeds on the same disk, the old allocation unit is marked as unusable. This process happens automatically only on blocks that are read. It is possible that some blocks and extents on an ASM disk group are seldom read; e.g. the secondary extents. The ASMCMD remap invokes bad block remapping for the specified blocks. One can use the ASMCMD remap command if the storage array returns an error on a physical block, then the ASMCMD repair can initiate a read on that block to trigger the repair.

```
ASMCMD> asmcmd remap DATA DATA_00016000-7600
```

## 66.10 SYSASM role

This feature introduces a new SYSASM role that is specifically intended for performing ASM administration tasks. Using the SYSASM role instead of the SYSDBA role separates ASM administration from database administration. In Oracle Database 11g, the OS group for SYSASM and SYSDBA is the same, and the default installation group for SYSASM is dba but they can be different. In future releases, separate OS groups will have to be created to separate the SYSDBA and SYSASM privileges.

SYSDBA will only be able to query views and manipulate ASM files. SYSASM will be required to manage disk group configurations. In this release, you can connect to an ASM instance as a member of the OSASM, which defaults to the dba group.

```
SQL> CONNECT / AS SYSASM
```

You also have the possibility to use the combination of CREATE USER and GRANT SYSASM SQL statements from an ASM instance to create a new SYSASM or SYSDBA user. This is possible as long as the name of the user is an existing OS user name. These commands update the password file for the local ASM instance, and do not need the instance to be up and running. Similarly, you can revoke the SYSASM role from a user using the REVOKE command, and you can drop a user from the password file using the DROP USER command.

```
SQL> CREATE USER ossysasmusername IDENTIFIED by passwd;SQL> GRANT SYSASM
TO ossysasmusername;
SQL> CONNECT ossysasmusername / passwd AS SYSASM;
SQL> DROP USER ossysasmusername;
```

Note: With Oracle Database 11g Release 1, if you log in to an ASM instance as SYSDBA, warnings are written in the corresponding alert.log file when you issue commands that alter disk group configuration.

## 66.11 Mount/Drop FORCE disk group

With Oracle Database 11g, ASM will fail to mount a disk group if there are any missing disks or failure groups during mount, unless the new FORCE option is used when mounting the incomplete disk group. This allows you to correct configuration errors like ASM\_DISKSTRING set incorrectly, or connectivity issues before trying the mount again without incurring unnecessary rebalance operations.

```
ALTER DISKGROUP data MOUNT FORCE | [NOFORCE] ;
```

Disk groups mounted with the FORCE option will have one or more disks offline if they were not available at the time of the mount. You must take corrective actions before DISK\_REPAIR\_TIME expires to restore those devices. Failing to online those devices would result in the disks being expelled from the disk group and the need for a rebalance operation to restore redundancy for all the files in the disk group.

Mount with NOFORCE is the default mode. In the NOFORCE mode, all of the disks that belong to a disk group must be accessible for the mount to succeed. Drop disk group FORCE marks the headers of disks belonging to a disk group that cannot be mounted by the ASM instance as FORMER. However, the ASM instance first determines whether the disk group is being used by any other ASM instance using the same storage subsystem.

If it is being used, and if the disk group is in the same cluster, or on the same node, then the statement fails. If the disk group is in a different cluster, then the system further checks to determine whether the disk group is mounted by any instance in the other cluster. If it is mounted elsewhere, then the statement fails. Please exercise caution when using the force option. DROP DISKGROUP FORCE is much safer than using dd to overwrite disk headers.

```
DROP DISKGROUP data FORCE INCLUDING CONTENTS;
```

## 66. ASM Enhancements in 11G R2

This section describes the Oracle Automatic Storage Management (Oracle ASM) 11g release 2 (11.2.0.1) new features:

### 66.1 Disk Group Rename

The renamedg tool enables you to change the name of a cloned disk group. The disk group must be dismounted on all nodes in the cluster before running renamedg on the disk group.

renamedg renames a disk group using a two-step process:

1. Phase one : This phase generates a configuration file to be used in phase two.
2. Phase two : This phase uses the configuration file to perform the renaming of the disk group.

The syntax is:

```
renamedg {-help | help=true}
```

```
renamedg
```

```
[phase={ one|two |both }] dgname=diskgroup
newdname=newdiskgroup [config=configfile]
[asm_diskstring=discoverystring, discoverystring ...]
[clean={true|false}] [check={true|false}]
[confirm={true|false}] [verbose={ true|false}]
[keep_voting_files={true|false}]
```

- phase={one|two|both}
- Specifies the phase to be run. Allowed values are `one`, `two`, or `both`. This argument is optional. The default is `both`. Typically you would run both phases. If a problem occurs during the second phase, then you can re-run phase `two` using the generated configuration file.
- dgname=diskgroup
- Specifies the name of the disk group that to be renamed.
- newdname=newdiskgroup
- Specifies the new name for the disk group.
- config=configfile
- Specifies the path to the configuration file to be generated during phase one or specifies the path to the configuration file to be used during phase two. This argument is optional. The default configuration file is named `renamedg_config` and is located in the directory in which the command is run. The single quotations may be required on some platforms.
- asm\_diskstring=discoverystring, discoverystring ...
- Specifies the Oracle ASM discovery strings. The `asm_diskstring` value must be specified if the Oracle ASM disks are not in the default location for the platform. The single quotations may be required on some platforms, usually when wildcard characters are specified.
- clean={true|false}
- Specifies whether to tolerate errors that is otherwise ignored. The default is true.
- check={true|false}
- Specifies a boolean value that is used in the second phase. If true, then the tool prints the list of changes that are to be made to the disks. No writes are issued. It is an optional parameter that defaults to false.
- confirm={true|false}
- Specifies a boolean value that is used in the second phase. If false, then the tool prints the changes that are to be made and seeks confirmation before actually making the changes. It is an optional value that defaults to false. If check is set to true, then the value of this parameter is redundant.

- `verbose={true|false}`
- Specifies verbose execution when `verbose=true`. The default is false.
- `keep_voting_files={true|false}`

Specifies whether voting files are kept in the renamed disk group. The default is false which deletes the voting files from the renamed disk group.

## 66.2 Specifying the Sector Size for Disk Drives

Oracle ASM provides the capability to specify a sector size of 512 bytes or 4096 kilobytes with the `SECTOR_SIZE` disk group attribute when creating disk groups. Oracle ASM provides support for 4 KB sector disk drives without a performance penalty.

You can use the optional `SECTOR_SIZE` disk group attribute with the `CREATE DISKGROUP` SQL statement to specify disks with the sector size set to the value of `SECTOR_SIZE` for the disk group. Oracle ASM provides support for 4 KB sector disk drives without negatively affecting performance. The `SECTOR_SIZE` disk group attribute can be set only during disk group creation.

The values for `SECTOR_SIZE` can be set to 512, 4096, or 4K if the disks support those values. The default value is platform dependent. The `COMPATIBLE.ASM` and `COMPATIBLE.RDBMS` disk group attributes must be set to 11.2 or higher to set the sector size to a value other than the default value.

Note: Oracle Automatic Storage Management Cluster File System (Oracle ACFS) does not support 4 KB sector drives. There is a performance penalty for Oracle ACFS when using 4 KB sector disk drives in 512 sector emulation mode.

The following validations apply to the sector size disk group attribute.

- Oracle ASM prevents disks of different sector sizes from being added to the same disk group. This validation occurs during `CREATE DISKGROUP`, `ALTER DISKGROUP ADD DISK`, and `ALTER DISKGROUP MOUNT` operations.
- If the `SECTOR_SIZE` attribute is explicitly specified when creating a disk group, then Oracle ASM attempts to verify that all disks discovered through disk search strings have a sector size equal to the specified value. If one or more disks were found to have a sector size different from the specified value, or if Oracle ASM was not able to verify a disk sector size, then the create operation fails.
- Oracle ASM also attempts to verify disk sector size during the mount operation and the operation fails if one or more disks have a sector size different than the value of the `SECTOR_SIZE` attribute.
- If the `SECTOR_SIZE` attribute is not specified when creating a disk group and Oracle ASM can verify that all discovered disks have the same sector value, then that value is assumed for the disk group sector size that is created. If the disks have different sector sizes, the create operation fails.
- When new disks are added to an existing disk group using the `ALTER DISKGROUP .. ADD DISK` SQL statement, you must ensure that the new disks to be added have the same value as the `SECTOR_SIZE` disk group attribute. If the new disks have different sector sizes, the alter operation fails.
- You can determine the sector size value that has either been assumed or explicitly set for a successful disk group creation by querying the `V$ASM_ATTRIBUTE` view or run the `ASMCMD lsattr` command. You can also query the `SECTOR_SIZE` column in the `V$ASM_DISKGROUP` view.

```
SQL> SELECT name, value FROM V$ASM_ATTRIBUTE WHERE name = 'sector_size'
AND group_number = 1;
```

| NAME        | VALUE |
|-------------|-------|
| sector_size | 512   |

```
SQL> SELECT group_number, sector_size FROM V$ASM_DISKGROUP WHERE
group_number = 1;
```

| GROUP_NUMBER | SECTOR_SIZE |
|--------------|-------------|
| 1            | 512         |

Not all disks support all of the possible SECTOR\_SIZE values. The sector size attribute setting must be compatible with the physical hardware.

We can use the SECTOR\_SIZE attribute with the CREATE DISKGROUP SQL statement to specify the sector size of the disk drive on which the Oracle ASM disk group is located.

```
Creating a disk group of 4K sector size
CREATE DISKGROUP data NORMAL REDUNDANCY
FAILGROUP controller1 DISK
 '/devices/diska1', '/devices/diska2', '/devices/diska3', '/devices/diska4'
FAILGROUP controller2 DISK
 '/devices/diskb1', '/devices/diskb2', '/devices/diskb3',
 '/devices/diskb4'
ATTRIBUTE 'compatible.asm' = '11.2', 'compatible.rdbms' = '11.2',
 'sector_size'=4096;
```

## 66.3 Oracle ASM Configuration Assistant (ASMCA)

Oracle ASM Configuration Assistant provides a GUI interface for installing and configuring Oracle ASM instances, disk groups, volumes, and file systems.

In addition, the ASMCA command-line interface provides functionality for installing and configuring Oracle ASM instances, disk groups, volumes, and file systems in a non-GUI utility.

## 66.4 ASMCMD Enhancements

This feature extends Oracle ASM Command Line Utility (ASMCMD) to provide full functionality, so that any operation that can be performed with SQL commands can be performed with ASMCMD. The added functionality includes the following areas:

- Disk, failure group, and disk group operations
- Disk group attribute operations
- Oracle ASM File Access Control user and group operations
- Template operations
- Oracle ASM instance operations
- File operations
- Oracle ASM volume operations

Changes were also made to standardize ASMCMD command-line and command options.

### ASMCMD commands:

| Category                              | Commands                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ASMCMD Instance Management Commands   | <a href="#">dsget</a> , <a href="#">dsset</a> , <a href="#">lsct</a> , <a href="#">lsop</a> , <a href="#">lspwusr</a> , <a href="#">orapwusr</a> , <a href="#">shutdown</a> , <a href="#">spbackup</a> , <a href="#">spcopy</a> , <a href="#">spget</a> , <a href="#">spmove</a> , <a href="#">spset</a> , <a href="#">startup</a>                                                                                                                          |
| ASMCMD File Management Commands       | <a href="#">cd</a> , <a href="#">cp</a> , <a href="#">du</a> , <a href="#">find</a> , <a href="#">ls</a> , <a href="#">lsop</a> , <a href="#">mkalias</a> , <a href="#">pwd</a> , <a href="#">rm</a> , <a href="#">rmlalias</a>                                                                                                                                                                                                                             |
| ASMCMD Disk Group Management Commands | <a href="#">chdg</a> , <a href="#">chkdg</a> , <a href="#">dropdg</a> , <a href="#">iostat</a> , <a href="#">lsattr</a> , <a href="#">lsdg</a> , <a href="#">lsdsk</a> , <a href="#">lsod</a> , <a href="#">md_backup</a> , <a href="#">md_restore</a> , <a href="#">mkdg</a> , <a href="#">mount</a> , <a href="#">offline</a> , <a href="#">online</a> , <a href="#">rebal</a> , <a href="#">remap</a> , <a href="#">setattr</a> , <a href="#">umount</a> |
| ASMCMD Template Management Commands   | <a href="#">chtmp</a> , <a href="#">lstmt</a> , <a href="#">mktmp</a> , <a href="#">rmtmp</a>                                                                                                                                                                                                                                                                                                                                                               |
| ASMCMD File Access Control Commands   | <a href="#">chgrp</a> , <a href="#">chmod</a> , <a href="#">chown</a> , <a href="#">groups</a> , <a href="#">grpmod</a> , <a href="#">lsgrp</a> , <a href="#">lsusr</a> , <a href="#">mkgrp</a> , <a href="#">mkusr</a> , <a href="#">passwd</a> , <a href="#">rmgrp</a> , <a href="#">rmusr</a>                                                                                                                                                            |
| ASMCMD Volume Management Commands     | <a href="#">volcreate</a> , <a href="#">voldelete</a> , <a href="#">voldisable</a> , <a href="#">volenable</a> , <a href="#">volinfo</a> , <a href="#">volresize</a> , <a href="#">volset</a> , <a href="#">volstat</a>                                                                                                                                                                                                                                     |

## 66.5 Intelligent Data Placement

Intelligent Data Placement enables you to specify disk regions on Oracle ASM disks for best performance. Using the disk region settings, you can ensure that frequently accessed data is placed on the outermost (hot) tracks which have greater speed and higher bandwidth. In addition, files with similar access patterns are located physically close, reducing latency. Intelligent Data Placement also enables the placement of primary and mirror extents into different hot or cold regions.

Intelligent Data Placement settings can be specified for a file or in disk group templates. The disk region settings can be modified after the disk group has been created. The disk region setting can improve I/O performance by placing more frequently accessed data in regions furthest from the spindle, while reducing your cost by increasing the usable space on a disk.

Intelligent Data Placement works best for the following:

- Databases with data files that are accessed at different rates. A database that accesses all data files in the same way is unlikely to benefit from Intelligent Data Placement.
- Disk groups that are more than 25% full. If the disk group is only 25% full, the management overhead is unlikely to be worth any benefit.
- Disks that have better performance at the beginning of the media relative to the end. Because Intelligent Data Placement leverages the geometry of the disk, it is well suited to JBOD (just a bunch of disks). In contrast, a storage array with LUNs composed of concatenated volumes masks the geometry from Oracle ASM.

The COMPATIBLE.ASM and COMPATIBLE.RDBMS disk group attributes must be set to 11.2 or higher to use Intelligent Data Placement.

Intelligent Data Placement can be managed with the ALTER DISKGROUP ADD or MODIFY TEMPLATE SQL statements and the ALTER DISKGROUP MODIFY FILE SQL statement.

The ALTER DISKGROUP TEMPLATE SQL statement includes a disk region clause for setting hot/mirrorhot or cold/mirrorcold regions in a template:

```
Sql> ALTER DISKGROUP data ADD TEMPLATE datafile_hot
 ATTRIBUTE (HOT MIRRORHOT);
```

The ALTER DISKGROUP ... MODIFY FILE SQL statement that sets disk region attributes for hot/mirrorhot or cold/mirrorcold regions:

```
Sql> ALTER DISKGROUP data MODIFY FILE
 '+data/orcl/datafile/users.266.629156903'
 ATTRIBUTE (HOT MIRRORHOT);
```

When you modify the disk region settings for a file, this action applies to new extensions of the file, but existing file contents are not affected until a rebalance operation. To apply the new Intelligent Data Placement policy for existing file contents, you can manually initiate a rebalance. A rebalance operation uses the last specified policy for the file extents. For information on the rebalance operation

## 67. ASM Enhancements in 12C R1

### General Oracle ASM Enhancements

This feature provides general enhancements to Oracle ASM, including:

Revised version of the physical metadata replication point

Oracle ASM now replicates physically addressed metadata, such as the disk header and allocation tables, within each disk. This enhancement ensures that Oracle ASM is more resilient to bad disk sectors and external corruptions. The disk group attribute `PHYS_META_REPLICATED` is provided to track the replication status of a disk group. Support for increased storage limits

Oracle ASM now supports 511 disk groups. The maximum Oracle ASM disk size is now 32 petabytes (PB).

The `ALTER DISKGROUP` statement has been updated with a `REPLACE` clause.

### 67.1 Oracle Flex ASM

Oracle Flex ASM enables an Oracle ASM instance to run on a separate physical server from the database servers. Many Oracle ASM instances can be clustered to support a large number of database clients.

This feature enables you to consolidate all the storage requirements into a single set of disk groups. All these disk groups can be managed by a small set of Oracle ASM instances running in a single cluster.

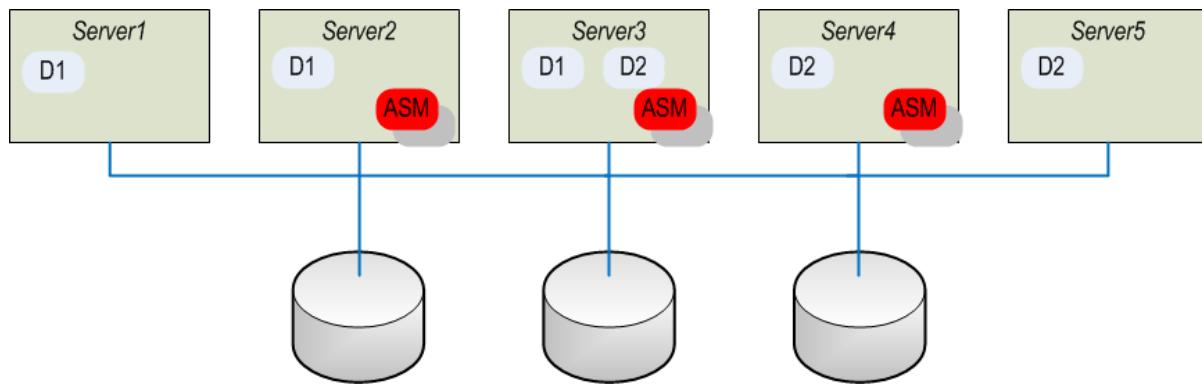
Oracle Flex ASM supports only Oracle Database 12c Release 1 (15.1) and later releases.

Databases are about storage, after all; and the storage layer is where ASM rules. In Oracle Database 12c, ASM has undergone significant changes and enhancements. It will take a lot more than just one article to go over all of them in detail, so here's an overview of the landmark enhancements and explain them with enough detail so that you can hit the ground running.

#### Flex ASM Cluster

Flex ASM Cluster is perhaps it is the biggest change in ASM since its introduction 10 years ago. Consider an Oracle Cluster that you have on a number of nodes. Each cluster node must run a certain set of processes including the ASM processes, which is called ASM instance. The ASM instance manages the storage attached to the server. However, this poses a serious potential issue for the database. If that ASM instance fails, the database instance also shuts down since it can't communicate with the storage anymore. The dependence of the ASM instance caused some to balk at the prospect of ASM as a viable storage management solution. Well, worry no more.

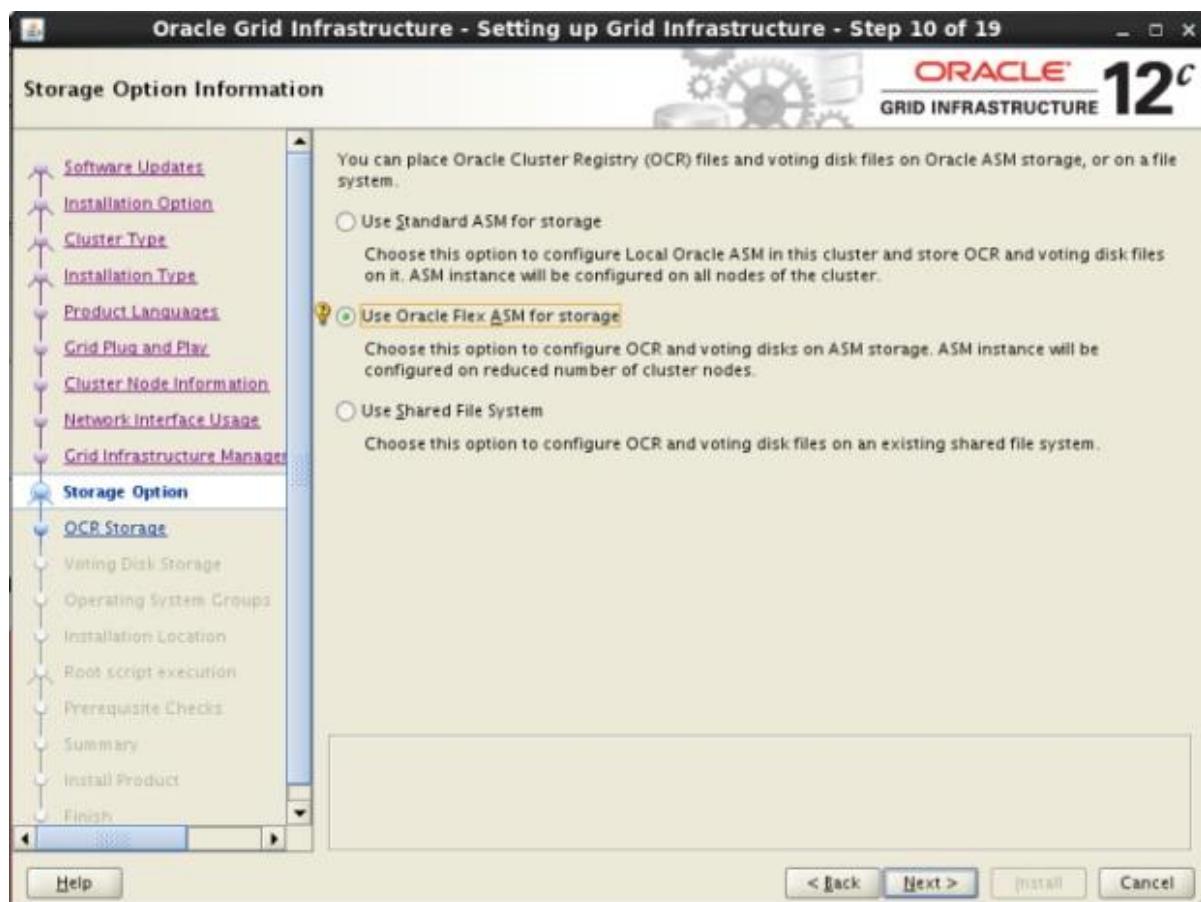
To solve this problem, Oracle Database 12c introduces a new concept in the ASM architecture that lets a database instance to connect to an ASM instance on a *different* node of the same cluster. The sever where there is a database instance but no ASM instance is called a client. However, recall that the ASM processes fulfill a very important function - to identify the exact contents of the database is asking for by looking up the extent map. Since there is no ASM instance, there is no extent map available on those client servers. To get the information on what is stored (the metadata), this client needs to connect to a real ASM instance with metadata, called a Flex ASM instance. By default, there are three Flex ASM instances in a cluster; but that number can be changed. Here is how the Flex ASM works in a 5 node cluster.



There are five nodes in this cluster named Server1 through Server5. There are two databases - D1 and D2. Instances of D1 run on Server1, Server 2 and Server3 while those of D2 run on Server3, Server4 and Server5. Prior to Oracle Database 12c, you would have had to run an ASM instance on each of these servers as well, most likely named +ASM1 through +ASM5. In Flex ASM architecture, however, you could choose to run the ASM instance (called Flex ASM Instance) on Servers 2, 3 and 4. The servers 1 and 5 do not need to run ASM instance. These servers are simply called ASM clients. The clients communicate with the Flex ASM nodes to get the metadata and then get the data from the disk to the database instance. The ASM instance does not run on these client nodes, saving some amount of memory and computing capacity.

However, not requiring to run ASM instances is not the only advantage of Flex ASM; it also insulates you from the damaging effect of an ASM instance going down. Let's examine the impact of the ASM instance crashing. Since there is no ASM instance running on 1 and 5, the case of ASM coming down does not apply to these two nodes. If ASM instance on Server3 crashes, Oracle will automatically start that ASM instance on any of the two nodes Server1 and Server5. Recall that Flex ASM allows for an ASM client to make the storage available to the database instance. So the Server3 (where the ASM instance crashed) will simply become an ASM Client for the database instance running there. The database instance will not crash.

You can enable Flex ASM when you install the cluster. Of course, it is relevant only when the cluster is installed; not in a single instance. While installing the Grid Infrastructure, you have an option to choose the type of ASM instance, shown in the screen below:



You choose the "Use Oracle Flex ASM for storage" option for Flex ASM. It is also possible to convert an existing normal ASM instance to Flex ASM. The default number of ASM instances to run on the cluster is 3; so Oracle will choose any three nodes on the cluster to run the instances. The other nodes will become client nodes. The number of nodes where Oracle ASM runs is known as ASM Cardinality. You can check that by the `srvctl` tool.

```
$ srvctl config asm
```

And look for "ASM instance count". If the ASM instances fail, the Oracle Clusterware automatically starts up enough ASM instances on other nodes to maintain the cardinality. You can also change the cardinality by the same tool:

```
$ srvctl modify asm -count 2
```

How does the database instance running on a client node decide which of the 3 Flex ASM instances (assuming you have 3) to connect to? That is done automatically by Flex Cluster infrastructure by choosing the least loaded ASM instance.

## 67.2 Oracle ASM Disk Scrubbing

Oracle ASM disk scrubbing checks logical data corruptions and repairs the corruptions automatically in normal and high redundancy disks groups. The feature is designed so that it does not have any impact to the regular input and output (I/O) operations in production systems. The scrubbing process repairs logical corruptions using the Oracle ASM mirror disks. Disk scrubbing uses Oracle ASM rebalancing to minimize I/O overhead.

The scrubbing process is visible in fields of the `V$ASM_OPERATION` view.

## 67.3 Oracle ASM Disk Resync Enhancements

The disk resync enhancements enable fast recovery from instance failure and faster resync performance overall. Oracle ASM disk resync enables multiple disks to be brought online simultaneously or to control the speed of the resync operation. Oracle ASM disk resync has a resync power limit to control resync parallelism and improve performance. Disk resync checkpoint functionality provides faster recovery from instance failures by enabling the resync to resume from the point at which the process was interrupted or stopped, instead of starting from the beginning.

## 67.4 Even Read For Disk Groups

The even read feature distributes data reads evenly across all the disks in a disk group. For each I/O request presented to the system, there may be one or more disks that contain the data. With this feature, each request to read can be sent to the least loaded of the possible source disks.

Even read functionality is enabled by default on all Oracle Database and Oracle ASM instances of version 15.1 and higher in non-Exadata environments. The functionality is enabled in an Exadata environment when there is a failure. Even read functionality is applicable only to disk groups with normal or high redundancy.

## 67.5 Shared Oracle ASM Password File in a Disk Group

This feature implements the infrastructure needed to address the bootstrapping issues of Oracle ASM shared password file in an Oracle ASM disk group. This feature solves the bootstrapping problem for storing shared Oracle ASM password files in a disk group.

## 67.6 Updated Key Management Framework

This feature updates Oracle key management commands to unify the key management application programming interface (API) layer. The updated key management framework makes interacting with keys in the wallet easier and adds new key metadata that describes how the keys are being used.

## 67.7 Oracle ASM Rebalance Enhancements

Oracle ASM rebalance enhancements improve scalability, performance, and reliability of the rebalance operation. This feature extends the rebalance process to operate on multiple disk groups in a single instance. In addition, this feature improves support for thin provisioning, user-data validation, and improved error handling.

In 12c ASM, a more detailed and more accurate work plan is created at the beginning of each rebalance operation.

In addition, DBAs can separately generate and view the work plan before performing a rebalance operation.

This allows DBA to plan and execute various changes such as adding storage, removing storage or moving between different storage systems.

DBAs can generate the work plan using the ESTIMATE WORK command.

Querying from V\$ASM\_ESTIMATE view give an idea of required time of that operation based on current workload on the system. So while planning such operation, DBAs needs to take consideration of system load and without running the original operation can make approximate estimation of their operation at ASM level.

If you want to drop a disk from a diskgroup and if you specify wrong disk name ,then this will estimation will fail. Need to give proper disk name.

## 68. Introduction to Oracle Data Guard

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data. Data Guard provides a comprehensive set of services that create, maintain, manage, and monitor one or more standby databases to enable production Oracle databases to survive disasters and data corruptions. Data Guard maintains these standby databases as transactionally consistent copies of the production database.

Then, if the production database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby database to the production role, minimizing the downtime associated with the outage. Data Guard can be used with traditional backup, restoration, and cluster techniques to provide a high level of data protection and data availability. With Data Guard, administrators can optionally improve production database performance by offloading resource-intensive backup and reporting operations to standby systems.

### 68.1 Data Guard Configurations

A Data Guard configuration consists of one production database and one or more standby databases. The databases in a Data Guard configuration are connected by Oracle Net and may be dispersed geographically. There are no restrictions on where the databases are located, provided they can communicate with each other.

For example, you can have a standby database on the same system as the production database, along with two standby databases on other systems at remote locations. You can manage primary and standby databases using the SQL command-line interfaces or the Data Guard broker interfaces, including a command-line interface (DGMGRL) and a graphical user interface that is integrated in Oracle Enterprise Manager.

#### 68.1.1 Primary Database

A Data Guard configuration contains one production database, also referred to as the primary database that functions in the primary role. This is the database that is accessed by most of your applications. The primary database can be either a single-instance Oracle database or an Oracle Real Application Clusters (RAC) database.

#### 68.1.2 Standby Databases

A standby database is a transactionally consistent copy of the primary database. Using a backup copy of the primary database, you can create up to nine standby databases and incorporate them in a Data Guard configuration. Once created, Data Guard automatically maintains each standby database by transmitting redo data from the primary database and then applying the redo to the standby database. Similar to a primary database, a standby database can be either a single-instance Oracle database or an Oracle RAC database. The types of standby databases are as follows:

##### Physical standby database

Provides a physically identical copy of the primary database, with on disk database structures that are identical to the primary database on a block-for-block basis. The database schema, including indexes, are the same.

A physical standby database is kept synchronized with the primary database, through Redo Apply, which recovers the redo data received from the primary database and applies the redo to the physical standby database. As of Oracle Database 11g release 1 (11.1), a physical standby database can receive and apply redo while it is open for read-only access. A physical standby database can therefore be used concurrently for data protection and reporting.

### Logical standby database

Contains the same logical information as the production database, although the physical organization and structure of the data can be different. The logical standby database is kept synchronized with the primary database through SQL Apply, which transforms the data in the redo received from the primary database into SQL statements and then executing the SQL statements on the standby database.

A logical standby database can be used for other business purposes in addition to disaster recovery requirements. This allows users to access a logical standby database for queries and reporting purposes at any time. Also, using a logical standby database, you can upgrade Oracle Database software and patch sets with almost no downtime. Thus, a logical standby database can be used concurrently for data protection, reporting, and database upgrades.

### Snapshot Standby Database

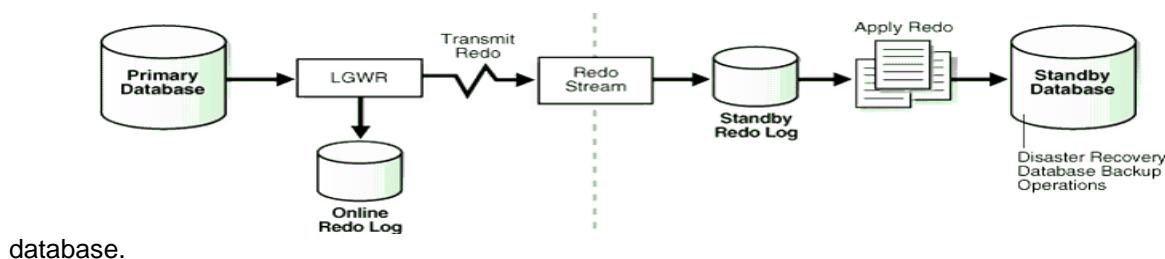
A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database. Like a physical or logical standby database, a snapshot standby database receives and archives redo data from a primary database. Unlike a physical or logical standby database, a snapshot standby database does not apply the redo data that it receives.

The redo data received by a snapshot standby database is not applied until the snapshot standby is converted back into a physical standby database, after first discarding any local updates made to the snapshot standby database.

A snapshot standby database is best used in scenarios that require a temporary, updatable snapshot of a physical standby database. Note that because redo data received by a snapshot standby database is not applied until it is converted back into a physical standby, the time needed to perform a role transition is directly proportional to the amount of redo data that needs to be applied.

### 68.1.3 Configuration Example

The figure below shows a typical Data Guard configuration that contains a primary database that transmits redo data to a standby database. The standby database is remotely located from the primary database for disaster recovery and backup operations. You can configure the standby database at the same location as the primary database. However, for disaster recovery purposes, Oracle recommends you configure standby databases at remote locations. It shows a typical Data Guard configuration in which redo is being applied out of standby redo log files to a standby



database.

## 69. Data Guard

### 69.1 Overview

Oracle Data Guard is the management, monitoring, and automation software infrastructure that creates, maintains, and monitors one or more standby DBs to protect enterprise data from failures, disasters, errors, and corruptions.

Data Guard maintains these standby DBs as transactionally consistent copies of the production database. These standby databases can be located at remote disaster recovery sites thousands of miles away from the production data center, or they may be located in the same city, same campus, or even in the same building. If the production database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby DB to the production role, thus minimizing the downtime associated with the outage, and preventing any data loss.

Data Guard can be used in combination with other Oracle High Availability (HA) solutions such as Real Application Clusters (RAC) and Recovery Manager (RMAN), to provide a high level of data protection and data availability that is unprecedented in the industry.

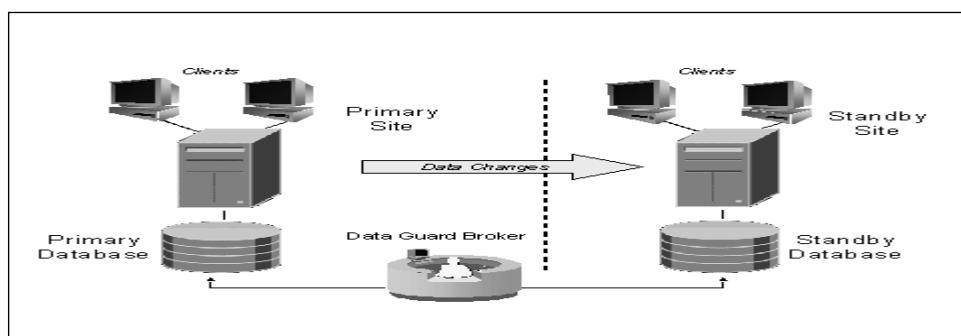


Figure 1-1. Hi-Level Overview of Oracle Data Guard

In order to implement the availability option using ORACLE 7/8/8i/9i we need to ensure that there is a 2<sup>nd</sup> server with

1. Similar hardware (may be less number of CPU/memory).
2. Same operating system including patches
3. Same oracle version including patches.

In case of Oracle 8i we can configure all the clients with an alias containing fail over option targeting standby server. Thus the moment production server goes down immediately we can cancel the MRM mode on the standby server and opens the database as a regular database (no longer treating that one as Hot Standby anymore).

Since this database was running in MRM mode the recovery has been performed already with generated archived files coming from the production sever. With 8i option we no longer need to waste time for the recovery. Crystal reports, analyzing the statistics and some other time taking process can be done on the second hot standby server. Without wasting time on the primary server for backup processes we can take the backup of the hot standby by shutting down the database (cold backup).

## 69.2 Concepts

1. Analyzing high availability and disaster protection.
2. Two loosely connected sites (Ethernet) primary and standby combines into a single easily managed disaster recovery solution.
3. Uniform management solution
4. Support for both GUI and CLI interfaces
5. Automating creating / configuring of physical hot standby by databases.
6. Oracle data guards log transport shifts the data in the form of archive log files to the standby database.
7. Fail over and switchover automation.
8. Monitoring alert and control mechanism

### 69.2.1 No Data Loss

Log transport service will not commit application on primary database until the modifications are also available on standby database but May or may not be applied so far.

### 69.2.2 No Data Divergence

No data divergence extends any data loss by prohibiting primary database modifications when connection to standby database is not available.

## 69.3 Overview of Oracle Data Guard Functional Components

### 69.3.1 Data Guard Configuration

A Data Guard configuration consists of one production (or primary) database and up to nine standby databases. The databases in a Data Guard configuration are connected by Oracle Net and may be dispersed geographically. There are no restrictions on where the databases are located, provided that they can communicate with each other. However, for disaster recovery, it is recommended that the standby databases are hosted at sites that are geographically separated from the primary site.

### 69.3.2 Role Management

Using Data Guard, the role of a database can be switched from a primary role to a standby role and vice versa, ensuring no data loss in the process, and minimizing downtime. There are two kinds of role transitions - a switchover and a failover. A switchover is a role reversal between the primary database and one of its standby databases.

This is typically done for planned maintenance of the primary system. During a switchover, the primary database transitions to a standby role and the standby database transitions to the primary role. The transition occurs without having to re-create either database. A failover is an irreversible transition of a standby database to the primary role. This is only done in the event of a catastrophic failure of the primary database, which is assumed to be lost and to be used again in the Data Guard configuration, it must be re-instantiated as a standby from the new primary.

## 69.4 Data Guard Protection Modes

In some situations, a business cannot afford to lose data at any cost. In other situations, some applications require maximum database performance and can tolerate a potential loss of data. Data Guard provides three distinct modes of data protection to satisfy these varied requirements:

1. Maximum Protection: This mode offers the highest level of data protection. Data is synchronously transmitted to the standby database from the primary database and transactions are not committed on the primary database unless the redo data is available on at least one standby database configured in this mode. If the last standby database configured in this mode becomes unavailable, processing stops on the primary database. This mode ensures no-data-loss.
2. Maximum Availability: This mode is similar to the maximum protection mode, including zero data loss. However, if a standby database becomes unavailable (for example, because of network connectivity problems), processing continues on the primary database. When the fault is corrected, the standby database is automatically resynchronized with the primary database.
3. Maximum Performance: This mode offers slightly less data protection on the primary database, but higher performance than maximum availability mode. In this mode, as the primary database processes transactions, redo data is asynchronously shipped to the standby database. The commit operation of the primary database does not wait for the standby database to acknowledge receipt of redo data before completing write operations on the primary database. If any standby destination becomes unavailable, processing continues on the primary database and there is little effect on primary database performance.

## 69.5 Data Guard Broker

The Oracle Data Guard Broker is a distributed management framework that automates and centralizes the creation, maintenance, and monitoring of Data Guard configurations. All management operations can be performed either through Oracle Enterprise Manager, which uses the Broker, or through the Broker's specialized command-line interface (DGMGRL).

## 69.6. Data Guard Architecture

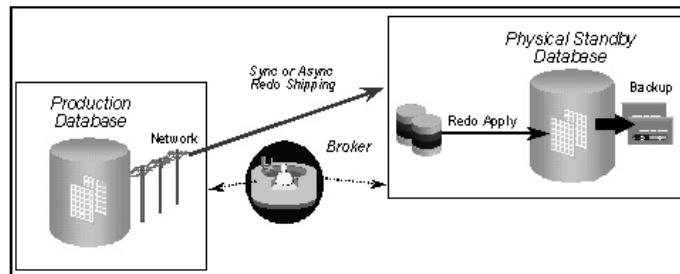


Figure 2-1. Data Guard Architecture

1. Primary database in production database which is used to create standby database.
2. Physical standby database is database replicated, created from backup of primary database
3. Log Transport Service (LTS) controls automated transportation of archive logs from primary to standby.
4. Network configuration, primary DB is connected to one or more remote standby DB over the network
5. Log apply service (LAS)- applies the Archive logs to standby database.
6. Data guard broker is the management and monitoring tool.

## 69.7 What's New in Oracle Data Guard 10g Release 1?

This section will highlight some of the key new features of Oracle Data Guard 10g Release 1.

### 69.7.1 Real Time Apply

With this feature, redo data can be applied on the standby database (whether Redo Apply or SQL Apply) as soon as they have written to a Standby Redo Log (SRL). Prior releases of Data Guard require this redo data to be archived at the standby database in the form of archive logs before they can be applied. The Real Time Apply feature allows standby databases to be closely synchronized with the primary database, enabling up-to-date and real-time reporting. This also enables faster switchover and failover times, which in turn reduces planned and unplanned downtime for the business.

The impact of a disaster is often measured in terms of Recovery Point Objective (RPO - i.e. how much data can a business afford to lose in the event of a disaster) and Recovery Time Objective (RTO - i.e. how much time a business can afford to be down in the event of a disaster). With Oracle Data Guard, when Maximum Protection is used in combination with Real Time Apply, businesses get the benefits of both zero data loss as well as minimal downtime in the event of a disaster and this makes Oracle Data Guard the only solution available today with the best RPO and RTO benefits for a business.

### 69.7.2 Integration with Flashback Database

Data Guard in 10g has been integrated with the Flashback family of features to bring the Flashback feature benefits to a Data Guard configuration. One such benefit is human error protection. In Oracle9i, administrators may configure Data Guard with an apply delay to protect standby databases from possible logical data corruptions that occurred on the primary database.

The side-effects of such delays are that any reporting that gets done on the standby database is done on old data, and switchover/failover gets delayed because the accumulated logs have to be applied first. In Data Guard 10g, with the Real Time Apply feature, such delayed-reporting or delayed-switchover/failover issues do not exist, and - if logical corruptions do land up affecting both the primary and standby database, the administrator may decide to use Flashback Database on both the primary and standby databases to quickly revert the databases to an earlier point-in-time to back out such user errors. Another benefit that such integration provides is during failovers. In releases prior to 10g, following any failover operation, the old primary database must be recreated (as a new standby database) from a backup of the new primary database,

if the administrator intends to bring it back in the Data Guard configuration. This may be an issue when the database sizes are fairly large, and the primary/standby databases are hundreds/thousands of miles away. However, in Data Guard 10g, after the primary server fault is repaired, the primary database may simply be brought up in mounted mode, "flashed back" (using flashback database) to the SCN at which the failover occurred, and then brought back as a standby database in the Data Guard configuration. No reinstatement is required.

### 69.7.3 Simplified Browser-based Interface

Administration of a Data Guard configuration can be done through the new streamlined browser-based HTML interface of Enterprise Manager, which enables complete standby database lifecycle management. The focus of such streamlined administration is on:

- Ease of use.
- Management based on best practices.
- Pre-built integration with other HA features.

## 69.8 What's New in Oracle Data Guard 10g Release 2?

This section will highlight some of the key new features of Oracle Data Guard 10g Release 2.

### 69.8.1 Fast-Start Failover

This capability allows Data Guard to automatically, and quickly fail over to a previously chosen, synchronized standby database in the event of loss of the primary database, without requiring any manual steps to invoke the failover, and without incurring any data loss. Following a fast-start failover, once the old primary database is repaired, Data Guard automatically reinstates it to be a standby database. This act restores high availability to the Data Guard configuration.

### 69.8.2 Improved Redo Transmission

Several enhancements have been made in the redo transmission architecture to make sure redo data generated on the primary database can be transmitted as quickly and efficiently as possible to the standby database(s).

### 69.8.3 Easy conversion of a physical standby database to a reporting database

A physical standby database can be activated as a primary database, opened read/write for reporting purposes, and then flashed back to a point in the past to be easily converted back to a physical standby database. At this point, Data Guard automatically synchronizes the standby database with the primary database. This allows the physical standby database to be utilized for read/write reporting and cloning activities.

### 69.8.4 Automatic deletion of applied archived redo log files in logical standby DB's

Archived logs, once they are applied on the logical standby database, are automatically deleted, reducing storage consumption on the logical standby and improving Data Guard manageability. Physical standby databases have already had this functionality since Oracle Database 10g Release 1, with Flash Recovery Area.

### 69.8.5 Fine-grained monitoring of Data Guard configurations

Oracle Enterprise Manager has been enhanced to provide granular, up-to-date monitoring of Data Guard configurations, so that administrators may make an informed and expedient decision regarding managing this configuration.

## 69.9 Data Guard Benefits

### 69.15.1 Disaster recovery and high availability

Data Guard provides an efficient and comprehensive disaster recovery and high availability solution. Automatic failover and easy-to-manage switchover capabilities allow quick role reversals between primary and standby DB's, minimizing the downtime of the primary database for planned and unplanned outages.

### 69.11.2 Complete data protection

A standby database also provides an effective safeguard against data corruptions and user errors. Storage level physical corruptions on the primary database do not propagate to the standby database. Similarly, logical corruptions or user errors that cause the primary database to be permanently damaged can be resolved. Finally, the redo data is validated at the time it is received at the standby database and further when applied to the standby database.

### **69.15.3 Efficient utilization of system resources**

A physical standby database can be used for backups and read-only reporting, thereby reducing the primary database workload and saving valuable CPU and I/O cycles. In Oracle Database 10g Release 2, a physical standby database can also be easily converted back and forth between being a physical standby database and an open read/write database.

A logical standby database allows its tables to be simultaneously available for read-only access while they are updated from the primary database. A logical standby database also allows users to perform data manipulation operations on tables that are not updated from the primary database. Finally, additional indexes and materialized views can be created in the logical standby database for better reporting performance.

### **69.15.4 Flexibility in data protection to balance availability against performance requirements**

Oracle Data Guard offers the maximum protection, maximum availability, and maximum performance modes to help enterprises balance data availability against system performance requirements.

### **69.15.5 Protection from communication failures**

If network connectivity is lost between the primary and one or more standby DB's, redo data cannot be sent from the primary to those standby databases. Once connectivity is re-established, the missing redo data is automatically detected by Data Guard and the necessary archive logs are automatically transmitted to the standby databases. The standby DB's are resynchronized with the primary database, with no manual intervention by the administrator.

### **69.15.6 Centralized and simple management**

Data Guard Broker automates the management and monitoring tasks across the multiple databases in a Data Guard configuration. Administrators may use either Oracle Enterprise Manager or the Broker's own specialized command-line interface (DGMGRL) to take advantage of this integrated management framework.

### **69.15.7 Integrated with Oracle database**

Data Guard is available as an integrated feature of the Oracle Database (Enterprise Edition) at no extra cost.

## 70. Data Guard Protection Modes

This chapter contains the following sections:

- Data Guard Protection Modes
- Setting the Data Protection Mode of a Primary Database

### **Data Guard Protection Modes**

In these descriptions, a synchronized standby database is meant to be one that meets the minimum requirements of the configured data protection mode and that does not have a redo gap.

#### **Maximum Availability**

This protection mode provides the highest level of data protection that is possible without compromising the availability of a primary database. Transactions do not commit until all redo data needed to recover those transactions has been written to the online redo log and to at least one synchronized standby database.

If the primary database cannot write its redo stream to at least one synchronized standby database, it effectively switches to maximum performance mode to preserve primary database availability and operates in that mode until it is again able to write its redo stream to a synchronized standby database. This mode ensures that no data loss will occur if the primary database fails, but only if a second fault does not prevent a complete set of redo data from being sent from the primary database to at least one standby database.

#### **Maximum Performance**

This protection mode provides the highest level of data protection that is possible without affecting the performance of a primary database. This is accomplished by allowing transactions to commit as soon as all redo data generated by those transactions has been written to the online log. Redo data is also written to one or more standby databases, but this is done asynchronously with respect to transaction commitment, so primary database performance is unaffected by delays in writing redo data to the standby database(s).

This protection mode offers slightly less data protection than maximum availability mode and has minimal impact on primary database performance.

This is the default protection mode.

#### **Maximum Protection**

This protection mode ensures that zero data loss occurs if a primary database fails. To provide this level of protection, the redo data needed to recover a transaction must be written to both the online redo log and to at least one synchronized standby database before the transaction commits. To ensure that data loss cannot occur, the primary database will shut down, rather than continue processing transactions, if it cannot write its redo stream to at least one synchronized standby database.

Because this data protection mode prioritizes data protection over primary database availability, Oracle recommends that a minimum of two standby databases be used to protect a primary database that runs in maximum protection mode to prevent a single standby database failure from causing the primary database to shut down.

#### **Setting the Data Protection Mode of a Primary Database**

Perform the following steps to change the data protection mode of a primary database:

Step 1 Select a data protection mode that meets your availability, performance and data protection requirements.

Step 2 Verify that redo transport is configured to at least one standby database

The value of the LOG\_ARCHIVE\_DEST\_n database initialization parameter that corresponds to

the standby database must include the redo transport attributes listed in Table 5-1 for the data protection mode that you are moving to.

If the primary database has more than one standby database, only one of those standby databases must use the redo transport settings listed in Table. The standby database must also have a standby redo log.

Table Required Redo Transport Attributes for Data Protection Modes

| Maximum Availability | Maximum Performance | Maximum Protection |
|----------------------|---------------------|--------------------|
| AFFIRM               | NOAFFIRM            | AFFIRM             |
| SYNC                 | ASYNC               | SYNC               |
| DB_UNIQUE_NAME       | DB_UNIQUE_NAME      | DB_UNIQUE_NAME     |

Step 3 Verify that the DB\_UNIQUE\_NAME database initialization parameter has been set to a unique name on the primary and standby database.

For example, if the DB\_UNIQUE\_NAME parameter has not been defined on either database, the following SQL statements might be used to assign a unique name to each database.

Execute this SQL statement on the primary database:

```
SQL> ALTER SYSTEM SET DB_UNIQUE_NAME='CHICAGO' SCOPE=SPFILE;
```

Execute this SQL statement on the standby database:

```
SQL> ALTER SYSTEM SET DB_UNIQUE_NAME='BOSTON' SCOPE=SPFILE;
```

Step 4 Verify that the LOG\_ARCHIVE\_CONFIG database initialization parameter has been defined on the primary and standby database and that its value includes a DG\_CONFIG list that includes the DB\_UNIQUE\_NAME of the primary and standby database.

For example, if the LOG\_ARCHIVE\_CONFIG parameter has not been defined on either database, the following SQL statement could be executed on each database to configure the LOG\_ARCHIVE\_CONFIG parameter:

```
SQL> ALTER SYSTEM SET
2> LOG_ARCHIVE_CONFIG='DG_CONFIG=(CHICAGO,BOSTON)' ;
```

Step 5 Skip this step unless you are raising your protection mode.

Shut down the primary database and restart it in mounted mode if the protection mode is being changed to Maximum Protection or being changed from Maximum Performance to Maximum Availability.

If the primary database is an Oracle Real Applications Cluster, shut down all of the instances and then start and mount a single instance. For example:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

Step 6 Set the data protection mode.

Execute the following SQL statement on the primary database

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE {AVAILABILITY |
PERFORMANCE | PROTECTION} ;
```

Step 7 Open the primary database.

If the database was restarted in Step 5, open the database:

```
SQL> ALTER DATABASE OPEN;
```

Step 8 Confirm that the primary database is operating in the new protection mode.

Perform the following query on the primary database to confirm that it is operating in the new protection mode:

```
SQL> SELECT PROTECTION_MODE FROM V$DATABASE;
```

## 71. Starting Up and Shutting Down a Physical Standby Database

This section describes how to start up and shut down a physical standby database.

### 71.1 Starting Up a Physical Standby Database

Use the SQL\*Plus STARTUP command to start a physical standby database. The SQL\*Plus STARTUP command starts, mounts, and opens a physical standby database in read-only mode when it is invoked without any arguments. Once mounted or opened, a physical standby database can receive redo data from the primary database.

**Note:** When Redo Apply is started on a physical standby database that has not yet received redo data from the primary database, an ORA-01112 message may be returned. This indicates that Redo Apply is unable to determine the starting sequence number for media recovery. If this occurs, manually retrieve an archived redo log file from the primary database and register it on the standby database, or wait for redo transport to begin before starting Redo Apply.

### 71.2 Shutting Down a Physical Standby Database

Use the SQL\*Plus SHUTDOWN command to stop Redo Apply and shut down a physical standby database. Control is not returned to the session that initiates a database shutdown until shutdown is complete. If the primary database is up and running, defer the standby destination on the primary database and perform a log switch before shutting down the physical standby database.

### 71.3 Opening a Physical Standby Database

A physical standby database can only be opened in read-only mode. An open physical standby database can continue to receive and apply redo data from the primary database. This allows read-only transactions to be offloaded from a primary database to a physical standby and increases the return on investment in a physical standby database.

A physical standby database instance cannot be opened if Redo Apply is active on any instance, even if one or more instances have already been opened. If Redo Apply is active, use the following SQL statement to stop Redo Apply before attempting to open a physical standby database instance:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

Once a physical standby database has been opened, Redo Apply can be started and stopped at any time. You can perform queries against the standby while Redo Apply is active. (This is also known Real Time Query.)

**Note:** A physical standby database cannot be open while Redo Apply is active unless the physical standby database and its primary database are running in 11g compatibility mode.

**Note:**

You must issue the SET TRANSACTION READ ONLY command before performing a distributed query on a physical standby database.

## 72. Standby Database Types

A standby database is a transactionally consistent copy of an Oracle production database that is initially created from a backup copy of the primary database. Once the standby database is created and configured, Data Guard automatically maintains the standby database by transmitting primary database redo data to the standby system, where the redo data is applied to the standby database.

### 72.1 A standby database can be one of these types:

1. Physical standby database
2. Logical standby database
3. Snapshot standby database.

If needed, either a physical or a logical standby database can assume the role of the primary database and take over production processing. A Data Guard configuration can include any combination of these types of standby databases.

#### 72.1.1 Physical Standby Databases

A physical standby database is an exact, block-for-block copy of a primary database. A physical standby is maintained as an exact copy through a process called Redo Apply, in which redo data received from a primary database is continuously applied to a physical standby database using the database recovery mechanisms.

### 72.2 Benefits of a Physical Standby Database

A physical standby database provides the following benefits:

- Disaster recovery and high availability
- A physical standby database is a robust and efficient disaster recovery and high availability solution. Easy-to-manage switchover and failover capabilities allow easy role reversals between primary and physical standby databases, minimizing the downtime of the primary database for planned and unplanned outages.

### 72.3 Data protection

A physical standby database can prevent data loss, even in the face of unforeseen disasters. A physical standby database supports all datatypes, and all DDL and DML operations that the primary database can support. It also provides a safeguard against data corruptions and user errors. Storage level physical corruptions on the primary database will not be propagated to a standby database. Similarly, logical corruptions or user errors that would otherwise cause data loss can be easily resolved.

### 72.4 Reduction in primary database workload

Oracle Recovery Manager (RMAN) can use a physical standby database to off-load backups from a primary database, saving valuable CPU and I/O cycles. A physical standby database can also be queried while Redo Apply is active, which allows queries to be offloaded from the primary to a physical standby, further reducing the primary workload.

### 72.5 Performance

The Redo Apply technology used by a physical standby database is the most efficient mechanism for keeping a standby database updated with changes being made at a primary database because it applies changes using low-level recovery mechanisms which bypass all SQL level code layers.

## 72.5.1 Logical Standby Databases

A logical standby database is initially created as an identical copy of the primary database, but it later can be altered to have a different structure. The logical standby database is updated by executing SQL statements. This allows users to access the standby database for queries and reporting at any time.

Thus, the logical standby database can be used concurrently for data protection and reporting operations. Data Guard automatically applies information from the archived redo log file or standby redo log file to the logical standby database by transforming the data in the log files into SQL statements and then executing the SQL statements on the logical standby database. .

Because the logical standby database is updated using SQL statements, it must remain open. Although the logical standby database is opened in read/write mode, its target tables for the regenerated SQL are available only for read-only operations. While those tables are being updated, they can be used simultaneously for other tasks such as reporting, summations, and queries. Moreover, these tasks can be optimized by creating additional indexes and materialized views on the maintained tables. A logical standby database has some restrictions on datatypes, types of tables, and types of DDL and DML operations.

## 72.6 Benefits of a Logical Standby Database:

A logical standby database is ideal for high availability (HA) while still offering data recovery (DR) benefits. Compared to a physical standby database, a logical standby database provides significant additional HA benefits:

## 72.7 Protection against additional kinds of failure

Because logical standby analyzes the redo and reconstructs logical changes to the database, it can detect and protect against certain kinds of hardware failure on the primary that could potentially be replicated through block level changes. Oracle supports having both physical and logical standbys for the same primary server.

## 72.8 Efficient use of resources

A logical standby database is open read/write while changes on the primary are being replicated. Consequently, a logical standby database can simultaneously be used to meet many other business requirements, for example it can run reporting workloads that would problematical for the primary's throughput. It can be used to test new software releases and some kinds of applications on a complete and accurate copy of the primary's data.

It can host other applications and additional schemas while protecting data replicated from the primary against local changes. It can be used to assess the impact of certain kinds of physical restructuring (for example, changes to partitioning schemes). Because a logical standby identifies user transactions and replicates only those changes while filtering out background system changes, it can efficiently replicate only transactions of interest.

## 72.9 Workload distribution

Logical standby provides a simple turnkey solution for creating up-to-the-minute, consistent replicas of a primary database that can be used for workload distribution. As the reporting workload increases, additional logical standbys can be created with transparent load distribution without affecting the transactional throughput of the primary server.

## 72.10 Optimized for reporting and decision support requirements

A key benefit of logical standby is that significant auxiliary structures can be created to optimize the reporting workload; structures that could have a prohibitive impact on the primary's

transactional response time. A logical standby can have its data physically reorganized into a different storage type with different partitioning, have many different indexes, have on-demand refresh materialized views created and maintained, and it can be used to drive the creation of data cubes and other OLAP data views.

## 72.11 Minimizing downtime on software upgrades

Logical standby can be used to greatly reduce downtime associated with applying patchsets and new software releases. A logical standby can be upgraded to the new release and then switched over to become the active primary. This allows full availability while the old primary is converted to a logical standby and the patchset is applied.

### 72.15.1 Snapshot Standby Databases

A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database. A snapshot standby database receives and archives, but does not apply, redo data from its primary database. Redo data received from the primary database is applied when a snapshot standby database is converted back into a physical standby database, after discarding all local updates to the snapshot standby database.

A snapshot standby database typically diverges from its primary database over time because redo data from the primary database is not applied as it is received. Local updates to the snapshot standby database will cause additional divergence. The data in the primary database is fully protected however, because a snapshot standby can be converted back into a physical standby database at any time, and the redo data received from the primary will then be applied.

## 72.12 Benefits of a Snapshot Standby Database

A snapshot standby database is a fully updatable standby database that provides disaster recovery and data protection benefits that are similar to those of a physical standby database. Snapshot standby databases are best used in scenarios where the benefit of having a temporary, updatable snapshot of the primary database justifies additional administrative complexity and increased time to recover from primary database failures.

The benefits of using a snapshot standby database include the following:

- It provides an exact replica of a production database for development and testing purposes, while maintaining data protection at all times
- It can be easily refreshed to contain current production data by converting to a physical standby and resynchronizing.
- The ability to create a snapshot standby, test, resynchronize with production, and then again create a snapshot standby and test, is a cycle that can be repeated as often as desired. The same process can be used to easily create and regularly update a snapshot standby for reporting purposes where read/write access to data is required.

## 72.13 User Interfaces for Administering Data Guard Configurations

You can use the following interfaces to configure, implement, and manage a Data Guard configuration:

### **Oracle Enterprise Manager(OEM)**

Enterprise Manager provides a GUI interface for the Data Guard broker that automates many of the tasks involved in creating, configuring, and monitoring a Data Guard environment.

### **SQL\*Plus Command-line interface**

Several SQL\*Plus statements use the STANDBY keyword to specify operations on a standby database. Other SQL statements do not include standby-specific syntax, but they are useful for performing operations on a standby database.

### **Initialization parameters**

Several initialization parameters are used to define the Data Guard environment.

### **Data Guard broker command-line interface (DGMGRL)**

The DGMGRL command-line interface is an alternative to using Oracle Enterprise Manager. The DGMGRL command-line interface is useful if you want to use the broker to manage a Data Guard configuration from batch programs or scripts.

## 73. Monitoring Stand by Databases

### 73.1 Primary DB Changes That Require Manual Intervention at a Physical Standby

Most structural changes made to a primary database are automatically propagated through redo data to a physical standby database. Table lists primary database structural and configuration changes which require manual intervention at a physical standby database.

Table: Primary Database Changes That Require Manual Intervention at a Physical Standby

| Primary Database Change                                                                                      | Action Required on Physical Standby Database                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Add a datafile or create a tablespace                                                                        | No action is required if the STANDBY_FILE_MANAGEMENT database initialization parameter is set to AUTO. If this parameter is set to MANUAL, the new datafile must be copied to the physical standby database.          |
| Drop or delete a tablespace or datafile                                                                      | Delete datafile from primary and physical standby database after the redo data containing the DROP or DELETE command is applied to the physical standby.                                                              |
| Use transportable tablespaces                                                                                | Move tablespace between the primary and the physical standby database.                                                                                                                                                |
| Rename a datafile                                                                                            | Rename the datafile on the physical standby database.                                                                                                                                                                 |
| Add or drop a redo log file group                                                                            | Evaluate the configuration of the redo log and standby redo log on the physical standby database and adjust as necessary.                                                                                             |
| Perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause                                   | Copy the datafile containing the unlogged changes to the physical standby database.                                                                                                                                   |
| Grant or revoke administrative privileges or change the password of a user who has administrative privileges | If the REMOTE_LOGIN_PASSWORDFILE initialization parameter is set to SHARED or EXCLUSIVE, replace the password file on the physical standby database with a fresh copy of the password file from the primary database. |
| Reset the TDE master encryption key                                                                          | Replace the database encryption wallet on the physical standby database with a fresh copy of the database encryption wallet from the primary database.                                                                |
| Change initialization parameters                                                                             | Evaluate whether a corresponding change must be made to the initialization parameters on the physical standby database.                                                                                               |

### 73.1.1 Adding a Datafile or Creating a Tablespace

The STANDBY\_FILE\_MANAGEMENT database initialization parameter controls whether the addition of a datafile to the primary database is automatically propagated to a physical standby databases.

- If the STANDBY\_FILE\_MANAGEMENT parameter on the physical standby database is set to AUTO, any new datafiles created on the primary database are automatically created on the physical standby database.
- If the STANDBY\_FILE\_MANAGEMENT database parameter on the physical standby database is set to MANUAL, a new datafile must be manually copied from the primary database to the physical standby databases after it is added to the primary database.

Note that if an existing datafile from another database is copied to a primary database, that it must also be copied to the standby database and that the standby control file must be re-created, regardless of the setting of STANDBY\_FILE\_MANAGEMENT parameter.

Using the STANDBY\_FILE\_MANAGEMENT Parameter with Raw Devices

Note:

Do not use the following procedure with databases that use Oracle Managed Files. Also, if the raw device path names are not the same on the primary and standby servers, use the DB\_FILE\_NAME\_CONVERT database initialization parameter to convert the path names.

By setting the STANDBY\_FILE\_MANAGEMENT parameter to AUTO whenever new datafiles are added or dropped on the primary database, corresponding changes are made in the standby database without manual intervention. This is true as long as the standby database is using a file system. If the standby database is using raw devices for datafiles, then the STANDBY\_FILE\_MANAGEMENT parameter will continue to work, but manual intervention is needed.

This manual intervention involves ensuring the raw devices exist before Redo Apply applies the redo data that will create the new datafile. On the primary database, create a new tablespace where the datafiles reside in a raw device. At the same time, create the same raw device on the standby database. For example:

```
SQL> CREATE TABLESPACE MTS2 -
> DATAFILE '/dev/raw/raw100' size 1m;
Tablespace created.

SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
The standby database automatically adds the datafile because the raw
devices exist. The standby alert log shows the following:
Fri Apr 8 09:49:31 2005
Media Recovery Log
/u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_08/o1_mf_1_7_
15ffgt0z_.arc
Recovery created file /dev/raw/raw100
Successfully added datafile 6 to media recovery
Datafile #6: '/dev/raw/raw100'
Media Recovery Waiting for thread 1 sequence 8 (in transit)
However, if the raw device was created on the primary system but not on
the standby, then Redo Apply will stop due to file-creation errors. For
example, issue the following statements on the primary database:
SQL> CREATE TABLESPACE MTS3 -
> DATAFILE '/dev/raw/raw101' size 1m;
Tablespace created.
```

```

SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
The standby system does not have the /dev/raw/raw101 raw device created.
The standby alert log shows the following messages when recovering the
archive:
Fri Apr 8 10:00:22 2005
Media Recovery Log
/u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_08/o1_mf_1_8_
15ffjrov_.arc
File #7 added to control file as 'UNNAMED00007'.
Originally created as:
'/dev/raw/raw101'
Recovery was unable to create the file as:
'/dev/raw/raw101'
MRP0: Background Media Recovery terminated with error 1269
Fri Apr 8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21857.trc:
ORA-01269: cannot add datafile '/dev/raw/raw101' - file could not be
created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-26441: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Some recovered datafiles maybe left media fuzzy
Media recovery may continue but open resetlogs may fail
Fri Apr 8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21857.trc:
ORA-01269: cannot add datafile '/dev/raw/raw101' - file could not be
created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-26441: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Fri Apr 8 10:00:22 2005
MTS; MRP0: Background Media Recovery process shutdown
ARCH: Connecting to console port...

```

## Recovering from Errors

To correct the problems perform the following steps:

Create the raw device on the standby database and assign permissions to the Oracle user.

Query the V\$DATAFILE view. For example:

```

SQL> SELECT NAME FROM V$DATAFILE;

NAME.

/u01/MILLER/MTS/system01.dbf
/u01/MILLER/MTS/undotbs01.dbf
/u01/MILLER/MTS/sysaux01.dbf
/u01/MILLER/MTS/users01.dbf
/u01/MILLER/MTS/mts.dbf
/dev/raw/raw100
/u01/app/oracle/product/8.1.0/dbs/UNNAMED00007

SQL> ALTER SYSTEM SET -
> STANDBY_FILE_MANAGEMENT=MANUAL;

SQL> ALTER DATABASE CREATE DATAFILE
2 '/u01/app/oracle/product/8.1.0/dbs/UNNAMED00007'
3 AS
4 '/dev/raw/raw101';

```

1. In the standby alert log you should see information similar to the following:

```

Fri Apr 8 10:09:30 2005
 alter database create datafile
 '/dev/raw/raw101' as '/dev/raw/raw101'

Fri Apr 8 10:09:30 2005
 Completed: alter database create datafile
 '/dev/raw/raw101' a

```
2. On the standby database, set STANDBY\_FILE\_MANAGEMENT to AUTO and restart Redo Apply:

```

SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=AUTO;
SQL> RECOVER MANAGED STANDBY DATABASE DISCONNECT;

```

At this point Redo Apply uses the new raw device datafile and recovery continues.

### 73.1.2 Dropping Tablespaces and Deleting Datafiles

When a tablespace is dropped or a datafile is deleted from a primary database, the corresponding datafile(s) must be deleted from the physical standby database. The following example shows how to drop a tablespace:

```

SQL> DROP TABLESPACE tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;

```

To verify that deleted datafiles are no longer part of the database, query the V\$DATAFILE view.

Delete the corresponding datafile on the standby system after the redo data that contains the previous changes is applied to the standby database. For example:

```
% rm /disk1/oracle/oradata/payroll/s2tbs_4.dbf
```

On the primary database, after ensuring the standby database applied the redo information for the dropped tablespace, you can remove the datafile for the tablespace. For example:

```
% rm /disk1/oracle/oradata/payroll/tbs_4.dbf
```

### 73.1.3 Using Drop Tablespace Including Contents And Datafiles

You can issue the SQL DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES statement on the primary database to delete the datafiles on both the primary and standby databases. To use this statement, the STANDBY\_FILE\_MANAGEMENT initialization parameter must be set to AUTO. For example, to drop the tablespace at the primary site:

```

SQL> DROP TABLESPACE INCLUDING CONTENTS -
> AND DATAFILES tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;

```

### 73.1.4 Using Transportable Tablespaces with a Physical Standby Database

You can use the Oracle transportable tablespaces feature to move a subset of an Oracle database and plug it in to another Oracle database, essentially moving tablespaces between the databases.

To move or copy a set of tablespaces into a primary database when a physical standby is being used, perform the following steps:

- Generate a transportable tablespace set that consists of datafiles for the set of tablespaces being transported and an export file containing structural information for the set of tablespaces.
- Transport the tablespace set:
  - Copy the datafiles and the export file to the primary database.
  - Copy the datafiles to the standby database.

- The datafiles must be copied in a directory defined by the DB\_FILE\_NAME\_CONVERT initialization parameter. If DB\_FILE\_NAME\_CONVERT is not defined, then issue the ALTER DATABASE RENAME FILE statement to modify the standby control file after the redo data containing the transportable tablespace has been applied and has failed. The STANDBY\_FILE\_MANAGEMENT initialization parameter must be set to AUTO.
- Plug in the tablespace.
- Invoke the Data Pump utility to plug the set of tablespaces into the primary database. Redo data will be generated and applied at the standby site to plug the tablespace into the standby database.

### 73.1.5 Renaming a Datafile in the Primary Database

When you rename one or more datafiles in the primary database, the change is not propagated to the standby database. Therefore, if you want to rename the same datafiles on the standby database, you must manually make the equivalent modifications on the standby database because the modifications are not performed automatically, even if the STANDBY\_FILE\_MANAGEMENT initialization parameter is set to AUTO.

The following steps describe how to rename a datafile in the primary database and manually propagate the changes to the standby database.

- To rename the datafile in the primary database, take the tablespace offline:
 

```
SQL> ALTER TABLESPACE tbs_4 OFFLINE;
```
- Exit from the SQL prompt and issue an operating system command, such as the following UNIX mv command, to rename the datafile on the primary system:
 

```
% mv /disk1/oracle/oradata/payroll/tbs_4.dbf
/disk1/oracle/oradata/payroll/tbs_x.dbf
```
- Rename the datafile in the primary database and bring the tablespace back online:
 

```
SQL> ALTER TABLESPACE tbs_4 RENAME DATAFILE
2> '/disk1/oracle/oradata/payroll/tbs_4.dbf'
3> TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
SQL> ALTER TABLESPACE tbs_4 ONLINE;
```
- Connect to the standby database, query the V\$ARCHIVED\_LOG view to verify all of the archived redo log files are applied, and then stop Redo Apply:
 

```
SQL> SELECT SEQUENCE#,APPLIED FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#,
SEQUENCE# APP

8 YES
9 YES
10 YES
11 YES
4 rows selected.
```
- Shut down the standby database:
 

```
SQL> SHUTDOWN;
```
- Rename the datafile at the standby site using an operating system command, such as the UNIX mv command:
 

```
% mv /disk1/oracle/oradata/payroll/tbs_4.dbf
/disk1/oracle/oradata/payroll/tbs_x.dbf
```
- Start and mount the standby database:
 

```
SQL> STARTUP MOUNT;
```
- Rename the datafile in the standby control file. Note that the STANDBY\_FILE\_MANAGEMENT initialization parameter must be set to MANUAL.
 

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/tbs_4.dbf'
2> TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
```
- On the standby database, restart Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> DISCONNECT;
```

If you do not rename the corresponding datafile at the standby system, and then try to refresh the standby database control file, the standby database will attempt to use the renamed datafile, but it will not find it. Consequently, you will see error messages similar to the following in the alert log:

```
ORA-00283: recovery session canceled due to errors
ORA-01157: cannot identify/lock datafile 4 - see DBWR trace file
ORA-01110: datafile 4: '/Disk1/oracle/oradata/payroll/tbs_x.dbf'
```

### 73.1.6 Add or Drop a Redo Log File Group

The configuration of the redo log and standby redo log on a physical standby database should be reevaluated and adjusted as necessary after adding or dropping a redo log file group on the primary database.

Take the following steps to add or drop a redo log file group or standby redo log file group on a physical standby database:

- Stop Redo Apply.
- If the STANDBY\_FILE\_MANAGEMENT initialization parameter is set to AUTO, change the value to MANUAL.
- Add or drop a log file group.
- Restore the STANDBY\_FILE\_MANAGEMENT initialization parameter and the Redo Apply options to their original states.
- Restart Redo Apply.

### 73.1.7 NOLOGGING or Unrecoverable Operations

When you perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause, the standby database is invalidated and may require substantial DBA administrative activities to repair. You can specify the SQL ALTER DATABASE or SQL ALTER TABLESPACE statement with the FORCELOGGING clause to override the NOLOGGING setting. However, this statement will not repair an already invalidated database.

### 73.1.8 Refresh the Password File

If the REMOTE\_LOGIN\_PASSWORDFILE database initialization parameter is set to SHARED or EXCLUSIVE, the password file on a physical standby database must be replaced with a fresh copy from the primary database after granting or revoking administrative privileges or changing the password of a user with administrative privileges.

Failure to refresh the password file on the physical standby database may cause authentication of redo transport sessions or connections as SYSDBA or SYSOPER to the physical standby database to fail.

### 73.1.9 Reset the TDE Master Encryption Key

The database encryption wallet on a physical standby database must be replaced with a fresh copy of the database encryption wallet from the primary database whenever the TDE master encryption key is reset on the primary database.

Failure to refresh the database encryption wallet on the physical standby database will prevent access to encrypted columns on the physical standby database that are modified after the master encryption key is reset on the primary database.

## 73.2 Recovering Through the OPEN RESETLOGS Statement

Data Guard allows recovery on a physical standby database to continue after the primary database has been opened with the RESETLOGS option. When an ALTER DATABASE OPEN RESETLOGS statement is issued on the primary database, the incarnation of the database changes, creating a new branch of redo data.

When a physical standby database receives a new branch of redo data, Redo Apply automatically takes the new branch of redo data. For physical standby databases, no manual intervention is required if the standby database did not apply redo data past the new resetlogs SCN (past the start of the new branch of redo data). The following table describes how to resynchronize the standby database with the primary database branch.

| If the standby database... .                                                                                                                                   | Then... .                                                                                                                                                      | Perform these steps... .                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Has not applied redo data past Redo the new resetlogs SCN (past the start of the new branch of redo data)                                                      | Apply Redo the automatically takes the new branch of redo data.                                                                                                | No manual intervention is necessary. The MRP automatically resynchronizes the standby database with the new branch of redo data.                                                        |
| Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is enabled on the standby database     | The standby database is recovered in the future of the new branch of redo data. The MRP automatically resynchronizes the standby database with the new branch. | Follow the procedure in "flashback_dg_specific_point.doc" to flashback a physical standby database. Restart Redo Apply to continue application of redo data onto new reset logs branch. |
| Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is not enabled on the standby database | The primary database has diverged from the standby on the primary database branch.                                                                             | Re-create the physical standby database following the procedures in "flashback_dg_specific_point.doc". The MRP automatically resynchronizes the standby database with the new branch.   |
| Is missing intervening archived redo log files from the branch of redo data                                                                                    | The MRP cannot continue until the missing log files are retrieved.                                                                                             | Locate and register missing archived redo log files from each branch.                                                                                                                   |
| Is missing archived redo log files from the end of the previous branch of redo data.                                                                           | The MRP cannot continue until the missing log files are retrieved.                                                                                             | Locate and register missing archived redo log files from the previous branch.                                                                                                           |

## 73.3 Monitoring Primary, Physical Standby, and Snapshot Standby Databases

This section describes where to find useful information for monitoring primary and standby databases.

Table summarizes common primary database management actions and where to find information related to these actions.

### Table Sources of Information About Common Primary Database Management Actions

| Primary Database Action                                                                                                    | Primary Site Information                                                  | Standby Site Information                                |
|----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|---------------------------------------------------------|
| Enable or disable a redo thread                                                                                            | Alert log<br>V\$THREAD                                                    | Alert log                                               |
| Display database role, protection mode, protection level, switchover status, fast-start failover information, and so forth | V\$DATABASE                                                               | V\$DATABASE                                             |
| Add or drop a redo log file group                                                                                          | Alert log<br>V\$LOG<br>STATUS column of V\$LOGFILE                        | Alert log                                               |
| CREATE CONTROLFILE                                                                                                         | Alert log                                                                 | Alert log                                               |
| Monitor Redo Apply                                                                                                         | Alert log<br>V\$ARCHIVE_DEST_STATUS                                       | V\$ARCHIVED_LOG<br>V\$LOG_HISTORY<br>V\$MANAGED_STANDBY |
| Change tablespace status                                                                                                   | V\$RECOVER_FILE<br>DBA_TABLESPACES<br>Alert log                           | V\$RECOVER_FILE<br>DBA_TABLESPACES                      |
| Add or drop a datafile or tablespace                                                                                       | DBA_DATA_FILES<br>Alert log                                               | V\$DATAFILE<br>Alert log                                |
| Rename a datafile                                                                                                          | V\$DATAFILE<br>Alert log                                                  | V\$DATAFILE<br>Alert log                                |
| Unlogged or unrecoverable operations                                                                                       | V\$DATAFILE<br>V\$DATABASE                                                | Alert log                                               |
| Monitor redo transport                                                                                                     | V\$ARCHIVE_DEST_STATUS<br>V\$ARCHIVED_LOG<br>V\$ARCHIVE_DEST<br>Alert log | V\$ARCHIVED_LOG<br>Alert log                            |
| Issue OPEN RESETLOGS or CLEAR UNARCHIVED LOGFILES statements                                                               | Alert log                                                                 | Alert log                                               |
| Change initialization parameter                                                                                            | Alert log                                                                 | Alert log                                               |

## 74. Dataguard Services

The following sections explain how Data Guard manages the transmission of redo data, the application of redo data, and changes to the database roles:

### Redo Transport Services

Control the automated transfer of redo data from the production database to one or more archival destinations.

### Apply Services

Apply redo data on the standby database to maintain transactional synchronization with the primary database. Redo data can be applied either from archived redo log files, or, if real-time apply is enabled, directly from the standby redo log files as they are being filled, without requiring the redo data to be archived first at the standby database.

### Role Transitions

Change the role of a database from a standby database to a primary database, or from a primary database to a standby database using either a switchover or a failover operation.

### 74.1 Redo Transport Services

Redo transport services control the automated transfer of redo data from the production database to one or more archival destinations.

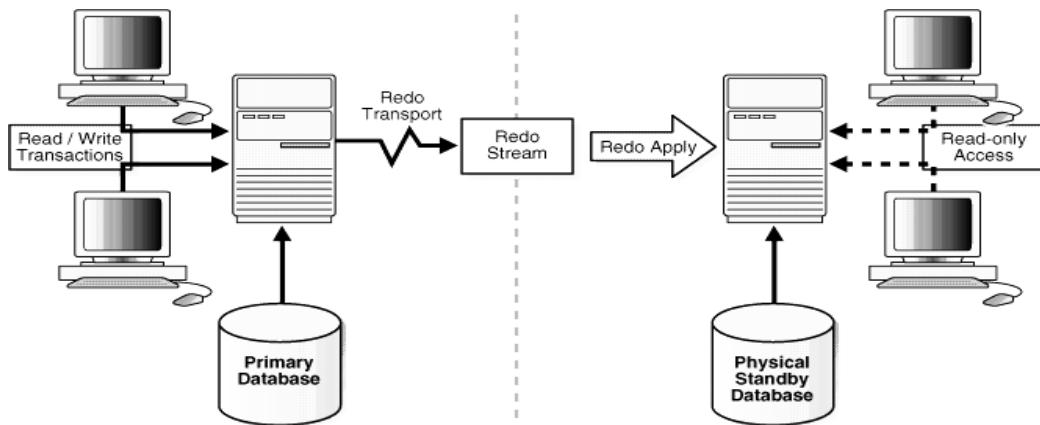
Redo transport services perform the following tasks:

- Transmit redo data from the primary system to the standby systems in the configuration
- Manage the process of resolving any gaps in the archived redo log files due to a network failure
- Automatically detect missing or corrupted archived redo log files on a standby system and automatically retrieve replacement archived redo log files from the primary database or another standby database

### 74.2 Apply Services

The redo data transmitted from the primary database is written to the standby redo log on the standby database. Apply services automatically apply the redo data on the standby database to maintain consistency with the primary database. It also allows read-only access to the data. The main differences between physical and logical standby databases is the manner in which apply services apply the archived redo data:

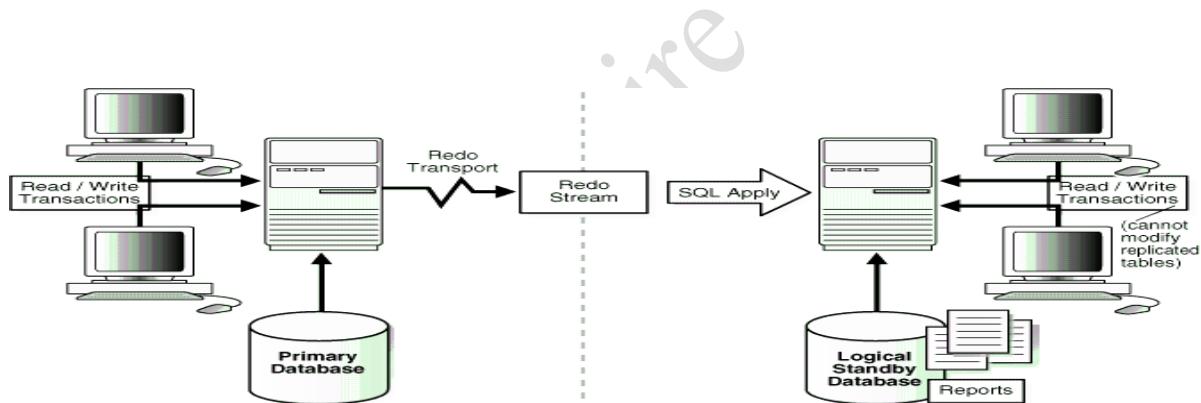
For physical standby databases, Data Guard uses Redo Apply technology, which applies redo data on the standby database using standard recovery techniques of an Oracle database, as shown in figure below.



### Automatic Updating of a Physical Standby Database

For logical standby databases, Data Guard uses SQL Apply technology, which first transforms the received redo data into SQL statements and then executes the generated SQL statements on the logical standby database, as shown in figure below.

### Automatic Updating of a Logical Standby Database



## 74.3 Role Transitions

A Data Guard configuration consists of one database that functions in the primary role and one or more databases that function in the standby role. Typically, the role of each database does not change. However, if Data Guard is used to maintain service in response to a primary database outage, you must initiate a role transition between the current primary database and one standby database in the configuration. To see the current role of the databases, query the DATABASE\_ROLE column in the V\$DATABASE view.

The number, location, and type of standby databases in a Data Guard configuration and the way in which redo data from the primary database is propagated to each standby database determine the role-management options available to you in response to a primary database outage.

This chapter describes how to manage role transitions in a Data Guard configuration. It contains the following topics:

- Introduction to Role Transitions.
- Role Transitions Involving Physical Standby Databases.
- Role Transitions Involving Logical Standby Databases.

The role transitions described in this chapter are invoked manually using SQL statements. You can also use the Oracle Data Guard broker to simplify role transitions and automate failovers.

### 74. 3.1 Introduction to Role Transitions

A database operates in one of the following mutually exclusive roles: primary or standby. Data Guard enables you to change these roles dynamically by issuing the SQL statements described in this chapter, or by using either of the Data Guard broker's interfaces. Oracle Data Guard supports the following role transitions:

#### Switchover

Allows the primary database to switch roles with one of its standby databases. There is no data loss during a switchover. After a switchover, each database continues to participate in the Data Guard configuration with its new role.

#### Failover

Changes a standby database to the primary role in response to a primary database failure. If the primary database was not operating in either maximum protection mode or maximum availability mode before the failure, some data loss may occur. If Flashback Database is enabled on the primary database, it can be reinstated as a standby for the new primary database once the reason for the failure is corrected.

### 74. 3.2 Preparing for a Role Transition

Before starting any role transition, perform the following preparations:

Verify that each database is properly configured for the role that it is about to assume.

#### Note:

- You must define the LOG\_ARCHIVE\_DEST\_n and LOG\_ARCHIVE\_DEST\_STATE\_n parameters on each standby database so that when a switchover or failover occurs, all standby sites continue to receive redo data from the new primary database.
- Ensure temporary files exist on the standby database that matches the temporary files on the primary database.
- Remove any delay in applying redo that may be in effect on the standby database that will become the new primary database.
- Before performing a switchover from an Oracle RAC primary database to a physical standby database, shut down all but one primary database instance. Any primary database instances shut down at this time can be started after the switchover completes.
- Before performing a switchover or a failover to an Oracle RAC physical standby database, shut down all but one standby database instance. Any standby database instances shut down at this time can be restarted after the role transition completes.

### 74. 3.3 Choosing a Target Standby Database for a Role Transition

For a Data Guard configuration with multiple standby databases, there are a number of factors to consider when choosing the target standby database for a role transition. These include the following:

- Locality of the standby database.
- The capability of the standby database (hardware specifications—such as the number of CPUs, I/O bandwidth available, and so on).
- The time it will take to perform the role transition. This is affected by how far behind the standby database is in terms of application of redo data, and how much flexibility you have in terms of trading off application availability with data loss.
- Standby database type.

The type of standby chosen as the role transition target determines how other standby databases in the configuration will behave after the role transition. If the new primary was a physical standby before the role transition, all other standby databases in the configuration will become standbys of the new primary. If the new primary was a logical standby before the role transition, then all other logical standbys in the configuration will become standbys of the new primary, but physical standbys in the configuration will continue to be standbys of the old primary and will therefore not protect the new primary.

In the latter case, a future switchover or failover back to the original primary database will return all standbys to their original role as standbys of the current primary. For the reasons described above, a physical standby is generally the best role transition target in a configuration that contains both physical and logical standbys.

**Note:** A snapshot standby cannot be the target of a role transition.

Data Guard provides the V\$DATAGUARD\_STATS view that can be used to evaluate each standby database in terms of the currency of the data in the standby database, and the time it will take to perform a role transition if all available redo data is applied to the standby database. For example:

```
SQL> COLUMN NAME FORMAT A18
SQL> COLUMN VALUE FORMAT A16
SQL> COLUMN TIME_COMPUTED FORMAT A24
SQL> SELECT * FROM V$DATAGUARD_STATS;
NAME VALUE TIME_COMPUTED
----- ----- -----
apply finish time +00 00:00:02.4 15-MAY-2005 10:32:49
 second(1)
 interval
apply lag +00 0:00:04 15-MAY-2005 10:32:49
 second(0)
 interval
transport lag +00 00:00:00 15-MAY-2005 10:32:49
 second(0)
 interval
```

The time at which each of the statistics is computed is shown in the TIME\_COMPUTED column. The V\$DATATGUARD\_STATS.TIME\_COMPUTED column is a timestamp taken when the metric in a V\$DATATGUARD\_STATS row is computed. This column indicates the freshness of the associated metric. This shows that for this standby database, there is no transport lag, that apply services has not applied the redo generated in the last 4 seconds (apply lag), and that it will take apply services 2.4 seconds to finish applying the unapplied redo (apply finish time).

The APPLY LAG and TRANSPORT LAG metrics are computed based on information received from the primary database, and these metrics become stale if communications between the primary and standby database are disrupted. An unchanging value in this column for the APPLY LAG and TRANSPORT LAG metrics indicates that these metrics are not being updated (or have become stale), possibly due to a communications fault between the primary and standby databases.

## 74. 3.4 Switchovers

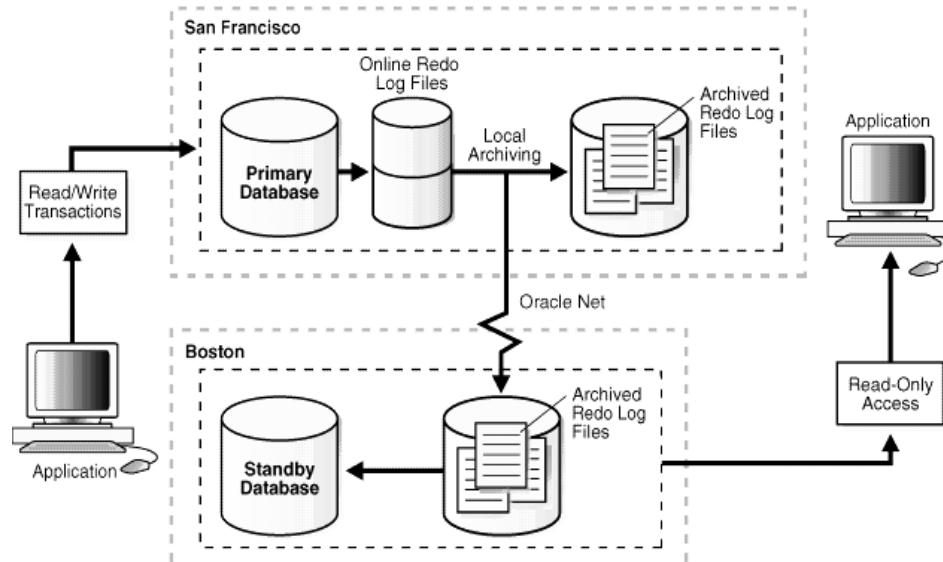
- A switchover is typically used to reduce primary database downtime during planned outages, such as operating system or hardware upgrades, or rolling upgrades of the Oracle database software and patch sets.
- A switchover takes place in two phases. In the first phase, the existing primary database undergoes a transition to a standby role. In the second phase, a standby database undergoes a transition to the primary role.
- Figure 8-1 shows a two-site Data Guard configuration before the roles of the databases are switched. The primary database is in San Francisco, and the standby database is in Boston.

This illustration shows a Data Guard configuration consisting of a primary database and a standby database.

An application is performing read/write transactions on the primary database in San Francisco, from which online redo logs are being archived locally and over Oracle Net services to the standby database in Boston. On the Boston standby location, the archived redo logs are being applied to the standby database, which is performing read-only transactions.

Figure 8-2 shows the Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Data Guard configuration temporarily has two standby databases.

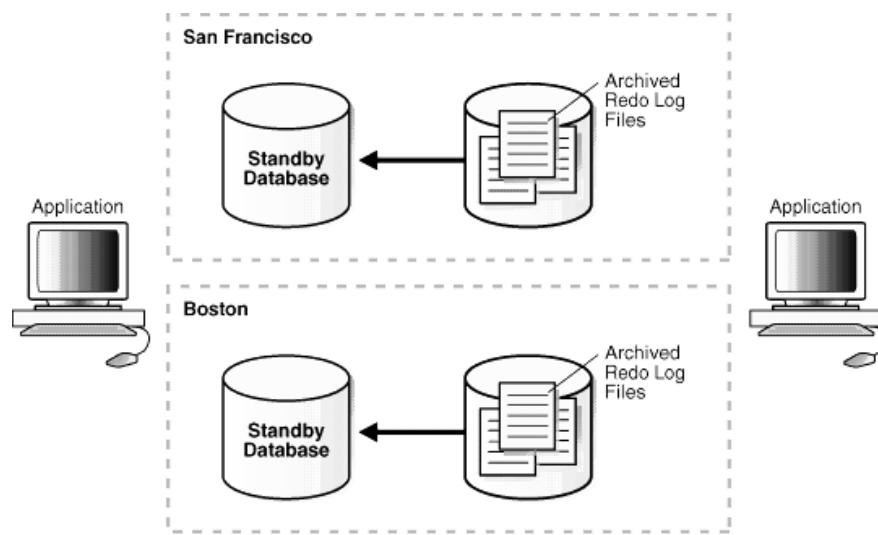
**Figure Standby Databases Before Switchover to the New Primary Database**



This illustration shows a Data Guard configuration during a switchover operation. The San Francisco database (originally the primary database) has changed to the standby role, but the Boston database has not yet changed to the primary role. At this point in time, both the San Francisco and Boston databases are operating in the standby role.

Applications that were previously sending read/write transactions to the San Francisco database are preparing to send read/write transactions to the Boston database. On the Boston standby database, the standby database online redo logs and local archived redo logs are still being generated. However, no redo logs are being sent or received over the Oracle Net network. Both of the standby databases are capable of operating in read-only mode.

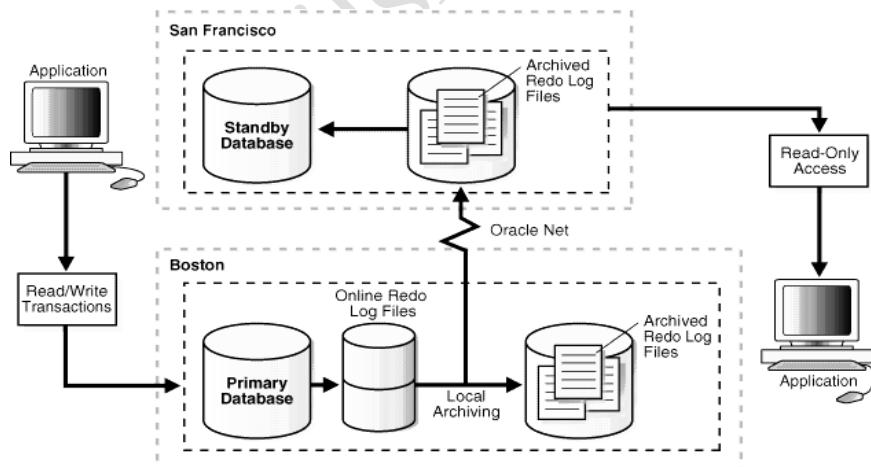
Figure shows the Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

**Figure Data Guard Environment After Switchover**

This illustration shows a Data Guard configuration after a switchover operation has occurred. The San Francisco database (originally the primary database) is now operating as the standby database and the Boston database is now operating as the primary database.

### Preparing for a Switchover

Ensure the prerequisites listed in Section 8.1.1 are satisfied. In addition, the following



prerequisites must be met for a **switchover**:

For switchovers involving a physical standby database, verify that the primary database is open and that redo apply is active on the standby database.

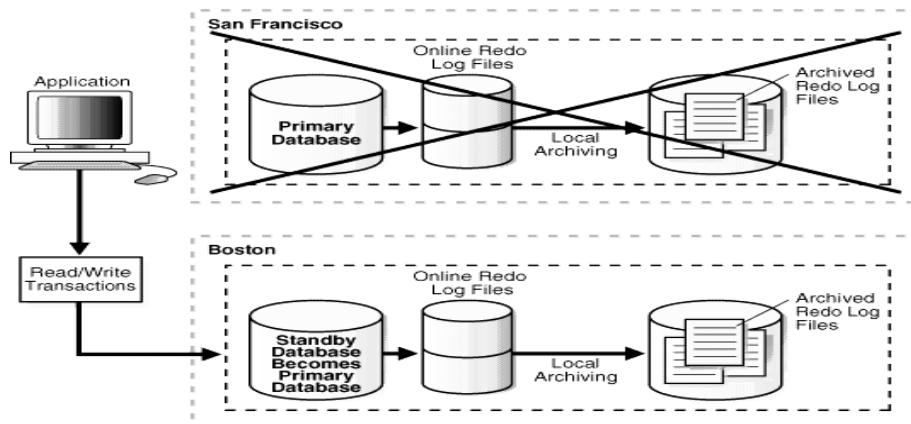
For switchovers involving a logical standby database, verify both the primary and standby database instances are open and that SQL Apply is active.

## 74. 3.5 Failovers

A failover is typically used only when the primary database becomes unavailable, and there is no possibility of restoring it to service within a reasonable period of time. The specific actions performed during a failover vary based on whether a logical or a physical standby database is involved in the failover, the state of the Data Guard configuration at the time of the failover, and

on the specific SQL statements used to initiate the failover. Figure shows the result of a failover from a primary database in San Francisco to a physical standby database in Boston.

**Figure Failover to a Standby Database**



This illustration shows a two-site Data Guard configuration after a system or software failure occurred. In this figure, the primary site (in San Francisco) is crossed out to indicate that the site is no longer operational. The Boston site that was originally a standby site is now operating as the new primary site. Applications that were previously sending read/write transactions to the San Francisco site when it was the primary site are now sending all read/write transactions to the new primary site in Boston. The Boston site is writing to online redo logs and local archived redo logs.

### 74. 3.6 Preparing for a Failover

If possible, before performing a failover, you should transfer as much of the available and unapplied primary database redo data as possible to the standby database. Ensure the prerequisites are satisfied. In addition, the following prerequisites must be met for a failover:

If a standby database currently running in maximum protection mode will be involved in the failover, first place it in maximum performance mode by issuing the following statement on the standby database:

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE;
```

Then, if appropriate standby databases are available, you can reset the desired protection mode on the new primary database after the failover completes.

This is required because you cannot fail over to a standby database that is in maximum protection mode. In addition, if a primary database in maximum protection mode is still actively communicating with the standby database, issuing the ALTER DATABASE statement to change the standby database from maximum protection mode to maximum performance mode will not succeed. Because a failover removes the original primary database from the Data Guard configuration, these features serve to protect a primary database operating in maximum protection mode from the effects of an unintended failover.

### 74. 3.7 Role Transition Triggers

The DB\_ROLE\_CHANGE system event is signaled whenever a role transition occurs. This system event is signaled immediately if the database is open when the role transition occurs, or the next time the database is opened if it is closed when a role transition occurs. The DB\_ROLE\_CHANGE system event can be used to fire a trigger that performs a set of actions whenever a role transition occurs.

## 74. 3.8 Role Transitions Involving Physical Standby Databases

This section describes how to perform switchovers and failovers involving a physical standby database.

### 74.3.8.1 Switchovers Involving a Physical Standby Database

This section describes how to perform a switchover. A switchover must be initiated on the current primary database and completed on the target standby database. The following steps describe how to perform a switchover.

**Step 1. Verify it is possible to perform a switchover.**

On the current primary database, query the SWITCHOVER\_STATUS column of the V\$DATABASE fixed view on the primary database to verify it is possible to perform a switchover. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO STANDBY

1 row selected
```

The TO STANDBY value in the SWITCHOVER\_STATUS column indicates that it is possible to switch the primary database to the standby role. If the TO STANDBY value is not displayed, then verify the Data Guard configuration is functioning correctly (for example, verify all LOG\_ARCHIVE\_DEST\_n parameter values are specified correctly).

After performing these steps, the SWITCHOVER\_STATUS column still displays SESSIONS ACTIVE, you can successfully perform a switchover by appending the WITH SESSION SHUTDOWN clause to the ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY statement described in Step 2.

#### Step 2. Initiate the switchover on the primary database.

To change the current primary database to a physical standby database role, use the following SQL statement on the primary database:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;
```

After this statement completes, the primary database is converted into a standby database. The current control file is backed up to the current SQL session trace file before the switchover. This makes it possible to reconstruct a current control file, if necessary.

#### Step 3. Shut down and restart the former primary instance.

Shutdown the former primary instance, and restart and mount the database:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

At this point in the switchover process, both databases are configured as standby databases (see Figure 8-2).

#### Step 4. Verify the switchover status in the V\$DATABASE view.

After you change the primary database to the physical standby role and the switchover notification is received by the standby databases in the configuration, you should verify if the switchover notification was processed by the target standby database by querying the SWITCHOVER\_STATUS column of the V\$DATABASE fixed view on the target standby database.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO_PRIMARY
1 row selected
```

If the value in the SWITCHOVER\_STATUS column is neither TO\_PRIMARY nor SESSIONS\_ACTIVE, verify that redo apply is active and that redo transport is working properly, and continue to query this view until either TO\_PRIMARY or SESSIONS\_ACTIVE is displayed.

If the value in the SWITCHOVER\_STATUS column is TO\_PRIMARY, go to step 5.

#### **Step 5. Switch the target physical standby database role to the primary role.**

You can switch a physical standby database from the standby role to the primary role when the standby database instance is either mounted in Redo Apply mode or open for read-only access. It must be in one of these modes so that the primary database switchover request can be coordinated. After the standby database is in an appropriate mode, issue the following SQL statement on the physical standby database that you want to change to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

#### **Step 6. Finish the transition of the standby database to the primary role.**

Issue the SQL ALTER DATABASE OPEN statement to open the new primary database:

```
SQL> ALTER DATABASE OPEN;
```

#### **Step 7. Start redo apply on the new physical standby database.**

For example, issue the following statement on the new physical standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT
LOGFILE
DISCONNECT FROM SESSION;
```

### **74. 3.8.2 Failovers Involving a Physical Standby Database**

This section describes how to perform failovers involving a physical standby database.

During failovers involving a physical standby database:

- In all cases, after a failover, the original primary database can no longer participate in the Data Guard configuration.
- In most cases, other logical or physical standby databases not directly participating in the failover remain in the configuration and do not have to be shut down or restarted.
- In some cases, it might be necessary to re-create all standby databases after configuring the new primary database.
- These cases are described, where appropriate, within the failover steps below.

#### **Note:**

Oracle recommends you use only the failover steps and commands described in the following sections to perform a failover. Do not use the ALTER DATABASE ACTIVATE STANDBY DATABASE to perform a failover, because this statement may cause data loss.

#### **Failover Steps**

This section describes the steps that must be performed to transition the selected physical standby database to the primary role. Any other physical or logical standby databases that are also part of the configuration will remain in the configuration and will not need to be shut down or restarted.

If the target standby database was operating in maximum protection mode or maximum availability mode, no gaps in the archived redo log files should exist, and you can proceed directly to Step 6. Otherwise, begin with Step 1 to determine if any manual gap resolution steps must be performed.

**Step 1: Identify and resolve any gaps in the archived redo log files.**

To determine if there are gaps in the archived redo log files on the target standby database, query the V\$ARCHIVE\_GAP view.

The V\$ARCHIVE\_GAP view contains the sequence numbers of the archived redo log files that are known to be missing for each thread. The data returned reflects the highest gap only.

For example:

```
SQL> SELECT THREAD#, LOW_SEQUENCE#, HIGH_SEQUENCE# FROM V$ARCHIVE_GAP;
THREAD# LOW_SEQUENCE# HIGH_SEQUENCE#
----- -----
1 90 92
```

In this example the gap comprises archived redo log files with sequences 90, 91, and 92 for thread 1. If possible, copy all of the identified missing archived redo log files to the target standby database from the primary database and register them. This must be done for each thread.

For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

**Step 2: Repeat Step 1 until all gaps are resolved.**

The query executed in Step 1 displays information for the highest gap only. After resolving that gap, you must repeat Step 1 until the query returns no rows.

**Step 3: Copy any other missing archived redo log files.**

To determine if there are any other missing archived redo log files, query the V\$ARCHIVED\_LOG view on the target standby database to obtain the highest sequence number for each thread.

For example:

```
SQL> SELECT UNIQUE THREAD# AS THREAD, MAX(SEQUENCE#)
 2> OVER (PARTITION BY thread#) AS LAST from V$ARCHIVED_LOG;
THREAD LAST
----- -----
1 100
```

Copy any available archived redo log files from the primary database that contains sequence numbers higher than the highest sequence number available on the target standby database to the target standby database and register them. This must be done for each thread.

For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

After all available archived redo log files have been registered, query the V\$ARCHIVE\_GAP view as described in Step 1 to verify no additional gaps were introduced in Step 3.

**Note:**

If, while performing Steps 1 through 3, you are not able to resolve gaps in the archived redo log files (for example, because you do not have access to the system that hosted the failed primary database), some data loss will occur during the failover.

**Step 4: Stop Managed Recovery.**

Issue the following statement to stop managed recovery:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

**Step 5: Verify that the standby database is ready to become a primary database.**

Query the SWITCHOVER\_STATUS column of the V\$DATABASE view on the target standby database. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO_PRIMARY
1 row selected
```

If the value in the SWITCHOVER\_STATUS column is neither TO\_PRIMARY nor SESSIONS\_ACTIVE, verify that redo apply is active and continue to query this view until either TO\_PRIMARY or SESSIONS\_ACTIVE is displayed.

If the value in the SWITCHOVER\_STATUS column is TO\_PRIMARY, go to step 6.

If the value in the SWITCHOVER\_STATUS column is SESSIONS\_ACTIVE, perform the steps described in Section A.4, "Problems Switching Over to a Standby Database" to identify and terminate active user or SQL sessions that might prevent a switchover from being processed. If, after performing these steps, the SWITCHOVER\_STATUS column still displays SESSIONS\_ACTIVE, you can proceed to Step 6, and append the WITH SESSION SHUTDOWN clause to the switchover statement.

**Step 6: Initiate a failover on the target physical standby database.**

Issue the following statement to initiate the failover:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH;
```

**Step 7: Convert the physical standby database to the primary role.**

Issue the following statement to open the new primary database:

```
SQL> ALTER DATABASE OPEN;
```

**Step 8: Finish the transition of the standby database to the primary database role.**

Issue the SQL ALTER DATABASE OPEN statement to open the new primary database:

```
SQL> ALTER DATABASE OPEN;
```

**Step 9: Back up the new primary database.**

Before issuing the STARTUP statement, back up the new primary database. Performing a backup immediately is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

As a result of the failover, the original primary database can no longer participate in the Data Guard configuration, and all other standby databases are now receiving and applying redo data from the new primary database.

**Step 10: Optionally, restore the failed primary database.**

After a failover, the original primary database can be converted into a physical standby database of the new primary database, or it can be re-created as a physical standby database from a backup of the new primary database .

Once the original primary database has been converted into a standby database, a switchover can be performed to restore it to the primary role.

### 74.3.8.3 Role Transitions Involving Logical Standby Databases

This section describes how to perform switchovers and failovers involving a logical standby database.

#### Switchovers Involving a Logical Standby Database

When you perform a switchover that changes roles between a primary database and a logical standby database, always initiate the switchover on the primary database and complete it on the logical standby database. These steps must be performed in the order in which they are described or the switchover will not succeed.

**Step 1: Verify it is possible to perform a switchover on the primary database.**

On the current primary database, query the SWITCHOVER\_STATUS column of the V\$DATABASE fixed view on the primary database to verify it is possible to perform a switchover.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO STANDBY
1 row selected
```

A value of TO STANDBY or SESSIONS ACTIVE in the SWITCHOVER\_STATUS column indicates that it is possible to switch the primary database to the logical standby role. If one of these values is not displayed, then verify the Data Guard configuration is functioning correctly (for example, verify all LOG\_ARCHIVE\_DEST\_n parameter values are specified correctly).

#### **Step 2: Prepare the current primary database for the switchover.**

To prepare the current primary database for a logical standby database role, issue the following SQL statement on the primary database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement notifies the current primary database that it will soon switch to the logical standby role and begin receiving redo data from a new primary database. You perform this step on the primary database in preparation to receive the LogMiner dictionary to be recorded in the redo stream of the current logical standby database, as described in step 3.

The value PREPARING SWITCHOVER is displayed in the V\$DATABASE.SWITCHOVER\_STATUS column if this operation succeeds.

#### **Step 3: Prepare the target logical standby database for the switchover.**

Use the following statement to build a LogMiner dictionary on the logical standby database that is the target of the switchover:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO PRIMARY;
```

This statement also starts redo transport services on the logical standby database that begins transmitting its redo data to the current primary database and to other standby databases in the Data Guard configuration. The sites receiving redo data from this logical standby database accept the redo data but they do not apply it.

Depending on the work to be done and the size of the database, the switchover can take some time to complete.

The V\$DATABASE.SWITCHOVER\_STATUS on the logical standby database initially shows PREPARING DICTIONARY while the LogMiner dictionary is being recorded in the redo stream. Once this has completed successfully, the SWITCHOVER\_STATUS column shows PREPARING SWITCHOVER.

**Step 4: Ensure the current primary database is ready for the future primary database's redo stream.**

Before you can complete the role transition of the primary database to the logical standby role, verify the LogMiner dictionary was received by the primary database by querying the SWITCHOVER\_STATUS column of the V\$DATABASE fixed view on the primary database. Without the receipt of the LogMiner dictionary, the switchover cannot proceed, because the current primary database will not be able to interpret the redo records sent from the future primary database. The SWITCHOVER\_STATUS column shows the progress of the switchover.

When the query returns the TO LOGICAL STANDBY value, you can proceed with Step 5. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO LOGICAL STANDBY
1 row selected
```

**Note:**

You can cancel the switchover operation by issuing the following statements in the order shown:

Cancel switchover on the primary database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
```

Cancel the switchover on the logical standby database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
```

**Step 5: Switch the primary database to the logical standby database role.**

To complete the role transition of the primary database to a logical standby database, issue the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement waits for all current transactions on the primary database to end and prevents any new users from starting new transactions, and establishes a point in time where the switchover will be committed.

Executing this statement will also prevent users from making any changes to the data being maintained in the logical standby database. To ensure faster execution, ensure the primary database is in a quiet state with no update activity before issuing the switchover statement (for example, have all users temporarily log off the primary database). You can query the V\$TRANSACTION view for information about the status of any current in-progress transactions that could delay execution of this statement.

The primary database has now undergone a role transition to run in the standby database role.

When a primary database undergoes a role transition to a logical standby database role, you do not have to shut down and restart the database.

**Step 6: Ensure all available redo has been applied to the target logical standby database that is about to become the new primary database.**

After you complete the role transition of the primary database to the logical standby role and the switchover notification is received by the standby databases in the configuration, you should verify the switchover notification was processed by the target standby database by querying the SWITCHOVER\_STATUS column of the V\$DATABASE fixed view on the target standby database. Once all available redo records are applied to the logical standby database, SQL Apply automatically shuts down in anticipation of the expected role transition.

The SWITCHOVER\_STATUS value is updated to show progress during the switchover. When the

status is TO PRIMARY, you can proceed with Step 7.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS

TO PRIMARY
1 row selected
```

Step 7: Switch the target logical standby database to the primary database role.

On the logical standby database that you want to switch to the primary role, use the following SQL statement to switch the logical standby database to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

There is no need to shut down and restart any logical standby databases that are in the Data Guard configuration. All other logical standbys in the configuration will become standbys of the new primary, but any physical standby databases will remain standbys of the original primary database.

#### **Step 8: Start SQL Apply on the new logical standby database.**

On the new logical standby database, start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

#### **Failovers Involving a Logical Standby Database**

This section describes how to perform failovers involving a logical standby database. A failover role transition involving a logical standby database necessitates taking corrective actions on the failed primary database and on all bystander logical standby databases. If Flashback Database was not enabled on the failed primary database, you must re-create the database from backups taken from the current primary database. Otherwise, you can convert a failed primary database to be a logical standby database for the new primary database.

Depending on the protection mode for the configuration and the attributes you chose for redo transport services, it might be possible to automatically recover all or some of the primary database modifications.

Step 1: Copy and register any missing archived redo log files to the target logical standby database slated to become the new primary database.

Depending on the condition of the components in the configuration, you might have access to the archived redo log files on the primary database. If so, do the following:

Determine if any archived redo log files are missing on the logical standby database.

Copy missing log files from the primary database to the logical standby database.

Register the copied log files.

You can register an archived redo log files with the logical standby database by issuing the following statement. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
 2> '/disk1/oracle/dbs/log-%r_%s_%t.arc';
Database altered.
```

**Step 2: Enable remote destinations.**

If you have not previously configured role-based destinations, identify the initialization parameters that correspond to the remote logical standby destinations for the new primary database, and manually enable archiving of redo data for each of these destinations.

For example, to enable archiving for the remote destination defined by the LOG\_ARCHIVE\_DEST\_2 parameter, issue the following statement:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE SCOPE=BOTH;
```

To ensure this change will persist if the new primary database is later restarted, update the appropriate text initialization parameter file or server parameter file. In general, when the database operates in the primary role, you must enable archiving to remote destinations, and when the database operates in the standby role, you must disable archiving to remote destinations.

**Step 3: Activate the new primary database.**

Issue the following statement on the target logical standby database (that you are transitioning to the new primary role):

```
SQL> ALTER DATABASE ACTIVATE LOGICAL STANDBY DATABASE FINISH APPLY;
```

This statement stops the RFS process, applies remaining redo data in the standby redo log file before the logical standby database becomes a primary database, stops SQL Apply, and activates the database in the primary database role.

If the FINISH APPLY clause is not specified, then unapplied redo from the current standby redo log file will not be applied before the standby database becomes the primary database.

**Step 4: Recovering other standby databases after a failover****Step 5: Back up the new primary database.**

Back up the new primary database immediately after the Data Guard database failover. Immediately performing a backup is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

**Step 6: Restore the failed primary database.**

After a failover, the original primary database can be converted into a logical standby database of the new primary database or it can be recreated as a logical standby database from a backup of the new primary database .

Once the original primary database has been converted into a standby database, a switchover can be performed to restore it to the primary role.

## 75. Redo Apply Services

This chapter describes how redo data is applied to a standby database. It includes the following topics:

- Introduction to Apply Services
- Apply Services Configuration Options
- Applying Redo Data to Physical Standby Databases
- Applying Redo Data to Logical Standby Databases

### 75.1 Introduction to Apply Services

Apply services automatically apply redo to standby databases to maintain synchronization with the primary database and allow transactionally consistent access to the data.

By default, apply services wait for the full archived redo log file to arrive on the standby database before applying it to the standby database. However, if you use a standby redo log, you can enable real-time apply, which allows Data Guard to recover redo data from the current standby redo log file as it is being filled.

Apply services use the following methods to maintain physical and logical standby databases:

- Redo apply (physical standby databases only)
- Uses media recovery to keep the primary and physical standby databases synchronized.
- SQL Apply (logical standby databases only)
- Reconstitutes SQL statements from the redo received from the primary database and executes the SQL statements against the logical standby database.

Logical standby databases can be opened in read/write mode, but the target tables being maintained by the logical standby database are opened in read-only mode for reporting purposes (providing the database guard was set appropriately). SQL Apply enables you to use the logical standby database for reporting activities, even while SQL statements are being applied.

The sections in this chapter describe Redo Apply, SQL Apply, real-time apply, and delayed apply in more detail.

### 75.2 Apply Services Configuration Options

This section contains the following topics:

- Using Real-Time Apply to Apply Redo Data Immediately
- Specifying a Time Delay for the Application of Archived Redo Log Files

#### Using Real-Time Apply to Apply Redo Data Immediately

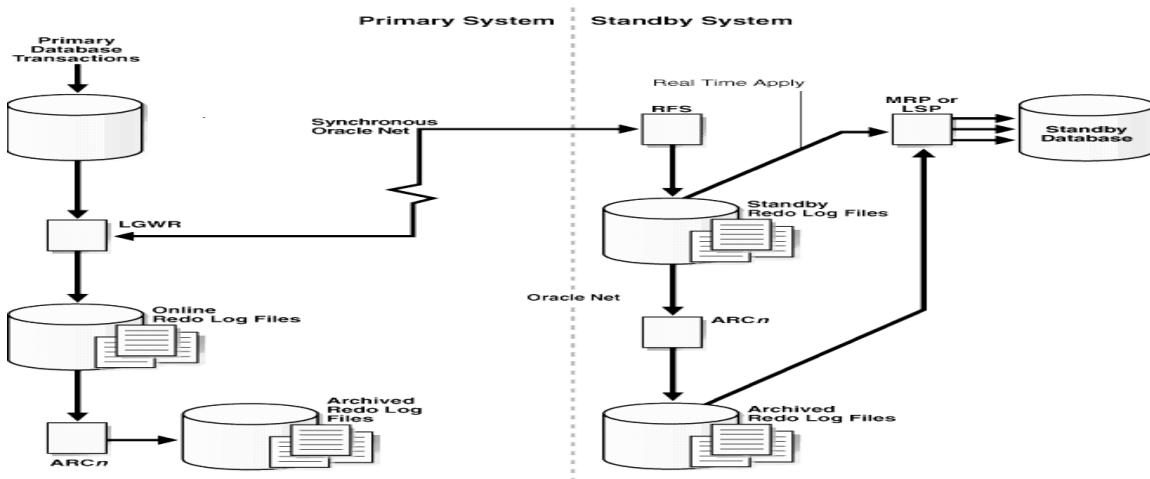
If the real-time apply feature is enabled, apply services can apply redo data as it is received, without waiting for the current standby redo log file to be archived. This results in faster switchover and failover times because the standby redo log files have been applied already to the standby database by the time the failover or switchover begins.

Use the ALTER DATABASE statement to enable the real-time apply feature, as follows:

- For physical standby databases, issue the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE statement.
- For logical standby databases, issue the ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE statement.

Standby redo log files are required to use real-time apply.

Figure shows a Data Guard configuration with a local destination and a standby destination. As the remote file server (RFS) process writes the redo data to standby redo log files on the standby database, apply services can recover redo from standby redo log files as they are being filled.



**(Figure) Applying Redo Data to a Standby Destination Using Real-Time Apply**

### 75.2.1 Specifying a Time Delay for the Application of Archived Redo Log Files

In some cases, you may want to create a time lag between the time when redo data is received from the primary site and when it is applied to the standby database. You can specify a time interval (in minutes) to protect against the application of corrupted or erroneous data to the standby database. When you set a `DELAY` interval, it does not delay the transport of the redo data to the standby database. Instead, the time lag you specify begins when the redo data is completely archived at the standby destination.

**Note:** If you define a delay for a destination that has real-time apply enabled, the delay is ignored.

#### Specifying a Time Delay

You can set a time delay on primary and standby databases using the `DELAY=minutes` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to delay applying archived redo log files to the standby database. By default, there is no time delay. If you specify the `DELAY` attribute without specifying a value, then the default delay interval is 30 minutes.

#### Canceling a Time Delay

You can cancel a specified delay interval as follows:

- For physical standby databases, use the `NODELAY` keyword of the `RECOVER MANAGED STANDBY DATABASE` clause:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY;
```

- For logical standby databases, specify the following SQL statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NODELAY;
```

These commands result in apply services immediately beginning to apply archived redo log files to the standby database, before the time interval expires.

#### Using Flashback Database as an Alternative to Setting a Time Delay

As an alternative to setting an apply delay, you can use Flashback Database to recover from the application of corrupted or erroneous data to the standby database. Flashback Database can quickly and easily flash back a standby database to an arbitrary point in time.

## 75.3 Applying Redo Data to Physical Standby Databases

By default, the redo data is applied from archived redo log files. When performing Redo Apply, a physical standby database can use the real-time apply feature to apply redo directly from the standby redo log files as they are being written by the RFS process. Note that apply services cannot apply redo data to a physical standby database when it is opened in read-only mode.

This section contains the following topics:

- Starting Redo Apply
- Stopping Redo Apply
- Monitoring Redo Apply on Physical Standby Databases

### 75.3.1 Starting Redo Apply

To start apply services on a physical standby database, ensure the physical standby database is started and mounted and then start Redo Apply using the SQL ALTER DATABASE RECOVER MANAGED STANDBY DATABASE statement.

You can specify that Redo Apply runs as a foreground session or as a background process, and enable it with real-time apply.

- To start Redo Apply in the foreground, issue the following SQL statement:
- SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
- If you start a foreground session, control is not returned to the command prompt until recovery is canceled by another session.
- To start Redo Apply in the background, include the DISCONNECT keyword on the SQL statement. For example:
- SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
- This statement starts a detached server process and immediately returns control to the user. While the managed recovery process is performing recovery in the background, the foreground process that issued the RECOVER statement can continue performing other tasks. This does not disconnect the current SQL session.
- To start real-time apply, include the USING CURRENT LOGFILE clause on the SQL statement. For example:

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE;

### 75.3.2 Stopping Redo Apply

To stop Redo Apply, issue the following SQL statement in another window:

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;

## 75.4 Applying Redo Data to Logical Standby Databases

SQL Apply converts the data from the archived redo log or standby redo log in to SQL statements and then executes these SQL statements on the logical standby database. Because the logical standby database remains open, tables that are maintained can be used simultaneously for other tasks such as reporting, summations, and queries.

This section contains the following topics:

- Starting SQL Apply
- Stopping SQL Apply on a Logical Standby Database
- Monitoring SQL Apply on Logical Standby Databases

### 75.4.1 Starting and Stopping SQL Apply

To start SQL Apply, start the logical standby database and issue the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

To start real-time apply on the logical standby database to immediately apply redo data from the standby redo log files on the logical standby database, include the IMMEDIATE keyword as shown in the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

### Stopping SQL Apply

To stop SQL Apply, issue the following statement on the logical standby database:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

When you issue this statement, SQL Apply waits until it has committed all complete transactions that were in the process of being applied. Thus, this command may not stop the SQL Apply processes immediately.

## 76. Redo Transport Services

This chapter describes how to configure and monitor Oracle redo transport services. The following topics are discussed:

- Introduction to Redo Transport Services
- Configuring Redo Transport Services
- Monitoring Redo Transport Services

### 76.1 Introduction to Redo Transport Services

Redo transport services performs the automated transfer of redo data between Oracle databases. The following redo transport destinations are supported:

- Oracle Data Guard standby databases
- Archive Log repository
  - This destination type is used for temporary offsite storage of archived redo log files. An archive log repository consists of an Oracle database instance and a physical standby control file. An archive log repository does not contain datafiles, so it cannot support role transitions.
  - The procedure used to create an archive log repository is identical to the procedure used to create a physical standby database, except for the copying of datafiles.
    - Oracle Streams downstream capture databases
    - Oracle Change Data Capture staging databases

An Oracle database can send redo data to up to nine redo transport destinations. Each redo transport destination is individually configured to receive redo data via one of two redo transport modes:

#### Synchronous

The synchronous redo transport mode transmits redo data synchronously with respect to transaction commitment. A transaction cannot commit until all redo generated by that transaction has been successfully sent to every enabled redo transport destination that uses the synchronous redo transport mode. This transport mode is used by the Maximum Protection and Maximum

#### Asynchronous

The asynchronous redo transport mode transmits redo data asynchronously with respect to transaction commitment. A transaction can commit without waiting for the redo generated by that transaction to be successfully sent to any redo transport destination that uses the asynchronous redo transport mode.

### 76.2 Configuring Redo Transport Services

This section describes how to configure redo transport services. The following topics are discussed:

- Redo Transport Security
- Configuring an Oracle Database to Send Redo Data
- Configuring an Oracle Database to Receive Redo Data

#### 76.2.1 Redo Transport Security

Redo transport uses Oracle Net sessions to transport redo data. These redo transport sessions are authenticated using either the Secure Socket Layer (SSL) protocol or a remote login password file.

## Redo Transport Authentication Using SSL

Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections. SSL uses RSA public key cryptography and symmetric key cryptography to provide authentication, encryption, and data integrity. SSL is automatically used for redo transport authentication between two Oracle databases if:

- The databases are members of the same Oracle Internet Directory (OID) enterprise domain and that domain allows the use of current user database links.
- The LOG\_ARCHIVE\_DEST\_n, FAL\_SERVER, and FAL\_CLIENT database initialization parameters that correspond to the databases use Oracle Net connect descriptors configured for SSL.
- Each database has an Oracle wallet or a supported hardware security module that contains a user certificate with a distinguished name (DN) that matches the DN in the OID entry for the database.

## Redo Transport Authentication Using a Password File

If the SSL authentication requirements are not met, each database must use a remote login password file. In a Data Guard configuration, all physical and snapshot standby databases must use a copy of the password file from the primary database, and that copy must be refreshed whenever the SYSOPER or SYSDBA privilege is granted or revoked, and after the password of any user with these privileges is changed.

When a password file is used for redo transport authentication, the password of the user account used for redo transport authentication is compared between the database initiating a redo transport session and the target database. The password must be the same at both databases to create a redo transport session.

By default, the password of the SYS user is used to authenticate redo transport sessions when a password file is used. The REDO\_TRANSPORT\_USER database initialization parameter can be used to select a different user password for redo transport authentication by setting this parameter to the name of any user who has been granted the SYSOPER privilege. For administrative ease, Oracle recommends that the REDO\_TRANSPORT\_USER parameter be set to the same value on the redo source database and at each redo transport destination.

### 76.2.2 Configuring an Oracle Database to Send Redo Data

This section describes how to configure an Oracle database to send redo data to a redo transport destination.

The LOG\_ARCHIVE\_DEST\_n database initialization parameter (where n is an integer from 1 to 10) is used to specify the location of a local archive redo log or to specify a redo transport destination. This section describes the latter use of this parameter.

There is a LOG\_ARCHIVE\_DEST\_STATE\_n database initialization parameter (where n is an integer from 1 to 10) that corresponds to each LOG\_ARCHIVE\_DEST\_n parameter. This parameter is used to enable or disable the corresponding redo destination. Table 6-1 shows the valid values that can be assigned to this parameter.

Table LOG\_ARCHIVE\_DEST\_STATE\_n Initialization Parameter Values

| Value     | Description                                                                                |
|-----------|--------------------------------------------------------------------------------------------|
| ENABLE    | Redo transport services can transmit redo data to this destination. This is the default.   |
| DEFER     | Redo transport services will not transmit redo data to this destination.                   |
| ALTERNATE | This destination will become enabled if communication to its associated destination fails. |

A redo transport destination is configured by setting the LOG\_ARCHIVE\_DEST\_n parameter to a character string that includes one or more attributes. This section briefly describes the most commonly used attributes.

- The SERVICE attribute, which is a mandatory attribute for a redo transport destination, must be the first attribute specified in the attribute list. The SERVICE attribute is used to specify the Oracle Net service name used to connect to a redo transport destination.
- The SYNC attribute is used to specify that the synchronous redo transport mode be used to send redo data to a redo transport destination.
- The ASYNC attribute is used to specify that the asynchronous redo transport mode be used to send redo data to a redo transport destination. The asynchronous redo transport mode will be used if neither the SYNC nor the ASYNC attribute is specified.
- The NET\_TIMEOUT attribute is used to specify how long the LGWR process will block waiting for an acknowledgement that redo data has been successfully received by a destination that uses the synchronous redo transport mode. If an acknowledgement is not received within NET\_TIMEOUT seconds, the redo transport connection is terminated and an error is logged.
- Oracle recommends that the NET\_TIMEOUT attribute be specified whenever the synchronous redo transport mode is used, so that the maximum duration of a redo source database stall caused by a redo transport fault can be precisely controlled.
- The AFFIRM attribute is used to specify that redo received from a redo source database is not acknowledged until it has been written to the standby redo log. The NOAFFIRM attribute is used to specify that received redo is acknowledged without waiting for received redo to be written to the standby redo log.
- The DB\_UNIQUE\_NAME attribute is used to specify the DB\_UNIQUE\_NAME of a redo transport destination. The DB\_UNIQUE\_NAME attribute must be specified if the LOG\_ARCHIVE\_CONFIG database initialization parameter has been defined and its value includes a DG\_CONFIG list.
- If the DB\_UNIQUE\_NAME attribute is specified, its value must match one of the DB\_UNIQUE\_NAME values in the DG\_CONFIG list. It must also match the value of the DB\_UNIQUE\_NAME database initialization parameter at the redo transport destination. If either match fails, an error is logged and redo transport will not be possible to that destination.
- The VALID\_FOR attribute is used to specify when redo transport services transmits redo data to a redo transport destination. Oracle recommends that the VALID\_FOR attribute be specified for each redo transport destination at every site in a Data Guard configuration so that redo transport services will continue to send redo data to all standby databases after a role transition, regardless of which standby database assumes the primary role.
- The REOPEN attribute is used to specify the minimum number of seconds between automatic reconnect attempts to a redo transport destination that is inactive because of a previous error.
- The COMPRESSION attribute is used to specify that redo data is transmitted to a redo transport destination in compressed form when resolving redo data gaps. Redo transport compression can significantly improve redo gap resolution time when network links with low bandwidth and high latency are used for redo transport.

The following example uses all of the LOG\_ARCHIVE\_DEST\_n attributes described in this section. Two redo transport destinations are defined and enabled. The first destination uses the asynchronous redo transport mode. The second destination uses the synchronous redo transport mode with a 30-second timeout. A DB\_UNIQUE\_NAME has been specified for both destinations, as has the use of compression when resolving redo gaps. If a redo transport fault occurs at either destination, redo transport will attempt to reconnect to that destination, but not more frequently than once every 52 seconds.

```

DB_UNIQUE_NAME=BOSTON
LOG_ARCHIVE_CONFIG='DG_CONFIG=(BOSTON,CHICAGO,DENVER)'
LOG_ARCHIVE_DEST_2='SERVICE=CHICAGO
ASYNC
NOAFFIRM
VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)
REOPEN=52
COMPRESSION=ENABLE
DB_UNIQUE_NAME=CHICAGO'
LOG_ARCHIVE_DEST_STATE_2='ENABLE'
LOG_ARCHIVE_DEST_3='SERVICE=DENVER
SYNC
AFFIRM
NET_TIMEOUT=30
VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)
REOPEN=52
COMPRESSION=ENABLE
DB_UNIQUE_NAME=DENVER'
LOG_ARCHIVE_DEST_STATE_3='ENABLE'

```

### Viewing Attributes With V\$ARCHIVE\_DEST

The V\$ARCHIVE\_DEST view can be queried to see the current settings and status for each redo transport destination.

### 76.2.3 Configuring an Oracle Database to Receive Redo Data

This section describes how to configure a redo transport destination to receive and to archive redo data from a redo source database. The following topics are discussed:

Creating and Managing a Standby Redo Log

Configuring Standby Redo Log Archival

#### Creating and Managing a Standby Redo Log

The synchronous and asynchronous redo transport modes require that a redo transport destination have a standby redo log. A standby redo log is used to store redo received from another Oracle database. Standby redo logs are structurally identical to redo logs, and are created and managed using the same SQL statements used to create and manage redo logs.

Redo received from another Oracle database via redo transport is written to the current standby redo log group by a RFS background process. When a log switch occurs on the redo source database, incoming redo is then written to the next standby redo log group, and the previously used standby redo log group is archived by an ARCh background process.

The process of sequentially filling and then archiving redo log file groups at a redo source database is mirrored at each redo transport destination by the sequential filling and archiving of standby redo log groups.

Each standby redo log file must be at least as large as the largest redo log file in the redo log of the redo source database. For administrative ease, Oracle recommends that all redo log files in the redo log at the redo source database and the standby redo log at a redo transport destination be of the same size.

The standby redo log must have at least one more redo log group than the redo log on the redo source database.

Perform the following query on a redo source database to determine the size of each log file and the number of log groups in the redo log:

```
SQL> SELECT GROUP#, BYTES FROM V$LOG;
```

Perform the following query on a redo destination database to determine the size of each log file and the number of log groups in the standby redo log:

```
SQL> SELECT GROUP#, BYTES FROM V$STANDBY_LOG;
```

Oracle recommends that a standby redo log be created on the primary database in a Data Guard configuration so that it is immediately ready to receive redo data following a switchover to the standby role.

The ALTER DATABASE ADD STANDBY LOGFILE SQL statement is used to create a standby redo log and to add standby redo log groups to an existing standby redo log.

For example, assume that the redo log on the redo source database has two redo log groups and that each of those contain one 600 MB redo log file. In this case, the standby redo log should have at least 3 standby redo log groups to satisfy the requirement that a standby redo log must have at least one more redo log group than the redo log at the redo source database.

The following SQL statements might be used to create a standby redo log that is appropriate for the previous scenario:

```

ALTER DATABASE ADD STANDBY LOGFILE
 ('/oracle/dbs/slog1.rdo') SIZE 600M;

ALTER DATABASE ADD STANDBY LOGFILE
 ('/oracle/dbs/slog2.rdo') SIZE 600M;

ALTER DATABASE ADD STANDBY LOGFILE
 ('/oracle/dbs/slog3.rdo') SIZE 600M;

```

**Caution:** Whenever a redo log group is added to the primary database in an Oracle Data Guard configuration, a standby redo log group must also be added to the standby redo log at each standby database in the configuration that uses the synchronous redo transport mode. If this is not done, a primary database that is running in the maximum protection data protection mode may shut down, and a primary database that is running in the maximum availability data protection mode may shift to the maximum performance data protection mode.

### Configuring Standby Redo Log Archival

This section describes how to configure standby redo log archival.

#### Standby Redo Log Archival to a Flash Recovery Area

Take the following steps to set up standby redo log archival to a flash recovery area:

1. Set the LOCATION attribute of a LOG\_ARCHIVE\_DEST\_n parameter to USE\_DB\_RECOVERY\_FILE\_DEST.
2. Set the VALID\_FOR attribute of the same LOG\_ARCHIVE\_DEST\_n parameter to a value that allows standby redo log archival.

The following are some sample parameter values that might be used to configure a physical standby database to archive its standby redo log to the flash recovery area:

```

LOG_ARCHIVE_DEST_2 = 'LOCATION=USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(STANDBY_LOGFILE,STANDBY_ROLE)'
LOG_ARCHIVE_DEST_STATE_2=ENABLE

```

Oracle recommends the use of a flash recovery area, because it simplifies the management of archived redo log files.

#### Standby Redo Log Archival to a Local File System Location

Take the following steps to set up standby redo log archival to a local file system location:

- Set the LOCATION attribute of a LOG\_ARCHIVE\_DEST\_n parameter to a valid pathname.  
 Set the VALID\_FOR attribute of the same LOG\_ARCHIVE\_DEST\_n parameter to a value that allows standby redo log archival.

The following are some sample parameter values that might be used to configure a physical standby database to archive its standby redo log to a local file system location:

```

LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive
VALID_FOR=(STANDBY_LOGFILE,STANDBY_ROLE)'
LOG_ARCHIVE_DEST_STATE_2=ENABLE

```

## 76.3 Monitoring Redo Transport Services

This section discusses the following topics:

1. Monitoring Redo Transport Status
2. Monitoring Synchronous Redo Transport Response Time
3. Redo Gap Detection and Resolution
4. Redo Transport Services Wait Events

### 76.3.1 Monitoring Redo Transport Status

This section describes the steps used to monitor redo transport status on a redo source database.

**Step 1:** Determine the most recently archived redo log file.

Perform the following query on the redo source database to determine the most recently archived sequence number for each thread:

```
SQL> SELECT MAX(SEQUENCE#), THREAD# FROM V$ARCHIVED_LOG GROUP BY THREAD#;
```

**Step 2:** Determine the most recently archived redo log file at each redo transport destination.

Perform the following query on the redo source database to determine the most recently archived redo log file at each redo transport destination:

```
SQL> SELECT DESTINATION, STATUS, ARCHIVED_THREAD#, ARCHIVED_SEQ#
 2> FROM V$ARCHIVE_DEST_STATUS
 3> WHERE STATUS <> 'DEFERRED' AND STATUS <> 'INACTIVE';
```

| DESTINATION        | STATUS | ARCHIVED_THREAD# | ARCHIVED_SEQ# |
|--------------------|--------|------------------|---------------|
| /private1/prmy/1ad | VALID  | 1                | 947           |
| standby1           | VALID  | 1                | 947           |

The most recently archived redo log file should be the same for each destination. If it is not, a status other than VALID may identify an error encountered during the archival operation to that destination.

**Step 3:** Find out if archived redo log files have been received at a redo transport destination.

A query can be performed at a redo source database to find out if an archived redo log file has been received at a particular redo transport destination. Each destination has an ID number associated with it. You can query the DEST\_ID column of the V\$ARCHIVE\_DEST view on a database to identify each destination's ID number.

Assume that destination 1 points to the local archived redo log and that destination 2 points to a redo transport destination. Perform the following query at the redo source database to find out if any log files are missing at the redo transport destination:

```
SQL> SELECT LOCAL.THREAD#, LOCAL.SEQUENCE# FROM
 2> (SELECT THREAD#, SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=1)
 3> LOCAL WHERE
 4> LOCAL.SEQUENCE# NOT IN
 5> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND
 6> THREAD# = LOCAL.THREAD#);
```

| THREAD# | SEQUENCE# |
|---------|-----------|
| 1       | 12        |
| 1       | 13        |
| 1       | 14        |

**Step 4:** Trace the progression of redo transmitted to a redo transport destination.

Set the LOG\_ARCHIVE\_TRACE database initialization parameter at a redo source database and at each redo transport destination to trace redo transport progress.

### 76.3.2 Monitoring Synchronous Redo Transport Response Time

- The V\$REDO\_DEST\_RESP\_HISTOGRAM view contains response time data for each redo transport destination. This response time data is maintained for redo transport messages sent via the synchronous redo transport mode.
- The data for each destination consists of a series of rows, with one row for each response time. To simplify record keeping, response times are rounded up to the nearest whole second for response times less than 300 seconds. Response times greater than 300 seconds are round up to 520, 1200, 2400, 4740, or 9520 seconds.
- Each row contains four columns: FREQUENCY, DURATION, DEST\_ID, and TIME.
- The FREQUENCY column contains the number of times that a given response time has been observed. The DURATION column corresponds to the response time. The DEST\_ID column identifies the destination. The TIME column contains a timestamp taken when the row was last updated.
- The response time data in this view is useful for identifying synchronous redo transport mode performance issues that can affect transaction throughput on a redo source database. It is also useful for tuning the NET\_TIMEOUT attribute.
- The next three examples show example queries for destination 2, which corresponds to the LOG\_ARCHIVE\_DEST\_2 parameter. To display response time data for a different destination, simply change the DEST\_ID in the query.

Perform the following query on a redo source database to display the response time histogram for destination 2:

```
SQL> SELECT FREQUENCY, DURATION FROM
2> V$REDO_DEST_RESP_HISTOGRAM WHERE DEST_ID=2 AND FREQUENCY>1;
```

Perform the following query on a redo source database to display the fastest response time for destination 2:

```
SQL> SELECT max(DURATION) FROM V$REDO_DEST_RESP_HISTOGRAM
2> WHERE DEST_ID=2 AND FREQUENCY>1;
```

Perform the following query on a redo source database to display the slowest response time for destination 2:

```
SQL> SELECT min (DURATION) FROM V$REDO_DEST_RESP_HISTOGRAM
2> WHERE DEST_ID=2 AND FREQUENCY>1;
```

#### Note:

The highest observed response time for a destination cannot exceed the highest specified NET\_TIMEOUT value specified for that destination, because synchronous redo transport mode sessions are terminated if a redo transport destination does not respond to a redo transport message within NET\_TIMEOUT seconds.

### 76.3.3 Redo Gap Detection and Resolution

A redo gap occurs whenever redo transmission is interrupted. When redo transmission resumes, redo transport services automatically detects the redo gap and resolves it by sending the missing redo to the destination.

The time needed to resolve a redo gap is directly proportional to the size of the gap and inversely proportional to the effective throughput of the network link between the redo source database and the redo transport destination. Redo transport services has two options that may reduce redo gap resolution time when low performance network links are used:

#### Redo Transport Compression

The COMPRESSION attribute of the LOG\_ARCHIVE\_DEST\_n parameter can be used to specify that redo transport compression be used to compress the redo sent to resolve a redo gap.

## Parallel Redo Transport Network Sessions

The MAX\_CONNECTIONS attribute of the LOG\_ARCHIVE\_DEST\_n parameter can be used to specify that more than one network session be used to send the redo needed to resolve a redo gap.

## 76.4 Manual Gap Resolution

In some situations, gap resolution cannot be performed automatically and it must be performed manually. For example, redo gap resolution must be performed manually on a logical standby database if the primary database is unavailable.

Perform the following query at the physical standby database to determine if there is redo gap on a physical standby database:

```
SQL> SELECT * FROM V$ARCHIVE_GAP;
 THREAD# LOW_SEQUENCE# HIGH_SEQUENCE#
----- -----
 1 7 10
```

The output from the previous example indicates that the physical standby database is currently missing log files from sequence 7 to sequence 10 for thread 1.

Perform the following query on the primary database to locate the archived redo log files on the primary database (assuming the local archive destination on the primary database is LOG\_ARCHIVE\_DEST\_1):

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND
2> DEST_ID=1 AND SEQUENCE# BETWEEN 7 AND 10;
 NAME

 /primary/thread1_dest/arcr_1_7.arc
 /primary/thread1_dest/arcr_1_8.arc
 /primary/thread1_dest/arcr_1_15.arc
```

### Note:

This query may return consecutive sequences for a given thread. In that case, there is no actual gap, but the associated thread was disabled and enabled within the time period of generating these two archived logs. The query also does not identify the gap that may exist at the tail end for a given thread. For instance, if the primary database has generated archived logs up to sequence 100 for thread 1, and the latest archived log that the logical standby database has received for the given thread is the one associated with sequence 72, this query will not return any rows, although we have a gap for the archived logs associated with sequences 78 to 100.

Copy these log files to the physical standby database and register them using the ALTER DATABASE REGISTER LOGFILE. For example:

```
SQL> ALTER DATABASE REGISTER LOGFILE
 '/physical_standby1/thread1_dest/arcr_1_7.arc';
SQL> ALTER DATABASE REGISTER LOGFILE
 '/physical_standby1/thread1_dest/arcr_1_8.arc';
SQL> ALTER DATABASE REGISTER LOGFILE
 '/physical_standby1/thread1_dest/arcr_1_15.arc';
```

### Note:

The V\$ARCHIVE\_GAP view on a physical standby database only returns the gap that is currently blocking Redo Apply from continuing. After resolving the gap, query the V\$ARCHIVE\_GAP view again on the physical standby database to determine if there is another gap sequence. Repeat this process until there are no more gaps.

To determine if there is a redo gap on a logical standby database, query the DBA\_LOGSTDBY\_LOG view on the logical standby database. For example, the following query indicates there is a gap in the sequence of archived redo log files because it displays two files for THREAD 1 on the logical standby database. (If there are no gaps, the query will show only one file for each thread.) The output shows that the highest registered file is sequence number 10, but there is a gap at the file shown as sequence number 6:

```
SQL> COLUMN FILE_NAME FORMAT a55
SQL> SELECT THREAD#, SEQUENCE#, FILE_NAME FROM
DBA_LOGSTDBY_LOG L
 2> WHERE NEXT_CHANGE# NOT IN
 3> (SELECT FIRST_CHANGE# FROM DBA_LOGSTDBY_LOG WHERE L.THREAD# =
THREAD#)
 4> ORDER BY THREAD#, SEQUENCE#;

THREADE# SEQUENCE# FILE_NAME
----- -----
1 6 /disk1/oracle/dbs/log-1292874008_6.arc
1 10 /disk1/oracle/dbs/log-1292874008_8.arc
```

Copy the missing log files, with sequence numbers 7, 8, and 9, to the logical standby system and register them using the ALTER DATABASE REGISTER LOGICAL LOGFILE statement. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-
1292874008_7.arc';
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-
1292874008_8.arc';
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-
1292874008_15.arc';
```

**Note:**

A query based on the DBA\_LOGSTDBY\_LOG view on a logical standby database, as specified above, only returns the gap that is currently blocking SQL Apply from continuing. After resolving the gap, query the DBA\_LOGSTDBY\_LOG view again on the logical standby database to determine if there is another gap sequence. Repeat this process until there are no more gaps.

### 76.4.1 Redo Transport Services Wait Events

Table lists several of the Oracle wait events used to track redo transport wait time on a redo source database. These wait events are found in the V\$SYSTEM\_EVENT dynamic performance view.

**Table Redo Transport Wait Events**

| Wait Event          | Description                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------|
| LNS wait on ATTACH  | Total time spent waiting for redo transport sessions to be established to all ASYNC and SYNC redo transport destinations   |
| LNS wait on SENDREQ | Total time spent waiting for redo data to be written to all ASYNC and SYNC redo transport destinations                     |
| LNS wait on DETACH  | Total time spent waiting for redo transport connections to be terminated to all ASYNC and SYNC redo transport destinations |

## 77. Logical Standby Dataguard

### 77.1 Introduction

Oracle started shipping Standby Database features with the release of Oracle 8.0.4. Standby Database features were enhanced with the help of user feedback in subsequent releases of Oracle. Although there are several ways to set up highly available (HA) databases, including Oracle 10g features like RAC, Veritas Cluster software (VCS), HP data guard and Sun cluster (SC) etc., these clustered systems avoid having single points-of-failure by having software and hardware redundancy.

In the case of failure, tasks being performed by the failed component are taken over by the backup component. Although these redundant features are good for high availability and scalability, they do not protect from user mistakes, data corruptions and other disasters that may destroy the database itself. That is where Oracle 10g Data Guard and Standby Database features protect your mission critical databases.

If you are using Oracle Enterprise edition, both Data Guard broker and Standby Database features are included at no cost. The term "Data Guard" is synonymous to Standby Database in many ways, as Oracle renamed the Standby Database feature to "Oracle Data Guard" in Oracle release 15.0.1.

DBAs have the option to set up two different types of standby databases. They are a physical standby database and a logical standby database. Physical standby databases are physically identical to primary databases, meaning all objects in the primary database are the same as in standby database. Logical Standby Databases are logically identical to primary databases although the physical organization and structure of the data can be different.

Physical Standby databases are traditionally standby databases, identical to primary databases on a block for block basis. It is updated by performing media recovery; imagine a DBA sitting in the office and recovering the database constantly. Logical Standby Databases are updated using SQL statements. The advantage of a logical standby database is that it can be used for recovery and reporting simultaneously. The logical standby feature as it can be used for my disaster recovery project as well as it can be used by data warehouse users for their reporting purpose.

### 77.2 When to choose Logical Standby database

**Reporting:** Synchronization of the logical standby database with the primary database is done using logminer technology, which transforms standard archived redo logs into SQL statements and applies them to the logical stand by database. Therefore, the logical standby database must remain open and the tables that are maintained can be used simultaneously for reporting.

**System Resources:** Besides the efficient utilization of system resources, reporting tasks, summations and queries can be optimized by creating additional indexes and materialized views, since both primary and logical standby database can have a different physical lay out by protecting switchover and failover for the primary database.

### 77.3 Prerequisite Conditions for creating a Logical Standby Database:

1. Determine if the primary database contains tables and datatypes that were not supported by a logical stand by database. If the primary database contains tables that were unsupported, log apply services will exclude the tables applying to the logical stand by database.

```
SQL> Select * from dba_logstdby_unsupported;
OWNER TABLE_NAME COLUMN_NAME
DATA_TYPE


```

| OWNER                       | TABLE_NAME           | COLUMN_NAME |
|-----------------------------|----------------------|-------------|
| WMSYS                       | WM\$UDTRIG_INFO      | TRIG_CODE   |
| LONG                        |                      |             |
| WMSYS                       | WM\$VERSIONED_TABLES | UNDO_CODE   |
| WM\$ED_UNDO_CODE_TABLE_TYPE |                      |             |
| 2 rows selected.            |                      |             |

2. To maintain data in a logical stand by database, SQL Apply operations must be able to identify the columns that uniquely identify each row that has been updated in the primary database. Tables that do not have primary keys or non-null unique indexes are identified by enabling supplemental logging.

```
SQL> select owner, table_name, bad_column from dba_logstdby_not_unique;
OWNER TABLE_NAME B
VCSUSER VCS N
```

Bad column 'N' indicates that the table contains enough column information to maintain the table in the logical standby database, whereas 'Y' indicates the table column is defined using an unbounded data type, such as LONG.

Add a primary key to the tables that do not have to improve performance. If the table has a primary key or a unique index with a non-null column, the amount of information added to the redo log is minimal.

3. Ensure that Primary database is in archivelog mode.

```
SQL> archive log list
Database log mode Archive Mode
Automatic archival Enabled
Archive destination /opt/app/oracle/admin/myDB/arch
Oldest online log sequence 345
Next log sequence to archive 347
Current log sequence 347
```

4. Ensure supplemental logging is enabled and log parallelism is enabled on the primary database. Supplemental logging must be enabled because the logical standby database cannot use archived redo logs that contain both supplemental log data and no supplemental log data.

```
SQL> select supplemental_log_data_pk, supplemental_log_data_ui from
v$database;
SUP SUP
--- ---
YES YES
```

If the supplemental logging is not enabled, execute the following

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY,UNIQUE INDEX)
COLUMNS;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

If log parallelism is not enabled, execute the following:

```
SQL> ALTER SYSTEM SET LOG_PARALLELISM=1 SCOPE=BOTH;
```

Start Resource manager if you plan to create a logical standby database using hot backup. If you do not have a resource\_manager plan, you can use one of the system defined plans and restart the primary database to make sure it is using the defined plan.

```
SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN=SYSTEM_PLAN SCOPE=BOTH;
SQL> SHUTDOWN
SQL> STARTUP
```

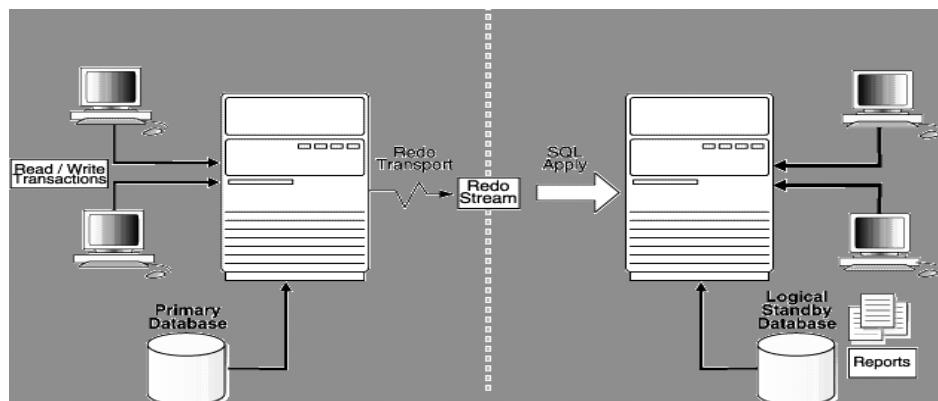
### 77.3.1 Improvements in Oracle Data Guard in Oracle 10gr2:

Automatic Deletion of applied archive logs: Once primary database Archived logs are applied to a Logical Standby Database, they are deleted automatically without DBA intervention. This makes it easier to maintain both primary and logical standby databases. Physical standby databases have had this functionality since Oracle 10g Release 1, by using Flash Recovery Area option.

No downtime required: The primary database is no longer required to shutdown or be put in QUIESCING state, as we can create the logical standby database from a hot backup of the primary database just like the physical standby database.

Online upgrades: A lot of DBAs have dreamed about this for long time: just like IBM's DB2 or Microsoft SQL Server, the DBA no longer required to shutdown the primary database to upgrade from Oracle 10g release 2 with Data Guard option. First, upgrade the logical standby database to the next release, test and validate the upgrade, do a role reversal by switching over to the upgraded database, and then finally upgrade the old primary database. New Datatypes Supported: I always used to hesitate whenever I thought of logical standby databases, as some of my databases never meet the pre-requisite conditions. In 10g relase2, Oracle supports most of the datatypes, such as NCLOB, LONG, LONGRAW, BINARY\_FLOAT, BINARY\_DOUBLE, IOTs.

Conclusion: SQL Apply with Logical Standby Database is a viable option for customers who need to implement a disaster recovery solution or maximum/high availability solution and use the same resources for reporting and decision support operations. The success in creating a Logical Standby Database depends a lot on how the tasks are executed and on the version is being used. It is very important, before starting the creation of a Logical Standby Database, to make sure that all the Initialization Parameters are set correctly, that all the steps are followed in the correct order and the appropriate parameters are used. If everything is done properly then you



should be able to do a clean configuration of the Logical Standby Database in the first go.

## 78. Cloning Oracle Database

This tells us how to create an Oracle clone database instance called TEST from the current PPRD database. It lists the steps for this procedure along with the comments of each command.

The datafiles are in the /u03/oradata/TEST and /u03/oradata/PPRD directories respectively for this example. The commands marked with an asterisk (\*) are only needed if this is the first time we have done this clone, and are not needed for subsequent e-cloning. (Note: "\$" means we're at the UNIX prompt, "SQL>" means we're in SQLPLUS as a DBA user ID).

**WARNING:** This cloning procedure may not work with some of the Oracle tools (such as the RMAN Recovery Manager's recovery catalog, because it uses an internal database ID instead of just the Oracle SID to identify the database). So, if we want to use those tools, we will need to find some other way to clone our Oracle database.

### Login as user oracle

```
$. oraenv
Set the oracle SID to the name of the instance to copy (PPRD here).
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
Generates a textual control file to edit for the new instance later.
SQL> SELECT value FROM V$PARAMETER WHERE name LIKE '%user_dump%';
Shows the directory containing the generated textual control file.
SQL> SELECT name FROM V$DATAFILE;
Shows the names of the datafiles to copy to the new database instance's
directory(s).
SQL> SELECT name FROM V$CONTROLFILE;
Shows the names of the control files to copy to the new directory(s).
SQL> SELECT member FROM V$LOGFILE;
Shows the names of the redo log files to copy to the new directory(s).
SQL> CONNECT / AS SYSDBA
SQL> SHUTDOWN IMMEDIATE
Shuts down the current instance to copy (PPRD here).
$ cd /u03/oradata
* $ mkdir TEST
$ cd TEST
$ cp -p /u03/oradata/PPRD/* .
$ ls *PPRD* | sed "s/^(.*\PPRD\(.*)/mv \1PPRD\2 \1TEST\2/" >rename.shl
$ sh rename.shl
```

These commands copy PPRD's datafiles to a new TEST directory and rename the files in the TEST directory to match the Oracle SID name. Similar groups of commands will be used for other existing PPRD directories with control files and redo log files. Also, create a directory for TEST's archive logs, if we want archiving turned on for TEST, but don't copy any of PPRD's archive logs into it (since we'll need to do a RESETLOGS on the startup of the new instance anyway).

```
* $ cd $ORACLE_HOME/dbs
* $ cp initPPRD.ora initTEST.ora
* $ vi initTEST.ora
```

Create the init.ora file for TEST and change all references to PPRD into TEST (vi command is ":1,\$s/PPRD/TEST/g"). Also, if the new TEST's files are on a different disk volume, change the volume names, as required, to match where we copied the files.

```
* $ mkdir /u00/oracle/admin/TEST
* $ mkdir /u00/oracle/admin/TEST/bdump
* $ mkdir /u00/oracle/admin/TEST/cdump
* $ mkdir /u00/oracle/admin/TEST/udump
```

These commands create the dump directories for TEST that were given in the initTEST.ora file

```
$ cd /u03/oradata/TEST
$ ls -ltr /u00/oracle/admin/PPRD/udump
$ cp /u00/oracle/admin/PPRD/udump/ora_28566.trc ctrl.sql
```

These commands find the textual control file created at the beginning (should be the last .trc file listed) and copy it to ctrl.sql.

```
$ vi ctrl.sql
```

Remove all the lines before the STARTUP NOMOUNT command (usually, the first 20 (or 22) lines, for which the vi command is ":1,20d"). Edit it to match the new TEST datafile names and Oracle SID name.

If the directories are similar, this could just mean changing all of the PPRD references into TEST (vi command is ":1,\$s/PPRD/TEST/g"). Also, change NORESETLOGS to RESETLOGS in the create controlfile command, resulting in the following line (use NOARCHIVELOG if we don't want archiving turned on in the new instance):

```
CREATE CONTROLFILE REUSE SET DATABASE "TEST" RESETLOGS ARCHIVELOG
```

Comment out the RECOVER DATABASE command (put # in front of it), and change the last line to:

```
ALTER DATABASE OPEN RESETLOGS;
* $ vi /etc/oratab
```

Add a line for the new instance, usually by copying the last line (vi command is "G:.co.") and changing the name in that copied line to the 4-character name of the new instance (vi command here is "RTEST<esc>"). Also, change the last character in that copied line to Y to have the database start up automatically during dbstart, or N for only manual startups. (Adding: TEST:/u00/oracle/product/v901:Y)

```
$. oraenv
Set the oracle SID to the name of the new instance (TEST here).
SQL> connect / as sysdba
SQL> @ctrl.sql
```

Creates the new control files pointing to the new datafile and log file locations, and opens the new database.

Note: If we get "ORA-1153 error: database name in file header does not match given name" when we try to run the CREATE CONTROLFILE, try this instead: don't copy the control files to the new directories (or, delete the control files from the new directories), and, edit ctrl.sql to take out the REUSE option in the CREATE CONTROLFILE command, then, try rerunning ctrl.sql.

```
SQL> SELECT * FROM GLOBAL_NAME;
```

Shows the original global name, such as PPRD.WORLD

```
SQL> UPDATE GLOBAL_NAME SET GLOBAL_NAME = "TEST.WORLD";
```

Changes the global name so that doing a "create database link" to access a remote database doesn't give us a "loopback" error. (Only do this if needed, since we don't know if this "world" change would adversely affect anything else in Oracle.)

```
$. oraenv
```

Set the Oracle SID to the name of the original instance (PPRD here).

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP
```

Restarts that original instance.

```
* $ lsnrctl status
```

Shows the pathname of the Listener Parameter File (listener.ora).

```
* $ cat /u00/oracle/product/v901/network/admin/listener.ora
```

Edit the listener.ora file to copy the PPRD lines and change the copy to match TEST (don't change any of the spacing!), giving:

```

(SID_DESC=
(SID_NAME=TEST)
(ORACLE_HOME=/u00/oracle/product/v663)
)
* $ lsnrctl stop
* $ lsnrctl start

```

The TEST instance has now been added to the SQL\*NET Listener. On the client network (such as Novell), we will need to edit our tnsnames.ora file in the orawin\network\admin directory to copy the PPRD instance's lines and change the copy to match TEST, similar to:

```

unix_test =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS =
(PROTOCOL = TCP)
(Host = myhost.domain.com)
(Port = 1521)
)
)
(CONNECT_DATA = (SID = TEST)
)
)
* $ su - jobsub
* $ cat start_jobsub.shl

```

These lines log into jobsub (if we are allowed to use "su"; otherwise, just login to jobsub, etc.) and edit the jobsub startup script to include the TEST instance (copy PPRD's lines and change to match TEST):

```

ORACLE_SID=TEST; export ORACLE_SID; . oraenv
echo "==== Starting jobsubmission for $ORACLE_SID..... "
nohup sh $BANNER_LINKS/gurjobs.shl > gurjobsTEST.out 2>&1 &
$ su - jobsub
$ kill -9 -1
These lines kill the current jobsub processes for all instances.
$ su - jobsub
$ start_jobsub.shl

```

Starts jobsub for all instances, including for TEST. (Or, instead of killing jobsub and restarting it, we could have just entered ". oraenv" to set the TEST instance, and the "nohup" line to start jobsub for it.)

If we want to change the internal database ID of the cloned copy so that we can use utilities such as RMAN on that copy which requires unique database ID's, we can use the Oracle 9i "nid" (new ID) utility to generate a new database ID, as shown below. Note that we haven't tried this, yet, but, we can give it a try if we need to use RMAN on a copy-datafile clone.

```

$. oraenv
Set the oracle SID to the name of the new instance (TEST here).
SQL> CONNECT / AS SYSDBA
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> HOST
$ nid SYS/<syspassword>
Answer the prompt with Y
$ Exit
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP MOUNT
SQL> ALTER DATABASE OPEN RESETLOGS;
SQL> SHUTDOWN IMMEDIATE
SQL> STARTUP
SQL> EXIT

```

## 79. Oracle Data Guard Broker

This chapter describes the Oracle Data Guard broker, its architecture and components, and how it automates the creation, control, and monitoring of a Data Guard configuration. It contains the following topics:

1. Overview of Oracle Data Guard and the Broker
2. Benefits of Data Guard Broker
3. Data Guard Broker Management Model
4. Data Guard Broker Components
5. Data Guard Broker User Interfaces
6. Data Guard Monitor

See Oracle Data Guard Concepts and Administration for the definition of a Data Guard configuration and for complete information about Oracle Data Guard concepts and terminology.

### 79.1 Overview of Oracle Data Guard and the Broker

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data. Data Guard provides a comprehensive set of services that create, maintain, manage, and monitor one or more standby databases to enable production Oracle databases to survive disasters and data corruptions. Data Guard maintains these standby databases as transactionally consistent copies of the primary database.

If the primary database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby database to the production role, thus minimizing the downtime associated with the outage. Data Guard can be used with traditional backup, recovery, and cluster techniques, as well as the Flashback Database feature to provide a high level of data protection and data availability.

#### 79.1.1 Data Guard Configurations and Broker Configurations

A Data Guard configuration consists of one primary database and up to nine standby databases. The databases in a Data Guard configuration are connected by Oracle Net and may be dispersed geographically. There are no restrictions on where the databases are located as long as they can communicate with each other. For example, you can have a standby database on the same system as the primary database, along with two standby databases on another system.

The Data Guard broker logically groups these primary and standby databases into a broker configuration that allows the broker to manage and monitor them together as an integrated unit. You can manage a broker configuration using either the Oracle Enterprise Manager graphical user interface or the Data Guard command-line interface.

#### 79.1.2 Oracle Data Guard Broker

The Oracle Data Guard broker is a distributed management framework that automates and centralizes the creation, maintenance, and monitoring of Data Guard configurations. The following list describes some of the operations the broker automates and simplifies:

- o Creating Data Guard configurations that incorporate a primary database, a new or existing (physical, logical, or snapshot) standby database, redo transport services, and log apply services, where any of the databases could be Oracle Real Application Clusters (RAC) databases.
- o Adding additional new or existing (physical, snapshot, logical, RAC or non-RAC) standby databases to an existing Data Guard configuration, for a total of one primary database, and from 1 to 9 standby databases in the same configuration.
- o Managing an entire Data Guard configuration, including all databases, redo transport services, and log apply services, through a client connection to any database in the

- configuration.
- Managing the protection mode for the broker configuration.
  - Invoking switchover or failover with a single command to initiate and control complex role changes across all databases in the configuration.
  - Configuring failover to occur automatically upon loss of the primary database, increasing availability without manual intervention.
  - Monitoring the status of the entire configuration, capturing diagnostic information, reporting statistics such as the redo apply rate and the redo generation rate, and detecting problems quickly with centralized monitoring, testing, and performance tools.

You can perform all management operations locally or remotely through the broker's easy-to-use interfaces: the Data Guard management pages in Oracle Enterprise Manager, which is the broker's graphical user interface (GUI), and the Data Guard command-line interface called DGMGRL.

These interfaces simplify the configuration and management of a Data Guard configuration. Table 1-1 provides a comparison of configuration management using the broker's interfaces and using SQL\*Plus.

### Table Configuration Management With and Without the Broker

|                              | <b>With the Broker</b>                                                                                                                                                                                                                                           | <b>Without the Broker</b>                                                                                                                                                                                                                                                              |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| General                      | Provides primary and standby database management as one unified configuration.                                                                                                                                                                                   | You must manage the primary and standby databases separately.                                                                                                                                                                                                                          |
| Standby Database Creation    | Provides the Enterprise Manager wizards that automate and simplify the steps required to create a configuration with an Oracle database on each site, including creating the standby control file, online redo log files, datafiles, and server parameter files. | <p>You must manually:</p> <ul style="list-style-type: none"> <li>▪ Copy the database files to the standby database.</li> <li>▪ Create a control file on the standby database.</li> <li>▪ Create server parameter or initialization parameter files on the standby database.</li> </ul> |
| Configuration and Management | Enables you to configure and manage multiple databases from a single location and automatically unifies all of the databases in the broker configuration.                                                                                                        | <p>You must manually:</p> <ul style="list-style-type: none"> <li>▪ Set up redo transport services and log apply services on each database in the configuration.</li> <li>▪ Manage the primary database and standby databases individually.</li> </ul>                                  |

|            | <b>With the Broker</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | <b>Without the Broker</b>                                                                                                                                                                                                                                                                                                                                                                |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Control    | <ul style="list-style-type: none"> <li>▪ Automatically set up redo transport services and log apply services. Simplify management of these services, especially in an Oracle RAC environment.</li> <li>▪ Simplifies switchovers and failovers by allowing you to invoke them through a single command.</li> <li>▪ Automates failover by allowing the broker to determine if failover is necessary and to initiate failover to a specified target standby database, with no need for DBA intervention and with either no loss of data or with a configurable amount of data loss.</li> <li>▪ Integrates Cluster Ready Services (CRS) Foot 1 and instance management over database role transitions.</li> <li>▪ Provides mouse-driven database state changes and a unified presentation of configuration and database status.</li> <li>▪ Provides mouse-driven property changes.</li> </ul> | <p>You must manually:</p> <ul style="list-style-type: none"> <li>▪ Use multiple SQL*Plus statements to manage the database.</li> <li>▪ Coordinate sequences of multiple commands across multiple database sites to execute switchover and failover operations.</li> <li>▪ Coordinate sequences of multiple commands to manage services and instances during role transitions.</li> </ul> |
| Monitoring | <ul style="list-style-type: none"> <li>▪ Provides continuous monitoring of the configuration health, database health, and other runtime parameters.</li> <li>▪ Provides a unified updated status and detailed reports.</li> <li>▪ Provides integration with Oracle Enterprise Manager events.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | <p>You must manually:</p> <ul style="list-style-type: none"> <li>▪ Monitor the status and runtime parameters using fixed views on each database—there is no unified view of status for all of the databases in the configuration.</li> <li>▪ Provide a custom method for monitoring Oracle Enterprise Manager events.</li> </ul>                                                         |

## 79.2 Benefits of Data Guard Broker

The broker's interfaces improve usability and centralize management and monitoring of the Data Guard configuration. Available as a feature of the Enterprise Edition and Personal Edition of the Oracle database, the broker is also integrated with the Oracle database and Oracle Enterprise Manager. These broker attributes result in the following benefits:

**Disaster protection:** By automating many of the manual tasks required to configure and monitor a Data Guard configuration, the broker enhances the high availability, data protection, and disaster protection capabilities that are inherent in Oracle Data Guard. Access is possible through a client to any system in the Data Guard configuration, eliminating any single point of failure. If the primary database fails, the broker automates the process for any one of the standby databases to replace the primary database and take over production processing. The database availability that Data Guard provides makes it easier to protect your data.

**Higher availability and scalability with Oracle Real Application Clusters (RAC) Databases:** While Oracle Data Guard broker enhances disaster protection by maintaining transactionally consistent copies of the primary database, Data Guard, configured with Oracle high availability solutions such as Oracle Real Application Clusters (RAC) databases, further enhances the availability and scalability of any given copy of that database. The intrasite high availability of an Oracle RAC

database complements the intersite protection that is provided by Data Guard broker.

Consider that you have a cluster system hosting a primary Oracle RAC database comprised of multiple instances sharing access to that database. Further consider that an unplanned failure has occurred. From a Data Guard broker perspective, the primary database remains available as long as at least one instance of the clustered database continues to be available for transporting redo data to the standby databases. Oracle Clusterware manages the availability of instances of an Oracle RAC database.

Cluster Ready Services (CRS), a subset of Oracle Clusterware, works to rapidly recover failed instances to keep the primary database available. If CRS is unable to recover a failed instance, the broker continues to run automatically with one less instance. If the last instance of the primary database fails, the broker provides a way to fail over to a specified standby database. If the last instance of the primary database fails, and fast-start failover is enabled, the broker can continue to provide high availability by automatically failing over to a pre-determined standby database.

The broker is integrated with CRS so that database role changes occur smoothly and seamlessly. This is especially apparent in the case of a planned role switchover (for example, when a physical standby database is directed to take over the primary role while the former primary database assumes the role of standby).

The broker and CRS work together to temporarily suspend service availability on the primary database, accomplish the actual role change for both databases during which CRS works with the broker to properly restart the instances as necessary, and then start services defined on the new primary database. The broker manages the underlying Data Guard configuration and its database roles while CRS manages service availability that depends upon those roles. Applications that rely on CRS for managing service availability will see only a temporary suspension of service as the role change occurs in the Data Guard configuration.

Note that while CRS helps to maintain the availability of the individual instances of an Oracle RAC database, the broker coordinates actions that maintain one or more physical or logical copies of the database across multiple geographically dispersed locations to provide disaster protection. Together, the broker and Oracle Clusterware provide a strong foundation for Oracle's high-availability architecture.

**Automated creation of a Data Guard configuration:** The broker helps you to logically define and create a Data Guard configuration consisting of a primary database and (physical or logical, snapshot, RAC or non-RAC) standby databases. The broker automatically communicates between the databases in a Data Guard configuration using Oracle Net Services. The databases can be local or remote, connected by a LAN or geographically dispersed over a WAN.

Oracle Enterprise Manager provides a wizard that automates the complex tasks involved in creating a broker configuration, including:

- Adding an existing standby database, or a new standby database created from existing backups taken through Enterprise Manager
- Configuring the standby control file, server parameter file, and datafiles
- Initializing communication with the standby databases
- Creating standby redo log files
- Enabling Flashback Database if you plan to use fast-start failover

Although DGMGRL cannot automatically create a new standby database, you can use DGMGRL commands to configure and monitor an existing standby database, including those created using Enterprise Manager. **Easy configuration of additional standby databases:** After you create a Data Guard configuration consisting of a primary and a standby database, you can add up to eight new or existing, physical, snapshot, or logical standby databases to each Data Guard configuration. Oracle Enterprise Manager provides an Add Standby Database wizard to guide you through the process of adding more databases. It also makes all Oracle Net Services configuration changes necessary to support redo transport services and log apply services across the configuration. **Simplified, centralized, and extended management:** You can issue commands to manage many aspects of the broker configuration. These include:

- Simplify the management of all components of the configuration, including the primary and standby databases, redo transport services, and log apply services.
- Coordinate database state transitions and update database properties dynamically with the broker recording the changes in a broker configuration file that includes profiles of all the databases in the configuration. The broker propagates the changes to all databases in the configuration and their server parameter files.
- Simplify the control of the configuration protection modes (to maximize protection, to maximize availability, or to maximize performance).
- Invoke the Enterprise Manager verify operation to ensure that redo transport services and log apply services are configured and functioning properly.

**Simplified switchover and failover operations:** The broker simplifies switchovers and failovers by allowing you to invoke them using a single key click in Oracle Enterprise Manager or a single command at the DGMGRL command-line interface (referred to in this documentation as manual failover). For lights-out administration, you can enable fast-start failover to allow the broker to determine if a failover is necessary and to initiate the failover to a pre-specified target standby database automatically, with no need for DBA intervention. Fast-start failover can be configured to occur with no data loss or with a configurable amount of data loss.

Fast-start failover allows you to increase availability with less need for manual intervention, thereby reducing management costs. Manual failover gives you control over exactly when a failover occurs and to which target standby database. Regardless of the method you choose, the broker coordinates the role transition on all databases in the configuration. Once failover is complete, the broker posts the DB\_DOWN event to notify applications that the new primary is available.

Note that you can use the DBMS\_DG PL/SQL package to enable an application to initiate a fast-start failover when it encounters specific conditions.

Only one command is required to initiate complex role changes for switchover or failover operations across all databases in the configuration. The broker automates switchover and failover to a specified standby database in the broker configuration. Enterprise Manager enables you to select a new primary database from a set of viable standby databases (enabled and running, with normal status). The DGMGRL SWITCHOVER and FAILOVER commands only require you to specify the target standby database before automatically initiating and completing the many steps in switchover or failover operations across the multiple databases in the configuration.

**Built-in monitoring and alert and control mechanisms:** The broker provides built-in validation that monitors the health of all of the databases in the configuration. From any system in the configuration connected to any database, you can capture diagnostic information and detect obvious and subtle problems quickly with centralized monitoring, testing, and performance tools. Both Enterprise Manager and DGMGRL retrieve a complete configuration view of the progress of redo transport services on the primary database and the progress of Redo Apply or SQL Apply on the standby database.

The ability to monitor local and remote databases and respond to events is significantly enhanced by the broker's health check mechanism and tight integration with the Oracle Enterprise Manager event management system. **Transparent to application:** Use of the broker is possible for any database because the broker works transparently with applications; no application code changes are required to accommodate a configuration that you manage with the broker.

## 79.3 Data Guard Broker Management Model

The broker simplifies the management of a Data Guard environment by performing operations on the following logical objects:

1. Configuration of databases
2. A single database

The broker supports one or more Data Guard configurations, each of which includes a profile for the primary database and each standby database. A supported broker configuration consists of:

- o A configuration object, which is a named collection of database profiles. A database profile is a description of a database object including its current state, current status, and properties. The configuration object profiles one primary database and its standby databases that can include a mix of physical, snapshot, and logical standby databases. The databases of a given configuration are typically distributed across multiple host systems.
- o Database objects, corresponding to primary or standby databases. The broker uses a database object's profile to manage and control the state of a single database on a given system. The database object may be comprised of one or more instance objects if this is an Oracle RAC database.
- o Instance objects. The broker treats a database as a collection of one or more named instances. The broker automatically discovers the instances and associates them with their database.

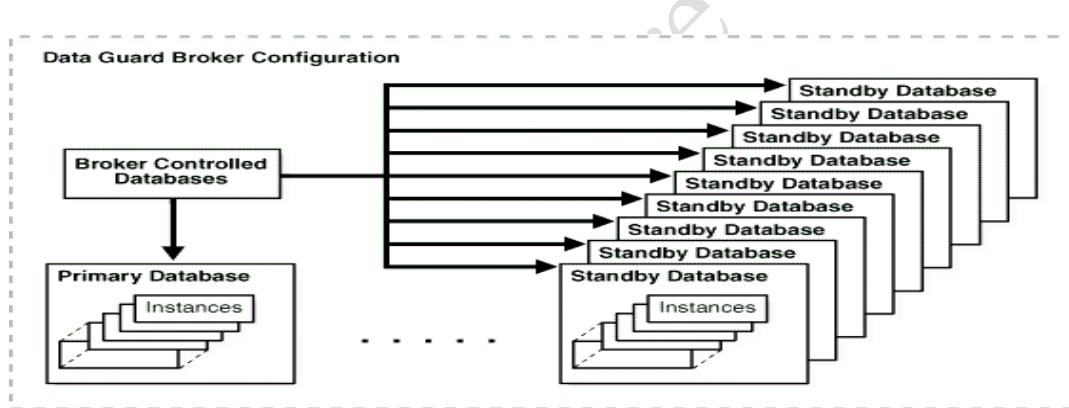


Figure Relationship of Objects Managed by the Data Guard Broker

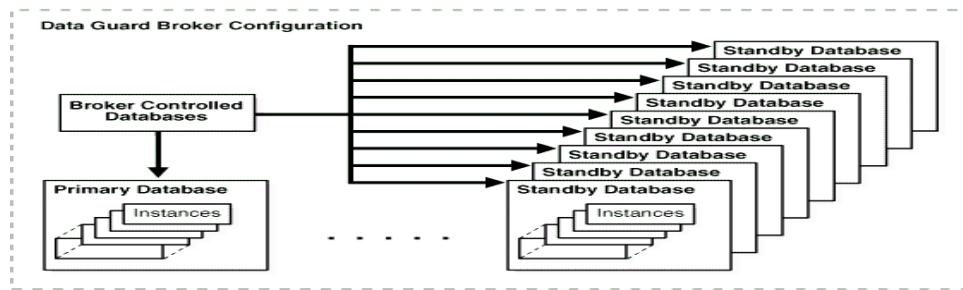
### Data Guard Broker Components

The Oracle Data Guard broker consists of the following components:

1. Oracle Enterprise Manager
2. Data Guard Command-Line Interface (DGMGRL)
3. Data Guard Monitor

Oracle Enterprise Manager and the Data Guard command-line interface (DGMGRL) are the broker client interfaces that help you define and manage a configuration consisting of a collection of primary and standby databases. DGMGRL also includes commands to create an observer, a process that facilitates fast-start failover. Section 1.5 describes these interfaces in more detail.

The Data Guard monitor is the broker server-side component that is integrated with the Oracle database. Data Guard monitor is composed of several processes, including the DMON process, and broker configuration files that allow you to control the databases of that configuration, modify their behavior at runtime, monitor the overall health of the configuration, and provide notification of other operational characteristics. Section 1.6 describes the Data Guard monitor in more detail.

**Figure Oracle Data Guard Broker**

Description of "Figure 1-2 Oracle Data Guard Broker"

## 79.3 Data Guard Broker User Interfaces

You can use either of the broker's user interfaces to create a broker configuration and to control and monitor the configuration. The following sections describe the broker's user interfaces:

1. Oracle Enterprise Manager
2. Data Guard Command-Line Interface (DGMGRL)

### 79.3.1 Oracle Enterprise Manager

Oracle Enterprise Manager works with the Data Guard monitor to automate and simplify the management of a Data Guard configuration.

With Enterprise Manager, the complex operations of creating and managing standby databases are simplified through Data Guard management pages and wizards, including:

- An Add Standby Database wizard that helps you to create a broker configuration, if one does not already exist, having a primary database and a local or remote standby database. The wizard can create a physical, snapshot, or logical standby database or import an existing physical, snapshot, or logical (RAC or non-RAC) standby database. If the wizard creates a physical, snapshot, or logical standby database, the wizard also automates the creation of the standby control file, server parameter file, online and standby redo log files, and the standby datafiles.
- A switchover operation that helps you switch roles between the primary database and a standby database.
- A failover operation that changes one of the standby databases to the role of a primary database.
- Performance tools and graphs that help you monitor and tune redo transport services and log apply services.
- Property pages that allow you to set database properties on any database and, if applicable, the settings are immediately propagated to all other databases and server parameter files in the configuration.
- Event reporting through e-mail.

In addition, it makes all Oracle Net Services configuration changes necessary to support redo transport services and log apply services.

### 79.3.2 Data Guard Command-Line Interface (DGMGRL)

The Data Guard command-line interface (DGMGRL) enables you to control and monitor a Data Guard configuration from the DGMGRL prompt or within scripts. You can perform most of the activities required to manage and monitor the databases in the configuration using DGMGRL

commands. DGMGRL also includes commands to create an observer process that continuously monitors the primary and target standby databases and evaluates whether failover is necessary, and then initiates a fast-start failover when conditions warrant.

Table DGMGRL Commands

| Command           | Description                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------|
| <b>ADD</b>        | Adds a standby database to the broker configuration                                           |
| <b>CONNECT</b>    | Connects to an Oracle instance                                                                |
| <b>CONVERT</b>    | Converts a database between a physical standby database and a snapshot standby database       |
| <b>CREATE</b>     | Creates a broker configuration                                                                |
| <b>DISABLE</b>    | Disables a configuration, a database, fast-start failover, or a fast-start failover condition |
| <b>EDIT</b>       | Edits a configuration, database, or instance                                                  |
| <b>ENABLE</b>     | Enables a configuration, a database, fast-start failover, or a fast-start failover condition  |
| <b>EXIT</b>       | Exits the program                                                                             |
| <b>FAILOVER</b>   | Changes a standby database to be the primary database                                         |
| <b>HELP</b>       | Displays description and syntax for individual commands                                       |
| <b>QUIT</b>       | Exits the program                                                                             |
| <b>REINSTATE</b>  | Changes a database marked for reinstatement into a viable standby database                    |
| <b>REM</b>        | Comment to be ignored by DGMGRL                                                               |
| <b>REMOVE</b>     | Removes a configuration, database, or instance                                                |
| <b>SHOW</b>       | Displays information about a configuration, database, instance, or fast-start failover        |
| <b>SHUTDOWN</b>   | Shuts down a currently running Oracle instance                                                |
| <b>START</b>      | Starts the fast-start failover observer                                                       |
| <b>STARTUP</b>    | Starts an Oracle database instance                                                            |
| <b>STOP</b>       | Stops the fast-start failover observer                                                        |
| <b>SWITCHOVER</b> | Switches roles between the primary database and a standby database                            |

## 79.4 Data Guard Monitor

The configuration, control, and monitoring functions of the broker are implemented by server-side software and configuration files that are maintained for each database that the broker manages. The software is called the Data Guard monitor.

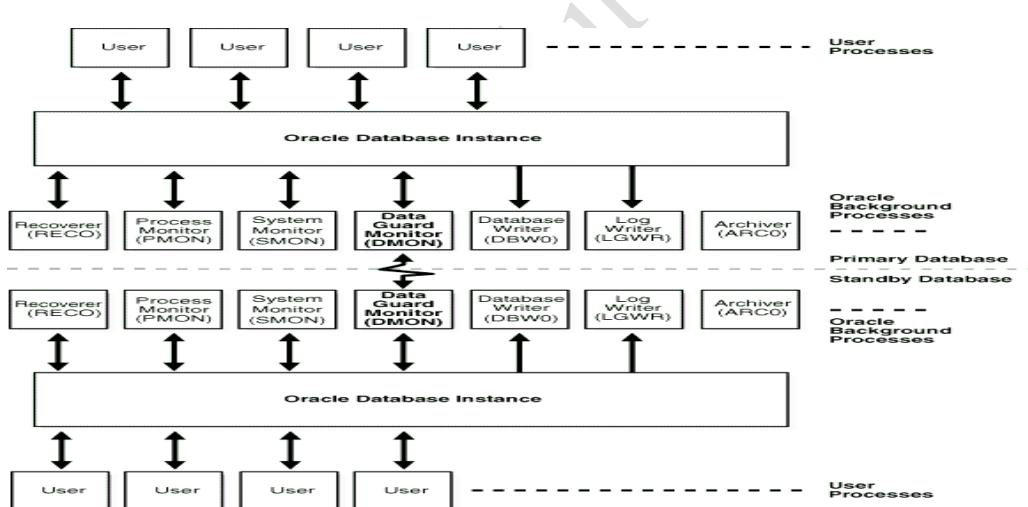
The following sections describe how the Data Guard monitor interacts with the Oracle database and with remote Data Guard monitors to manage the broker configuration.

### 79.4.1 Data Guard Monitor (DMON) Process

The Data Guard monitor process (DMON) is an Oracle background process that runs for every database instance that is managed by the broker. When you start the Data Guard broker, a DMON process is created. Whether you use Oracle Enterprise Manager or DGMGRL to manage a database, the DMON process is the server-side component that interacts with the local database and the DMON processes of the other databases to perform the requested function. The DMON process is also responsible for monitoring the health of the broker configuration and for ensuring that every database has a consistent description of the configuration.

Figure shows the broker's DMON process as one of several background processes that constitute an instance of the Oracle database. Each database instance shown in the figure has its own DMON process.

Figure Databases with Broker (DMON) Processes. The zigzag arrow in the center of Figure 1-3 represents the two-way Oracle Net Services communication channel that exists between the DMON processes of two databases in the same broker configuration.



This two-way communication channel is used to pass requests between databases and to monitor the health of all of the databases in the broker configuration.

### 79.4.2 Configuration Management

The broker's DMON process persistently maintains profiles about all database objects in the broker configuration in a binary configuration file. A copy of this file is maintained by the DMON process for each of the databases that belong to the broker configuration. If it is an Oracle RAC database, each database's copy of the file is shared by all instances of the database.

This configuration file contains profiles that describe the states and properties of the databases in the configuration. For example, the file records the databases that are part of the configuration,

the roles and properties of each of the databases, and the state of each database in the configuration.

The configuration data is managed transparently by the DMON process to ensure that the configuration information is kept consistent across all of the databases. The broker uses the data in the configuration file to configure and start the databases, control each database's behavior, and provide information to DGMGRL and Oracle Enterprise Manager.

Whenever you add databases to a broker configuration, or make a change to an existing database's properties, each DMON process records the new information in its copy of the configuration file.

### 79.4.3 Database Property Management

Associated with each database are various properties that the DMON process uses to control the database's behavior. The properties are recorded in the configuration file as a part of the database's object profile that is stored there. Many database properties are used to control database initialization parameters related to the Data Guard environment.

To ensure that the broker can update the values of parameters in both the database itself and in the configuration file, you must use a server parameter file to control static and dynamic initialization parameters. The use of a server parameter file gives the broker a mechanism that allows it to reconcile property values selected by the database administrator (DBA) when using the broker with any related initialization parameter values recorded in the server parameter file.

When you set values for database properties in the broker configuration, the broker records the change in the configuration file and propagates the change to all of the databases in the Data Guard configuration.

Note: The broker supports both the default and nondefault server parameter file filenames. If you use a nondefault server parameter filename, the initialization parameter file must include the complete filename and location of the server parameter file. If this is an Oracle RAC database, there must be one nondefault server parameter file for all instances.

## 80. What's New in Oracle Data Guard?

This preface describes the new features added to Oracle Data Guard in release 15.1 and provides links to additional information. The features and enhancements described in this preface were added to Oracle Data Guard in 11g Release 1 (15.1). The new features are described under the following main areas:

- New Features Common to Redo Apply and SQL Apply
- New Features Specific to Redo Apply and Physical Standby Databases
- New Features Specific to SQL Apply and Logical Standby Databases

### New Features Common to Redo Apply and SQL Apply

The following enhancements to Oracle Data Guard in 11g Release 1 (15.1) improve ease-of-use, manageability, performance, and include innovations that improve disaster-recovery capabilities:

- Compression of redo traffic over the network in a Data Guard configuration
- This feature improves redo transport performance when resolving redo gaps by compressing redo before it is transmitted over the network.
- Redo transport response time histogram
- The V\$REDO\_DEST\_RESP\_HISTOGRAM dynamic performance view contains a histogram of response times for each SYNC redo transport destination. The data in this view can be used to assist in the determination of an appropriate value for the LOG\_ARCHIVE\_DEST\_n.NET\_TIMEOUT attribute.
- Faster role transitions
- Strong authentication for redo transport network sessions
- Redo transport network sessions can now be authenticated using SSL. This provides strong authentication and makes the use of remote login password files optional in a Data Guard configuration.
- Simplified Data Guard management interface
- The SQL statements and initialization parameters used to manage a Data Guard configuration have been simplified through the deprecation of redundant SQL clauses and initialization parameters.
- Enhancements around DB\_UNIQUE\_NAME
- You can now find the DB\_UNIQUE\_NAME of the primary database from the standby database by querying the new PRIMARY\_DB\_UNIQUE\_NAME column in the V\$DATABASE view. Also, Oracle Data Guard release 11g ensures each database's DB\_UNIQUE\_NAME is different. After upgrading to 11g, any databases with the same DB\_UNIQUE\_NAME will not be able to communicate with each other.
- Use of physical standby database for rolling upgrades
- A physical standby database can now take advantage of the rolling upgrade feature provided by a logical standby. Through the use of the new KEEP IDENTITY clause option to the SQL ALTER DATABASE RECOVER TO LOGICAL STANDBY statement, a physical standby database can be temporarily converted into a logical standby database for the rolling upgrade, and then reverted back to the original configuration of a primary database and a physical standby database when the upgrade is done.
- Heterogeneous Data Guard Configuration

This feature allows a mix of Linux and Windows primary and standby databases in the same Data Guard configuration.

### New Features Specific to Redo Apply and Physical Standby Databases

The following list summarizes the new features that are specific to Redo Apply and physical standby databases in Oracle Database 11g Release 1 :

- Real-time query capability of physical standby
- This feature makes it possible to query a physical standby database while Redo Apply is active.
- Snapshot standby
- A snapshot standby database is new type of updatable standby database that provides full data protection for a primary database.
- Lost-write detection using a physical standby
- A "lost write" is a serious form of data corruption that can adversely impact a database. It occurs when an I/O subsystem acknowledges the completion of a block write in the database, while in fact

the write did not occur in the persistent storage. This feature allows a physical standby database to detect lost writes to a primary or physical standby database.

## Improved Integration with RMAN

A number of enhancements in RMAN help to simplify backup and recovery operations across all primary and physical standby databases, when using a catalog. Also, you can use the RMAN DUPLICATE command to create a physical standby database over the network without a need for pre-existing database backups.

### New Features Specific to SQL Apply and Logical Standby Databases

The following list summarizes the new features for SQL Apply and logical standby databases in Oracle Database 11g Release 1 (15.1):

Support for additional object datatypes and PL/SQL package support

- XML stored as CLOB (Support for additional PL/SQL Package)
- DBMS\_RLS (row level security or Virtual Private Database)

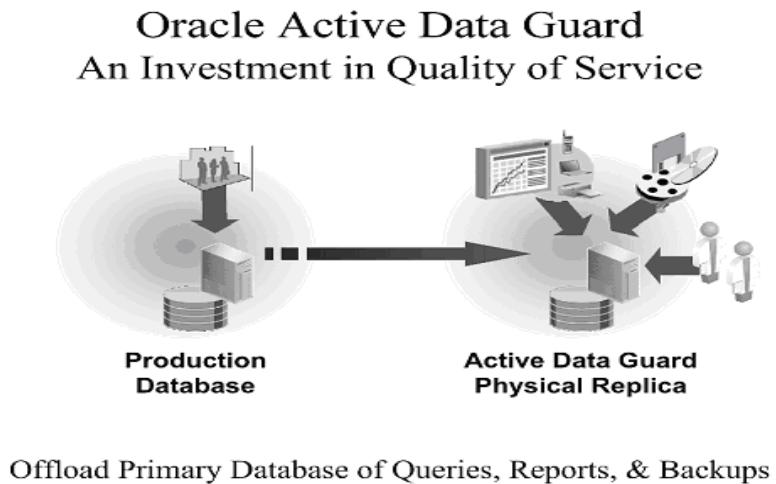
## DBMS\_FGA

- Support Transparent Data Encryption (TDE)
- Data Guard SQL Apply can be used to provide data protection for the primary database with Transparent Data Encryption enabled. This allows a logical standby database to provide data protection for applications with advanced security requirements.
- Dynamic setting of Data Guard SQL Apply parameters
- You can now configure specific SQL Apply parameters without requiring SQL Apply to be restarted. Using the DBMS\_LOGSTDBY.APPLY\_SET package, you can dynamically set initialization parameters, thus improving the manageability, uptime, and automation of a logical standby configuration.
- In addition, the APPLY\_SET and APPLY\_UNSET subprograms include two new parameters: LOG\_AUTO\_DEL\_RETENTION\_TARGET and EVENT\_LOG\_DEST.
- Enhanced RAC switchover support for logical standby databases
- When switching over to a logical standby database where either the primary database or the standby database is using Oracle RAC, the SWITCHOVER command can be used without having to shut down any instance, either at the primary or at the logical standby database.
- Enhanced DDL handling in Oracle Data Guard SQL Apply
- SQL Apply will execute parallel DDLs in parallel (based on availability of parallel servers).
- Use of the PL/SQL DBMS\_SCHEDULER package to create Scheduler jobs on a standby database
- Scheduler Jobs can be created on a standby database using the PL/SQL DBMS\_SCHEDULER package and can be associated with an appropriate database role so that they run when intended (for example, when the database is the primary, standby, or both).

## 81. Active Dataguard

### 81.1 Traditional Data Guard

The most popular use of Oracle Data Guard is to synchronize a physical standby database with its production counterpart for data protection and high availability. Prior to Oracle Database 11g, physical standby databases typically operate in continuous Redo Apply mode (i.e. continuously applying changes from the production database) to ensure that a database failover can be accomplished within seconds of an outage at the production site. Redo Apply must be stopped to enable read access to a Data Guard 10g standby database, eventually resulting in a replica that has stale data and extending the time required to complete a failover operation.



### 81.2 Oracle Active Data Guard

Oracle Active Data Guard enables a physical standby database to be open for read-only access – for reporting, simple or complex queries, sorting, web-based access, etc. while changes from the production database are being applied to it. All queries reading from the physical replica execute in real-time, and return current results. This means any operation that requires up-to-date read-only access can be offloaded to the replica, enhancing and protecting the performance of the production database. This capability makes it possible for Active Data Guard to be deployed for a wide variety of business applications. Examples include:

1. Telecommunications: Technician access to service schedules, customer inquiries to check status of service requests.
2. Healthcare: Fast access to up-to-date medical records.
3. Finance and Administration: Ad-hoc queries and reports
4. Transportation: Package tracking queries, schedule status
5. Web-business: Catalog browsing, order status, scale-out using reader farms

Active Data Guard also provides support for RMAN block-change tracking, enabling very fast incremental backups to be offloaded from the production database to the standby database. Fast incremental backups can be orders of magnitude faster than backups taken on physical standby databases in earlier releases.

## 81.3 Unique Advantages of Oracle Active Data Guard

Active Data Guard is an evolution of Data Guard technology, providing unique performance advantages while leveraging all other enhancements included in Oracle Data Guard 11g. For example, any Data Guard 11g physical standby database can be easily converted to a Snapshot Standby. A Snapshot Standby is open read-write and is ideally suited as a test system, able to process transactions independent of the primary database.

A Snapshot Standby maintains protection by continuing to receive data from the production database, archiving it for later use. When tests are complete, a single command discards changes made while open read-write and quickly resynchronizes the standby database with the primary.

It is easy to see how all of these capabilities build upon each other – and because they all use the same common infrastructure, they generate significant dividends for Oracle customers. The same replica maintained by Oracle Active Data Guard to improve the quality of service and performance of the primary database, can be used during non-peak hours as a test database using Snapshot Standby, and at all times can also serve as a disaster recovery solution.

So rather than maintain multiple replicas on costly redundant storage using different technologies to address different requirements – Oracle Active Data Guard provides a common infrastructure and single replica with one management interface to achieve the same objectives. Easier, less expensive, more functional – Oracle Active Data Guard is hard to beat.

## 82. Snapshot Standby Databases

A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database. A snapshot standby database receives and archives, but does not apply, redo data from its primary database. Redo data received from the primary database is applied when a snapshot standby database is converted back into a physical standby database, after discarding all local updates to the snapshot standby database.

A snapshot standby database typically diverges from its primary database over a time because redo data from the primary database is not applied as it is received. Local updates to the snapshot standby database will cause additional divergence. The data in the primary database is fully protected however, because a snapshot standby can be converted back into a physical standby database at any time, and the redo data received from the primary will then be applied.

### 82.1 Benefits of a Snapshot Standby Database

A snapshot standby database is a fully updatable standby database that provides disaster recovery and data protection benefits that are similar to those of a physical standby database. Snapshot standby databases are best used in scenarios where the benefit of having a temporary, updatable snapshot of the primary database justifies additional administrative complexity and increased time to recover from primary database failures.

The benefits of using a snapshot standby database include the following:

- It provides an exact replica of a production database for development and testing purposes, while maintaining data protection at all times
- It can be easily refreshed to contain current production data by converting to a physical standby and resynchronizing

The ability to create a snapshot standby, test, resynchronize with production, and then again create a snapshot standby and test, is a cycle that can be repeated as often as desired. The same process can be used to easily create and regularly update a snapshot standby for reporting purposes where read/write access to data is required.

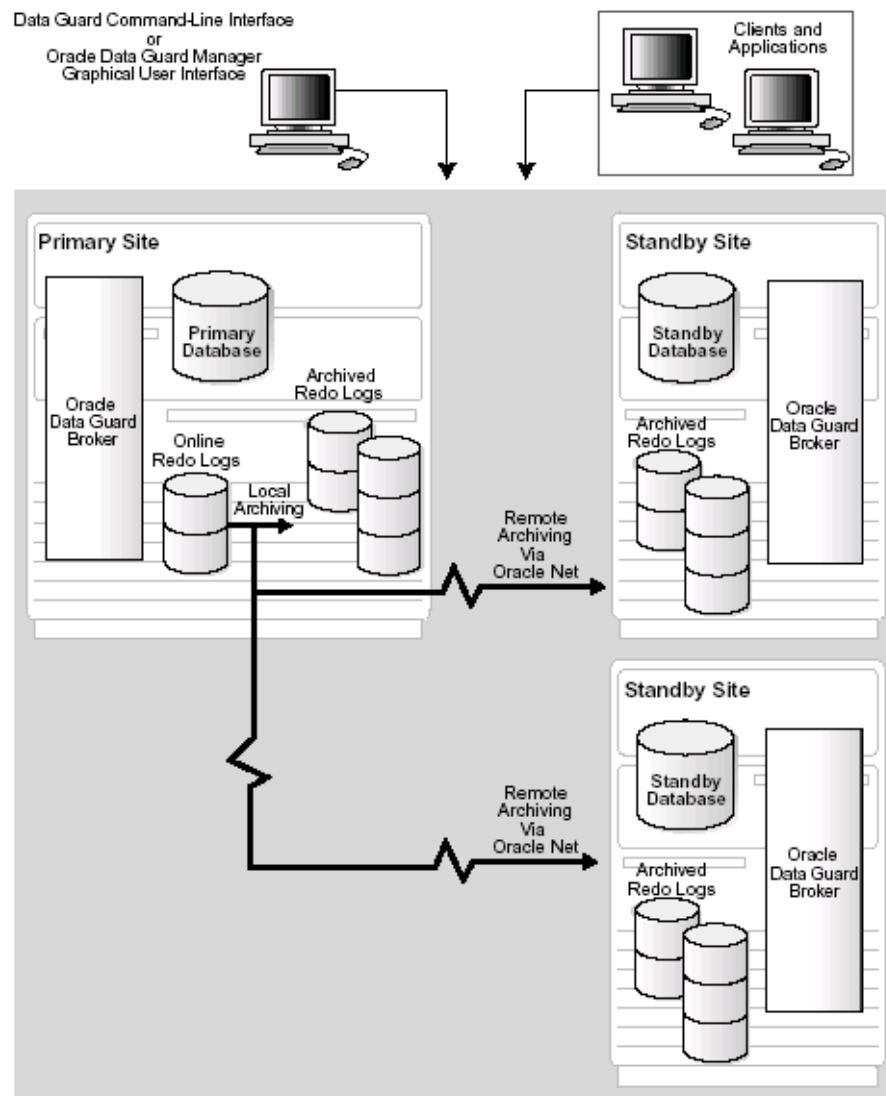
### 82.2 Using a Snapshot Standby Database

Once a physical standby database has been converted into a snapshot standby database, it can be opened in read-write mode and it is fully updatable. A snapshot standby database continues to receive and archive redo data from the primary database, and this redo data will be automatically applied when the snapshot standby database is converted back into a physical standby database. A snapshot standby database has the following characteristics:

Redo data gap detection and resolution works just as it does on a physical standby database.

- If the primary database moves to new database branch (for example, because of a Flashback Database or an OPEN RESETLOGS, the snapshot standby database will continue accepting redo from new database branch.
- A snapshot standby database cannot be the target of a switchover or failover. A snapshot standby database must first be converted back into a physical standby database before performing a role transition to it.
- After a switchover or failover between the primary database and one of the physical or logical standby databases in a configuration, the snapshot standby database can receive redo data from the new primary database after the role transition.
- A snapshot standby database cannot be the only standby database in a Maximum Protection Data Guard configuration.

# Data Guard Scenario



## 83. Data Guard Enhancements in Oracle 11 G R2

The following sections describe new features in this release that provide improvements in Oracle Data Guard.

### 83.1 Redo Apply and SQL Apply

- A Data Guard configuration can now consist of a primary database and up to 30 standby databases.
- The `FAL_CLIENT` database initialization parameter is no longer required.
- The default archive destination used by the Oracle Automatic Storage Management (Oracle ASM) feature and the fast recovery area feature has changed from `LOG_ARCHIVE_DEST_10` to `LOG_ARCHIVE_DEST_1`.
- Redo transport compression is no longer limited to compressing redo data only when a redo gap is being resolved. When compression is enabled for a destination, all redo data sent to that destination is compressed.
- The new `ALTER SYSTEM FLUSH REDO` SQL statement can be used at failover time to flush unsent redo from a mounted primary database to a standby database, thereby allowing a zero data loss failover to be performed even if the primary database is not running in a zero data loss data protection mode.

### 83.2 Redo Apply

- You can configure apply lag tolerance in a real-time query environment by using the new `STANDBY_MAX_DATA_DELAY` parameter.
- You can use the new `ALTER SESSION SYNC WITH PRIMARY` SQL statement to ensure that a suitably configured physical standby database is synchronized with the primary database as of the time the statement is issued.
- The `V$DATAGUARD_STATS` view has been enhanced to a greater degree of accuracy in many of its columns, including apply lag and transport lag.
- You can view a histogram of apply lag values on the physical standby. To do so, query the new `V$STANDBY_EVENT_HISTOGRAM` view.
- A corrupted data block in a primary database can be automatically replaced with an uncorrupted copy of that block from a physical standby database that is operating in real-time query mode. A corrupted block in a physical standby database can also be automatically replaced with an uncorrupted copy of the block from the primary database.

### 83.3 SQL Apply

- Logical standby databases and the LogMiner utility support tables with basic table compression, OLTP table compression, and hybrid columnar compression.
- Logical standby and the LogMiner utility support tables with SecureFile LOB columns. Compression and encryption operations on SecureFile LOB columns are also supported. (De-duplication operations and fragment-based operations are not supported.)
- Changes made in the context of XA global transactions on an Oracle RAC primary database are replicated on a logical standby database.
- Online redefinition performed at the primary database using the `DBMS_REDEFINITION` PL/SQL package is transparently replicated on a logical standby database.
- Logical Standby supports the use of editions at the primary database, including the use of edition-based redefinition to upgrade applications with minimal downtime.
- Logical standby databases support Streams Capture. This allows you to offload processing from the primary database in one-way information propagation configurations and make the logical standby the hub that propagates information to multiple databases. Streams Capture can also propagate changes that are local to the logical standby database.

## 83.4 Compressed Table Support in Logical Standby Databases and Oracle LogMiner

Compressed tables (that is, tables with compression that support both OLTP and direct load operations) are supported in logical standby databases and Oracle LogMiner.

With support for this additional storage attribute, logical standby databases can now provide data protection and reporting benefits for a wider range of tables.

## 83.5 Configurable Real-Time Query Apply Lag Limit

A physical standby database can be open for read-only access while redo apply is active only if the Oracle Active Data Guard option is enabled. This capability is known as real-time query.

The new STANDBY\_MAX\_DATA\_DELAY session parameter can be used to specify a session-specific apply lag tolerance, measured in seconds, for queries issued by non-administrative users to a physical standby database that is in real-time query mode.

This capability allows queries to be safely offloaded from the primary database to a physical standby database, because it is possible to detect if the standby database has become unacceptably stale.

## 83.6 Support Up to 30 Standby Database

The number of standby databases that a primary database can support is increased from 9 to 30 in this release.

The capability to create 30 standby databases, combined with the functionality of the Oracle Active Data Guard option, allows the creation of reader farms that can be used to offload large scale read-only workloads from a production database.

## 83.7 Automatic Repair of Corrupt Data Blocks

A physical standby database operating in real-time query mode can also be used to repair corrupt data blocks in a primary database. If possible, any corrupt data block encountered when a primary database is accessed is automatically replaced with an uncorrupted copy of that block from a physical standby database operating in real-time query mode. Note that for this to work, the standby database must be synchronized with the primary database.

If a corrupt data block is discovered on a physical standby database, the server attempts to automatically repair the corruption by obtaining a copy of the block from the primary database if the following database initialization parameters are configured on the standby database:

The LOG\_ARCHIVE\_CONFIG parameter is configured with a DG\_CONFIG list and a LOG\_ARCHIVE\_DEST\_n parameter is configured for the primary database

The FAL\_SERVER parameter is configured and its value contains an Oracle Net service name for the primary database

If automatic repair is not possible, an ORA-1578 error is returned.

## 83.8 Manual Repair of Corrupt Data Blocks

The RMAN RECOVER BLOCK command is used to manually repair a corrupted data block. This command searches several locations for an uncorrupted copy of the data block. By default, one of the locations is any available physical standby database operating in real-time query mode. The EXCLUDE STANDBY option of the RMAN RECOVER BLOCK command can be used to exclude physical standby databases as a source for replacement blocks

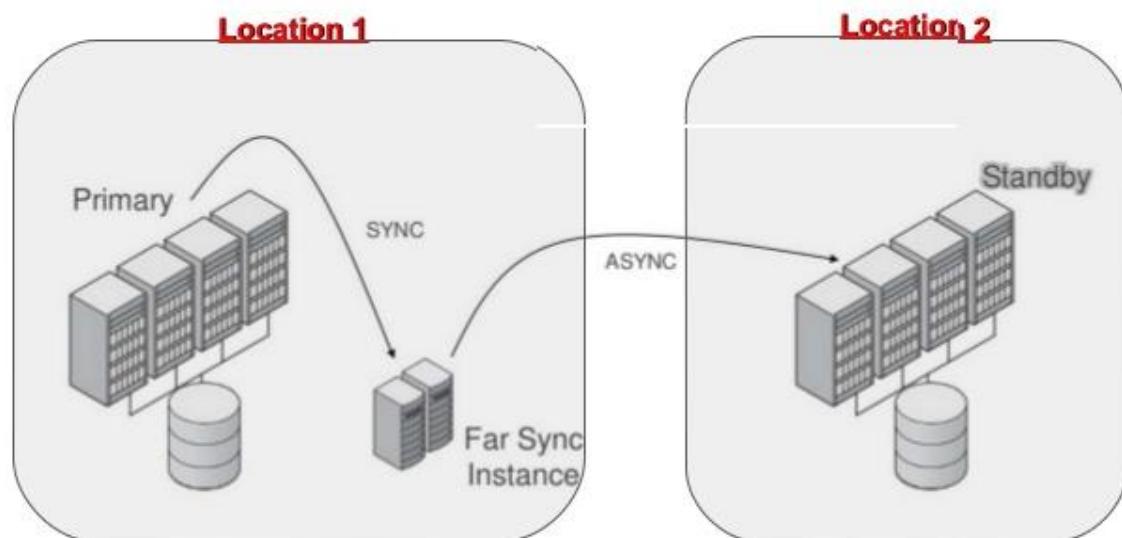
## 84. Data Guard Enhancements in Oracle 12C

### 84.1 Far Sync

An Oracle Data Guard far sync instance is a remote Oracle Data Guard destination that accepts redo from the primary database and then ships that redo to other members of the Oracle Data Guard configuration. A far sync instance manages a control file, receives redo into standby redo logs (SRLs), and archives those SRLs to local archived redo logs, but that is where the similarity with standbys ends. A far sync instance does not have user data files, cannot be opened for access, cannot run redo apply, and can never function in the primary role or be converted to any type of standby database.

Far sync instances are part of the Oracle Active Data Guard Far Sync feature, which requires an Oracle Active Data Guard license.

#### Far Sync



A far sync instance consumes very little disk and processing resources, yet provides the ability to failover to a terminal destination with zero data loss, as well as offload the primary database of other types of overhead (for example, redo transport).

All redo transport options available to a primary when servicing a typical standby destination are also available to it when servicing a far sync instance. And all redo transport options are available to a far sync instance when servicing terminal destinations (for example, performing redo transport compression, if you have a license for the Oracle Advanced Compression option).

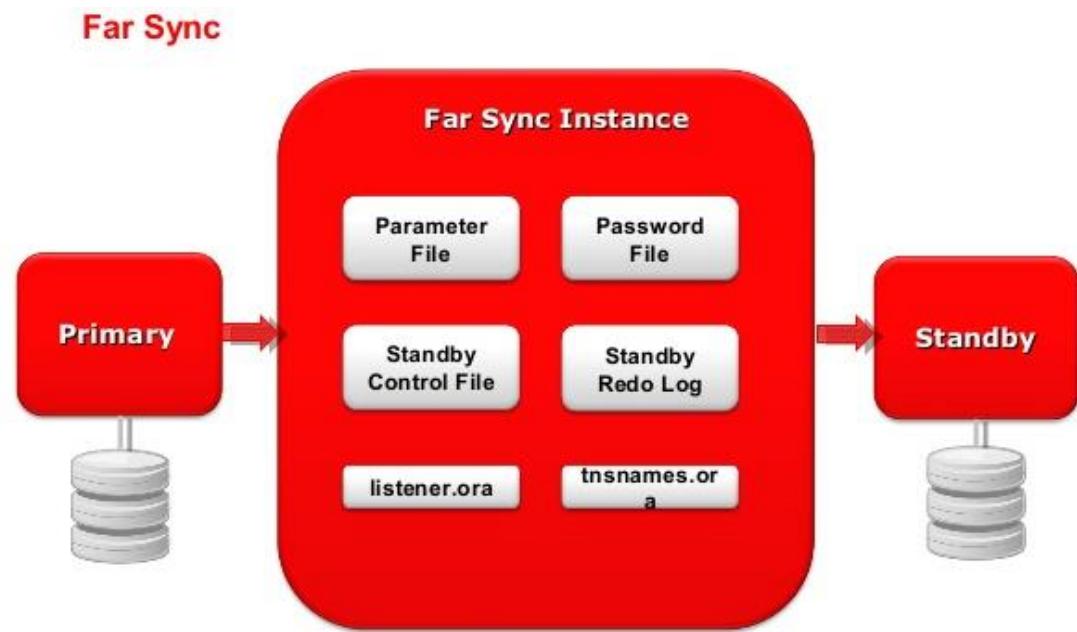
Many configurations have a primary database shipping redo to a standby database using asynchronous transport at the risk of some data loss at failover time. Using synchronous redo transport to achieve zero data loss may not be a viable option because of the impact on the commit response times at the primary due to network latency between the two databases.

Creating a far sync instance close to the primary has the benefit of minimizing impact on commit response times to an acceptable threshold (due to the smaller network latency between primary and far sync instance) while allowing for higher data protection guarantees -- if the primary were to fail, and assuming the far sync instance was synchronized at the time of the failure, the far sync instance and the terminal standby would coordinate a final redo shipment from the far sync

instance to the standby to ship any redo not yet available to the Standby and then perform a zero-data-loss failover.

This chapter contains the following sections:

- Creating a Far Sync Instance
- Additional Configurations
- Supported Protection Modes for Far Sync Instances



## 84.1 Creating a Far Sync Instance

Creating a far sync instance is similar to creating a physical standby except that data files do not exist at the far sync instance. Therefore, on a far sync instance there is no need to copy data files or restore data files from a backup. Once the far sync instance has been created, the configuration is modified to send redo synchronously from the primary database to the far sync instance in Maximum Availability mode and the far sync instance then forwards the redo asynchronously in real time. Lastly, the original asynchronous standby (referred to as the terminal standby) is configured to act as the alternate to the far sync instance in the event that communication with the far sync instance is interrupted.

### Note:

In a configuration that contains a far sync instance, there must still be a direct network connection between the primary database and the remote standby database. The direct connection between the primary and the remote standby is used to perform health checks and switchover processing tasks. It is not used for redo transport unless the standby has been configured as an alternate destination in case the far sync instance fails and there is no alternate far sync configured to maintain the protection level.

This section describes the following:

- Creating and Configuring a Far Sync Instance
- Configuring an ALTERNATE Destination

### 84.1.1 Creating and Configuring a Far Sync Instance

Take the following steps to create a far sync instance:

Create the control file for the far sync instance, as shown in the following example (the primary database does not have to be open, but it must at least be mounted):

```
SQL> ALTER DATABASE CREATE FAR SYNC INSTANCE CONTROLFILE AS
 '/arch2/chicagoFS/control01.ctl';
```

The resulting control file enables chicagoFS to operate as a far sync instance that receives redo from primary database chicago. Note that the path and file name shown are just an example; you could use any path or file name that you want.

Create a parameter file (PFILE) from the server parameter file (SPFILE) used by the primary database. Although most of the initialization settings in the parameter file are also appropriate for the far sync instance, some modifications must be made. For example, on a far sync instance, the DB\_FILE\_NAME\_CONVERT and LOG\_FILE\_NAME\_CONVERT parameters must be set, and the DB\_UNIQUE\_NAME of the far sync instance and the location of the far sync instance control file must be modified. Example shows sample parameter file content for a far sync instance with a DB\_UNIQUE\_NAME of chicagoFS.

#### Some of the Initialization Parameters Used For Far Sync Instances

##### Primary Database chicago

```
DB_UNIQUE_NAME=chicago
CONTROL_FILES='/arch1/chicago/control01.ctl'
DB_FILE_NAME_CONVERT='/boston','/chicago/'
LOG_FILE_NAME_CONVERT='/boston','/chicago/'
FAL_SERVER=boston
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston)'
LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIR
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'
```

##### Far Sync Instance chicagoFS

```
DB_UNIQUE_NAME=chicagoFS
CONTROL_FILES='/arch2/chicagoFS/control01.ctl'
DB_FILE_NAME_CONVERT='/chicago','/chicagoFS','/boston','/chicagoFS/'
LOG_FILE_NAME_CONVERT='/chicago','/chicagoFS','/boston','/chicagoFS/'
FAL_SERVER=chicago
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston)'
LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicagoFS'
LOG_ARCHIVE_DEST_2='SERVICE=boston ASYNC
VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=boston'
```

## Physical Standby boston

```
DB_UNIQUE_NAME=boston
CONTROL_FILES='/arch3/boston/control01.ctl'
DB_FILE_NAME_CONVERT='/chicago/','/boston/'
LOG_FILE_NAME_CONVERT='/chicago/','/boston/'
FAL_SERVER='chicagoFS','chicago'
LOG_ARCHIVE_CONFIG=DG_CONFIG=(chicago,chicagoFS,boston)
LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2='SERVICE=chicago ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago'
```

Create a server parameter file (spfile) from the edited parameter file (pfile) to facilitate any subsequent changes to parameter values. If you do not use an spfile, then a warning is returned in the SHOW CONFIGURATION output when the far sync instance is added to a Data Guard broker configuration.

.Use an operating system copy utility to copy the far sync instance control file created in Step 1 and the server parameter file (spfile) created in Step 3 from the primary system to the appropriate locations on the far sync instance system.

Create standby redo logs in the same way they are created for a regular standby.

Because the LOG\_FILE\_NAME\_CONVERT parameter was specified on the far sync instance the standby redo logs will be created automatically when redo transport begins from the primary.

### Note:

Standby redo log files used at the far sync instance cannot be shared with other databases. for standby redo log files also apply at the far sync instance.

If the far sync instance will be hosted on a Windows system, use the ORADIM utility to create a Windows service.

For example:

```
WINNT> oradim -NEW -SID boston -STARTMODE manual
```

The ORADIM utility automatically determines the username for which this service should be created and prompts for a password for that username (if that username needs a password). See *Oracle Database Platform Guide for Microsoft Windows* for more information about using the ORADIM utility.

This step is optional if operating system authentication is used for administrative users and if SSL is used for redo transport authentication. If not, then copy the primary database's remote login password file to the appropriate directory on the far sync instance. The password file must be recopied whenever an administrative privilege (SYSDG, SYSOPER, SYSDBA, and so on) is granted or revoked, and after the password of any user with administrative privileges is changed.

On the far sync instance site, use Oracle Net Manager to configure a listener for the far sync instance.See *Oracle Database Net Services Administrator's Guide* for more information about the listener.

On the primary system, use Oracle Net Manager to create a network service name for the far sync instance (chicagoFS) that will be used by redo transport services.On the far sync instance system, use Oracle Net Manager to create a network service name for the primary (chicago) and the terminal standby (boston) that will be used by redo transport services.

The Oracle Net service name must resolve to a connect descriptor that uses the same protocol, host address, port, and service that you specified when you configured the listeners for the primary database, the far sync instance, and the terminal standby database. The connect descriptor must also specify that a dedicated server be used. See the *Oracle Database Net Services Administrator's Guide* for more information about service names.

Start the far sync instance in mount mode. Verify that the far sync instance is operating properly. For information about validating a configuration after you create a far sync instance, see Section 7.5, "Validating a Configuration." Increase the protection mode of the configuration to Maximum Availability. On the primary database, execute the following command:

```
SQL> ALTER DATABASE SET STANDBY TO MAXIMIZE AVAILABILITY;
```

## 84.2 Configuring an ALTERNATE Destination

After you perform the steps in Section 5.1.1, the far sync instance is now providing zero data loss capability for the configuration to the terminal standby at a remote site over the WAN. For the configuration to remain protected at a Maximum Performance level, in the event that communication with the far sync instance is lost, you can optionally configure the terminal standby to automatically become the alternate destination. This will reduce the amount of data loss by allowing Oracle Data Guard to ship redo asynchronously directly from the primary to the terminal standby, temporarily bypassing the far sync instance.

To configure an alternate destination, set the parameters on the primary database as follows:

### Primary Database chicago

```
LOG_ARCHIVE_DEST_STATE_2='ENABLE'
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'
LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'
LOG_ARCHIVE_DEST_3='SERVICE=boston ASYNC ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston'
```

This enables Oracle Data Guard to continue sending redo, asynchronously, to the terminal standby boston when it can no longer send the redo directly to the far sync instance chicagoFS. When the far sync instance becomes available again, Oracle Data Guard automatically resynchronizes the far sync instance chicagoFS and returns to the original configuration in which the primary sends redo to the far sync instance and the far sync instance forwards that redo to the terminal standby. When the synchronization is complete, the alternate destination (LOG\_ARCHIVE\_DEST\_3 in the preceding example) will again become dormant as the alternate.

## 84.3 Additional Configurations

This section provides examples of two additional far sync instance configurations. These examples describe variations that provide better data protection when you use far sync instances.

For continued protection after a role change, two far sync instances can be configured, one close to each database in the configuration, where only one is active at any time and the other one becomes active when a role change occurs between the primary database and the terminal standby database. This allows the configuration to remain in the desired protection mode regardless of which database is the primary database.

For more protection from system or network failures, two additional far sync instances can be configured that provide high availability for the active far sync instance. In this configuration one is the preferred active far sync instance and the other is the alternate far sync instance. Configuring an alternate far sync instance provides continued protection for the configuration if the preferred

far sync instance fails for some reason. The primary automatically starts shipping to the alternate far sync instance if it detects a failure at the preferred far sync instance. If the preferred far sync instance re-establishes itself, then the primary switches back to the preferred far sync instance and puts the alternate far sync instance back into the alternate state.

In these types of configurations, the primary uses only one of these far sync instances to redistribute redo at any given time.

### 84.3.1 Maintaining Protection After a Role Change

The configuration described inappropriate after a role transition where boston becomes the primary database and chicago becomes the terminal standby. The far sync instance chicagoFS would be too remote for boston to use it as a synchronous destination because the network latency between two sites is sufficiently large and would impact commit response times. To maintain the protection level of Maximum Availability for zero data loss, a second far sync instance close to boston must be established, in readiness for a future role transition event.

Using the same procedure as described in above create a far sync instance named bostonFS close to the standby database boston. The parameters that the new far sync instance bostonFS would then have.

#### Parameters Used to Set Up Protection After a Role Change

##### Far Sync Instance bostonFS

```
DB_UNIQUE_NAME=bostonFS
CONTROL_FILES='/arch2/bostonFS/control01.ctl'
FAL_SERVER=boston
LOG_FILE_NAME_CONVERT='boston','bostonFS'
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston,bostonFS)'
LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=bostonFS'
LOG_ARCHIVE_DEST_2='SERVICE=chicago ASYNC
VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=chicago'
```

The standby database boston would have been modified to have the following parameters set:

##### Physical Standby Database boston

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston,bostonFS)'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=bostonFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=bostonFS'
LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'
LOG_ARCHIVE_DEST_3='SERVICE=chicago ASYNC ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago'
```

Lastly, modify the parameters on chicago as follows:

##### Primary Database chicago

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston,bostonFS)'
FAL_SERVER='bostonFS',boston'
```

**Note:**

The far sync instance, bostonFS, will receive redo from boston and ship it to chicago only when boston is the primary database. However, even if boston is not the primary database, Oracle recommends keeping far sync instance bostonFS mounted in readiness for a future role transition.

### 84.3.2 Far Sync Instance High Availability

In the configurations described so far in this chapter, the ALTERNATE remote destination was setup directly between the two databases using asynchronous redo transport. This means that in the event of a failure of the active far sync instance, the protection level of the configuration falls back down to Maximum Performance, with potential data loss at failover time.

To maintain the protection level of Maximum Availability, a second far sync instance can be set up as the ALTERNATE to the far sync instance used by each database when it is the primary. Then, if the active far sync instance becomes unavailable, the primary database can automatically begin sending redo in synchronous mode to the alternate far sync instance, thereby maintaining the elevated protection level of Maximum Availability. As before, when the original far sync instance reestablishes itself, the primary will automatically resume using it as the active far sync destination, and the second local far sync instance will once again become the alternate.

These high availability far sync instances would be created using the same steps as given in above and then become the alternates to their local far sync instances instead of the terminal standby. When complete, chicago and boston would have their parameters configured as follows (assuming the names chicagoFS1 and bostonFS1 as the new far sync instance names).

#### Parameters Used to Set Up High Availability Far Sync Instances

##### Primary Database chicago

```

FAL_SERVER=bostonFS, bostonFS1, boston
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston,bostonFS,bostonFS1)'
LOG_ARCHIVE_DEST_STATE_2='ENABLE'
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'
LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'
LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC AFFIRM
ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS1'

```

##### Physical Standby boston

```

LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston,bostonFS,bostonFS1)'
LOG_ARCHIVE_DEST_STATE_2='ENABLE'
LOG_ARCHIVE_DEST_2='SERVICE=bostonFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=bostonFS'
LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'
LOG_ARCHIVE_DEST_3='SERVICE=bostonFS1 SYNC AFFIRM
ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=bostonFS1'

```

Oracle Data Guard will now be able to continue synchronously sending redo to a far sync instance, maintaining the required zero data loss protection mode of Maximum Availability

regardless of whether chicago or boston is the primary database. As before, when the failed far sync instance becomes available again, Oracle Data Guard will automatically resynchronize it and return to the original configuration in which the primary sends redo to the first far sync instance, who then forwards that redo to the terminal standby. When the synchronization is complete, the alternate destination (LOG\_ARCHIVE\_DEST\_3 in the preceding example) will again become dormant as the alternate.

## 84.4 Supported Protection Modes for Far Sync Instances

A far sync instance is supported in either maximum performance or maximum availability mode.

### 84.4.1 Far Sync Instances in Maximum Availability Mode Configurations

In maximum availability mode, the far sync instance is relatively close to the primary database to minimize network latency, and the primary services the far sync instance using SYNC transport.

**Note:**

There is no architectural limit to the distance that can separate the primary and far sync instance in maximum availability mode. The practical distance limit varies depending upon a given application's tolerance to the impact of network latency in a synchronous configuration. Also note that it is possible to reduce the performance impact for any given distance by using the new Oracle Data Guard FastSync feature (SYNC/NOAFFIRM).

Both SYNC/AFFIRM and SYNC/NOAFFIRM semantics are supported on the LOG\_ARCHIVE\_DEST\_n established at the primary for the far sync instance. See Section 6.1 for information about the trade-offs of using each one.

When a primary services a far sync instance using SYNC transport, all committed redo resides on disk at the far sync instance. This allows the far sync instance to use one of the terminal standby destinations for a no data loss failover if the primary database is lost.

The far sync instance uses ASYNC transport to ship the incoming redo to terminal standbys that can be much farther away. This extends no data loss protection to destinations that are too far away for a primary database to feasibly service directly with SYNC transport because of the degradation in transaction throughput that would result. This is a case where a far sync instance is beneficial even if there is only one standby destination in the configuration.

### 84.4.2 Far Sync Instances in Maximum Performance Mode Configurations

In maximum performance mode, the primary database services the far sync instance destination using ASYNC redo transport regardless of the physical distance between the primary and the far sync instance because high network latencies do not affect transaction throughput when a destination is serviced with ASYNC transport.

In maximum performance mode, a far sync instance can benefit Oracle Data Guard configurations that manage more than one remote destination. Although each ASYNC destination has a near-zero effect on primary database performance, if there are many remote destinations (for example, multiple Oracle Active Data Guard standbys that form a reader farm), then the effect can become measurable. When a far sync instance is used, there is zero incremental effect for each remote destination added to the configuration. Additionally, redo transport compression can also be offloaded to the far sync instance. When a far sync instance is used, the primary only has to service the far sync instance, which then services the rest of the configuration; the greater the number of destinations, the greater the performance benefit.

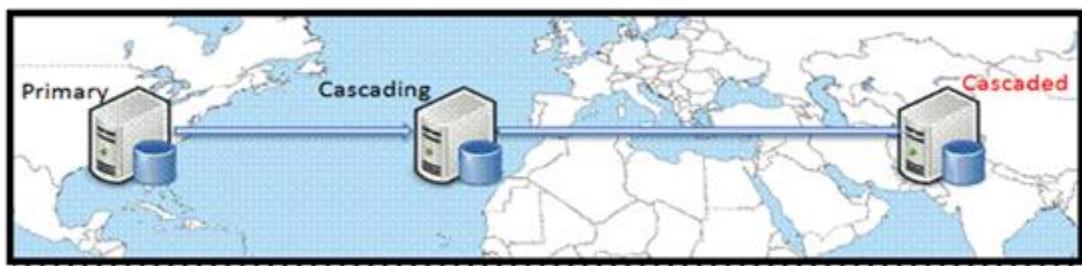
## 84.5 Real Time Cascading

### Introduction

Cascade standby database feature is introduced from 9i , Cascade standby database works same as standby database but it receives the redo logs from the first standby database but not primary database directly. In fact cascading standby database bypasses redo logs to the cascade standby. So there is much scope to reduce load , bandwidth utilization on primary database. But prior to 12c, Cascaded standby database will not be having fresh data same as cascading standby and now from 12c Cascaded standby database popularized with Real-Time Apply and now there is no more scope of LAG between primary and cascaded standby database. We will see more insight on the configuration, validation of Real-Time apply and usage of Data Guard Broker.

### Cascading Vs Cascade

Terminology looks very similar but there is lot of difference in terms of how they function, Let me clarify to you how exactly cascading and cascaded databases will be located from the below picture.



**Cascading standby** refers to standby database which receives redo first from primary database (or) Physical standby which have direct connection to the Primary database. Cascading standby database can work in Real-Time apply mode even prior to 12c. we can also name Cascading standby database as First or Local standby because its nearer to Primary database.

**Cascaded standby** will not be having direct connection to primary database instead of that cascading standby database serves the redo logs to the cascaded standby database, But cascaded standby have to wait for the log sequence until it is completely archived locally on cascading standby database(Prior to 12c).

### Real-Time Cascading

As we discussed in above statement prior to 12c when primary database redo is written to the standby redo logs of cascading standby database then the redo will be sent to cascaded database only after the standby redo log been archived, So there will always be one log sequence LAG with Cascading and cascaded standby database, but from 12c it is very much possible to forward the live redo as it is from cascading to cascaded standby database. Still it is up to you whether you prefer to go for Real-Time apply or not.

### Key Points for Real-Time Cascading

- Cascaded redo transport destination aliased as terminal destination
- Cascading can be either Real-Time apply or Non Real Time apply
- Cascading standby can cascade up to 30 terminal destinations
- Cascading standby should be either FARSYNC instance or Physical standby
- Cascading standby database can be in any protection mode and it doesn't matter to terminals
- To acquire Real-Time Apply on Cascaded the Standby redo logs of cascading should have been used at least once(ensure status is ACTIVE)
- Data Guard Broker is supported from 12c.
- Real-Time apply on Cascaded standby database comes with Active Data Guard license.

### Real-Time Cascading



**≤11g:** On Standby 1, redo is cascaded, after the standby redo log file has been archived locally.

**12c:** Standby 1 can cascade redo in real-time (as it is being written to the standby redo log file)

## 84.6 Other Features

### Other New Features

- The **USING CURRENT LOGFILE** clause is no longer required to start real-time apply.
- DML operations are allowed on **global temporary tables** on Oracle Active Data Guard standbys.
- The use of **sequences** in an Oracle Active Data Guard environment is now supported.
- When you perform a switchover from an Oracle RAC primary database to a physical standby database, it is no longer necessary to shut down all but one primary database instance.
- Application Continuity is supported for Oracle Data Guard switchovers to physical standby databases. It is also supported for fast-start failover to physical standbys in maximum availability data protection mode.



## 85. OEM Jobs & Events

### 85.1 Overview

The **DBMS\_JOB** package extensively is used to submit database jobs to run in the background, control the time or interval of a run, report failures, and much more. The problem with the package is that it can handle only PL/SQL code segments just anonymous blocks and stored program units. It cannot handle anything outside the database that is in an operating system command file or executable. To do so, we would have to resort to using an operating system scheduling utility such as cron in Unix or the AT command in Windows. Or, we could use a third-party tool, one that may even extend this functionality by providing a graphical user interface.

Even so, dbms\_job has a distinct advantage over these alternatives: it is active only when the database is up and running. If the database is down, the jobs don't run. A tool outside the database must manually check if the database is up and that can be difficult. Another advantage is that dbms\_job is internal to the database; hence we can access it via a database access utility such as SQL\*Plus.

The Oracle Database 10g Scheduler feature offers the best of all worlds: a job scheduler utility right inside the database that is sufficiently powerful to handle all types of jobs, not just PL/SQL code segments, and that can help us to create jobs either with or without associated programs and/or schedules. Best of all, it comes with the database at no additional cost. In this installment, we'll take a look at how it works.

### 85.2 Creating Jobs without Programs

Perhaps the concept can be best introduced through examples. Suppose we have created a shell script to move archived log files to a different filesystem as follows:

```
/home/arup/dbtools/move_arcs.sh
```

We can specify the OS executable directly without creating it as a program first.

```
begin
 dbms_scheduler.create_job
 (
 job_name => 'ARC_MOVE_2', schedule_name => 'EVERY_30_MINS',
 job_type => 'EXECUTABLE', job_action =>
 '/home/arup/dbtools/move_arcs.sh',
 enabled => true, comments => 'Move Archived Logs to a Different
 Directory'
);
end;
Similarly, we can create a job without a named schedule.
begin
 dbms_scheduler.create_job
 (
 job_name => 'ARC_MOVE_3', job_type => 'EXECUTABLE',
 job_action => '/home/arup/dbtools/move_arcs.sh',
 repeat_interval => 'FREQ=MINUTELY; INTERVAL=30', enabled => true,
 comments => 'Move Archived Logs to a Different Directory'
);
end;
```

One advantage of Scheduler over dbms\_job is pretty clear from our initial example: the ability to call OS utilities and programs, not just PL/SQL program units. This ability makes it the most comprehensive job management tool for managing Oracle Database and related jobs. However, we may have noted another, equally important advantage: the ability to define intervals in natural language. Note that in the above example we wanted our schedule to run every 30 minutes; hence

the parameter REPEAT\_INTERVAL is defined with a simple, English-like expression (not a PL/SQL one) :

```
'FREQ=MINUTELY; INTERVAL=30'
```

A more complex example may help convey this advantage even better. Suppose our production applications become most active at 7:00AM and 3:00PM. To collect system statistics, we want to run Statspack from Monday to Friday at 7:00AM and 3:00PM only. If we use DBMS\_JOB.SUBMIT to create a job, the NEXT\_DATE parameter will look something like this:

```
DECODE
(
 SIGN
 (
 15 - TO_CHAR(SYSDATE, 'HH24')
),
 1,
 TRUNC(SYSDATE) +15/24,
 TRUNC
 (
 SYSDATE +
 DECODE
 (
 TO_CHAR(SYSDATE, 'D'), 6, 3, 1
)
)
 +7/24
)
```

Is that code easy to understand? Not really.

Now let's see the equivalent job in DBMS\_SCHEDULER. The parameter REPEAT\_INTERVAL will be as simple as:

```
'FREQ=DAILY; BYDAY=MON,TUE,WED,THU,FRI; BYHOUR=7,15'
```

Furthermore, this parameter value can accept a variety of intervals, some of them very powerful. Here are some more examples:

Last Sunday of every month:

FREQ=MONTHLY; BYDAY=-1SUN

Every third Friday of the month:

FREQ=MONTHLY; BYDAY=3FRI

Every second Friday from the end of the month, not from the beginning:

FREQ=MONTHLY; BYDAY=-2FRI

The minus signs before the numbers indicate counting from the end, instead of the beginning.

What if we wanted to verify if the interval settings are correct? Wouldn't it be nice to see the various dates constructed from the calendar string? Well, we can get a preview of the calculation of next dates using the EVALUATE\_CALENDAR\_STRING procedure. Using the first example running Statspack every day from Monday through Friday at 7:00AM and 3:00PM we can check the accuracy of our interval string as follows:

```
set serveroutput on size 999999
declare
 l_start_date TIMESTAMP;
 l_next_date TIMESTAMP;
 l_return_date TIMESTAMP;
begin
 l_start_date := trunc(SYSTIMESTAMP);
 l_return_date := l_start_date;
 for ctr in 1..10 loop
 dbms_scheduler.evaluate_calendar_string(
 'FREQ=DAILY; BYDAY=MON,TUE,WED,THU,FRI; BYHOUR=7,15',

```

```

 l_start_date, l_return_date, l_next_date
);
 dbms_output.put_line('Next Run on: ' ||
 to_char(l_next_date,'mm/dd/yyyy hh24:mi:ss')
);
 l_return_date := l_next_date;
end loop;
end;
/

```

The output is:

```

Next Run on: 03/22/2004 07:00:00
Next Run on: 03/22/2004 15:00:00
Next Run on: 03/23/2004 07:00:00
Next Run on: 03/23/2004 15:00:00
Next Run on: 03/24/2004 07:00:00
Next Run on: 03/24/2004 15:00:00
Next Run on: 03/25/2004 07:00:00
Next Run on: 03/25/2004 15:00:00
Next Run on: 03/26/2004 07:00:00
Next Run on: 03/26/2004 15:00:00

```

This confirms that our settings are correct.

## 85.3 Associating Jobs with Programs

In the above case, we created a job independently of any program. Now let's create one that refers to an operating system utility or program, a schedule to specify how many times something should run, and then join the two to create a job.

First we need to make the database aware that our script is a program to be used in a job. To create this program, we must have the CREATE JOB privilege.

```

begin
 dbms_scheduler.create_program
 (
 program_name => 'MOVE_ARCS', program_type => 'EXECUTABLE',
 program_action => '/home/arup/dbtools/move_arcs.sh',
 enabled => TRUE, comments => 'Moving Archived Logs to Staging
Directory'
);
end;

```

Here we have created a named program unit, specified it as an executable, and noted what the program unit is called.

Next, we will create a named schedule to be run every 30 minutes called EVERY\_30\_MINS. We would do that with:

```

begin
 dbms_scheduler.create_schedule
 (
 schedule_name => 'EVERY_30_MINS',
 repeat_interval => 'FREQ=MINUTELY; INTERVAL=30',
 comments => 'Every 30-mins'
);
end;

```

Now that the program and schedule are created, we will associate the program to the schedule to create a job.

```

begin
 dbms_scheduler.create_job
 (

```

```

job_name => 'ARC_MOVE', program_name => 'MOVE_ARCS',
schedule_name => 'EVERY_30_MINS',
comments => 'Move Archived Logs to a Different Directory', enabled =>
TRUE
);
end;
/

```

This will create a job to be run every 30 minutes that executes the shell script move\_arcs.sh. It will be handled by the Scheduler feature inside the database no need for cron or the AT utility.

## 85.4 Classes, Plans, and Windows

A good job scheduling system worth its salt must support the ability to prioritize jobs. For instance, the statistics collection job suddenly goes into the OLTP workload window affecting performance there. To ensure the stats collection job doesn't consume resources affecting OLTP, we would use job classes, resource plans, and Scheduler Windows.

For example, while defining a job, we can make it part of a job class, which maps to a resource consumer group for allocation of resources. To do that, first we need to define a resource consumer group called, say, OLTP\_GROUP.

```

begin
 dbms_resource_manager.clear_pending_area();
 dbms_resource_manager.create_pending_area();
 dbms_resource_manager.create_consumer_group (
 consumer_group => 'oltp_group', comment => 'OLTP Activity Group'
);
 dbms_resource_manager.submit_pending_area();
end;
/

```

Next, we need to create a resource plan.

```

begin
 dbms_resource_manager.clear_pending_area();
 dbms_resource_manager.create_pending_area();
 dbms_resource_manager.create_plan('OLTP_PLAN', 'OLTP Database Activity
Plan');
 dbms_resource_manager.create_plan_directive(
 plan => 'OLTP_PLAN', group_or_subplan => 'OLTP_GROUP',
 comment => 'This is the OLTP Plan', cpu_p1 => 74, cpu_p2 => NULL,
 cpu_p3 => NULL, cpu_p4 => NULL, cpu_p5 => NULL, cpu_p6 => NULL,
 cpu_p7 => NULL, cpu_p8 => NULL, parallel_degree_limit_p1 => 4,
 active_sess_pool_p1 => NULL, queueing_p1 => NULL,
 switch_group => 'OTHER_GROUPS', switch_time => 10,
 switch_estimate => true, max_est_exec_time => 10, undo_pool => 600,
 max_idle_time => NULL, max_idle_blocker_time => NULL,
 switch_time_in_call => NULL
);
 dbms_resource_manager.create_plan_directive(
 plan => 'OLTP_PLAN', group_or_subplan => 'OTHER_GROUPS', comment =>
NULL,
 cpu_p1 => 20, cpu_p2 => NULL, cpu_p3 => NULL, cpu_p4 => NULL,
 cpu_p5 => NULL, cpu_p6 => NULL, cpu_p7 => NULL, cpu_p8 => NULL,
 parallel_degree_limit_p1 => 0, active_sess_pool_p1 => 0, queueing_p1
=> 0,
 switch_group => NULL, switch_time => NULL, switch_estimate => false,
 max_est_exec_time => 0, undo_pool => 10, max_idle_time => NULL,
 max_idle_blocker_time => NULL, switch_time_in_call => NULL
);
 dbms_resource_manager.submit_pending_area();
end;
/

```

Finally, we create a job class with the resource consumer group created earlier.

```
begin
 dbms_scheduler.create_job_class(
 job_class_name => 'OLTP_JOBS',
 logging_level => DBMS_SCHEDULER.LOGGING_FULL, log_history => 45,
 resource_consumer_group => 'OLTP_GROUP', comments => 'OLTP Related
Jobs'
);
end;
```

Let's examine the various parameters in this call. The parameter **LOGGING\_LEVEL** sets how much log data is tracked for the job class. The setting **LOGGING\_FULL** indicates that all activities on jobs in this class creation, deletion, run, alteration, and so on will be recorded in the logs. The logs can be seen from the view **DBA\_SCHEDULER\_JOB\_LOG** and are available for 45 days as specified in the parameter **LOG\_HISTORY** (the default being 30 days).

The resource consumer group associated with this class is also specified. The job classes can be seen from the view **DBA\_SCHEDULER\_JOB\_CLASSES**.

When we create a job, we can optionally associate a class to it. For instance, while creating **COLLECT\_STATS**, a job that collects optimizer statistics by executing a stored procedure **collect\_opt\_stats()**, we could have specified:

```
begin
 dbms_scheduler.create_job
 (
 job_name => 'COLLECT_STATS', job_type => 'STORED_PROCEDURE',
 job_action => 'collect_opt_stats', job_class => 'OLTP_JOBS',
 repeat_interval => 'FREQ=WEEKLY; INTERVAL=1', enabled => true,
 comments => 'Collect Optimizer Stats'
);
end;
```

This command will place the newly created job in the class **OLTP\_JOBS**, which is then governed by the resource plan **OLTP\_GROUP**, which will restrict how much CPU can be allocated to the process, the maximum number of executions before it is switched to a different group, the group to switch to, and so on.

Any job defined in this class will be governed by the same resource plan directive. This capability is particularly useful for preventing different types of jobs from taking over the resources of the system.

The Scheduler Window is a time frame with an associated resource plan used for activating that plan-thereby supporting different priorities for the jobs over a time frame. For example, some jobs, such as batch programs to update databases for real-time decision-support, need high priority during the day but become low priority at night (or vice-versa). We can implement this schedule by defining different resource plans and then activating those using Scheduler Windows.

## 85.5 Monitoring

After a job is issued, we can monitor its status from the view **DBA\_SCHEDULER\_JOB\_LOG**, where the column **STATUS** shows the current status of the job. If it shows **FAILED**, we can drill down further to find out the cause from the view **DBA\_SCHEDULER\_JOB\_RUN\_DETAILS**.

## 85.6 Administration

So far, we've discussed how to create several types of objects: programs, schedules, job classes, and jobs. What if we want to modify some of them to adjust to changing needs? Well, we can do that via APIs provided in the `DBMS_SCHEDULER` package.

From the Database tab of the Enterprise Manager 10g home page, click on the Administration link. This will bring up the Administration Screen, shown in Figure 1. All the Scheduler related tasks are found under the heading "Scheduler" to the bottom right-hand corner, shown inside a red ellipse in the figure.

The screenshot shows the Oracle Enterprise Manager 10g Administration page. The top navigation bar includes links for Setup, Preferences, Help, Logout, and Database. The Database tab is selected. The main content area is organized into several sections:

- Instance:** Memory Parameters, Undo Management, All Initialization Parameters.
- Storage:** Controlfiles, Tablespaces, Datafiles, Rollback Segments, Redo Log Groups, Archive Logs, Temporary Tablespace Groups.
- Security:** Users, Roles, Profiles.
- Enterprise Manager Administration:** Administrators, Notification Schedule, Blackouts.
- Schema:** Tables, Indexes, Views, Synonyms, Sequences, Database Links.
- Warehouse:** Cubes, OLAP Dimensions, Measure Folders.
- Configuration Management:** Last Collected Configuration, Database Usage Statistics.
- Workload:** Automatic Workload Repository, SQL Tuning Sets.
- Resource Manager:** Resource Monitors, Resource Consumer Group, Mappings.
- Scheduler:** Jobs, Schedules, Programs.

Figure 1: Administration page

All the tasks related to scheduler, such as creating, deleting, and maintaining jobs, can be easily accomplished through the hyper-linked task in this page. Let's see a few of these tasks. We created all these tasks earlier, so the clicking on the Jobs tab will show a screen similar to Figure



| Scheduled                                                                                                            |               |       |                                   |                                   |                   |               |
|----------------------------------------------------------------------------------------------------------------------|---------------|-------|-----------------------------------|-----------------------------------|-------------------|---------------|
| <a href="#">Edit</a> <a href="#">View</a> <a href="#">Delete</a> <a href="#">Run Now</a> <a href="#">Create Like</a> |               |       |                                   |                                   |                   |               |
| Select                                                                                                               | Name          | Owner | Scheduled Date                    | Last Run Date                     | Job Class         | Previous Runs |
| <input checked="" type="radio"/>                                                                                     | COLLECT_STATS | ARUP  | Mar 27, 2004<br>5:53:28 PM -05:00 | Mar 20, 2004<br>5:53:28 PM -05:00 | DEFAULT_JOB_CLASS | 1             |
| <input type="radio"/>                                                                                                | ARC_MOVE_3    | ARUP  | May 22, 2007<br>3:19:47 AM -05:00 | Mar 20, 2004<br>5:33:05 PM -05:00 | DEFAULT_JOB_CLASS | 2             |
| <input type="radio"/>                                                                                                | ARC_MOVE_2    | ARUP  | May 22, 2007<br>3:19:50 AM -05:00 | Mar 20, 2004<br>5:33:07 PM -05:00 | DEFAULT_JOB_CLASS | 2             |

Figure 2: Scheduled jobs

Clicking on the job COLLECT\_STATS allows us to modify its attributes. The screen shown in

Figure 3 shows up when we click on "Job Name."



Figure 3: Job parameters

As we can see, we can change parameters of the job as well as the schedule and options by clicking on the appropriate tabs. After all changes are made, we would press the button "Apply" to make the changes permanent. Before doing so, we may want to click the button marked "Show SQL", which shows the exact SQL statement that will be issued if for no other reason than to see what APIs are called, thereby enabling us to understand the workings behind the scene. We can also store the SQL in a script and execute it later, or store it as a template for the future.

## 86. Glossary

### **In-Memory:**

The In-Memory Column Store is an optional area of the SGA that stores copies of tables, partitions, and other database objects in a columnar format that is optimized for rapid scans.

The In-Memory Column Store stores copies of tables, partitions, and other database objects in the SGA. The In-Memory Column Store does not replace the database buffer cache. Instead, they complement each other so that both memory areas can store the same data in different formats. Rows stored in the In-Memory Column Store are divided into large memory regions in a columnar format. Within each region, a column resides separately in a contiguous area of memory.

### **Automatic Database Diagnostic Monitor (ADDM)**

This lets the Oracle Database diagnose its own performance and determine how identified problems could be resolved. It runs automatically after each AWR statistics capture, making the performance diagnostic data readily available.

### **Automatic Storage Management (ASM)**

A vertical integration of both the file system and the volume manager built specifically for Oracle database files. It extends the concept of stripe and mirrors everything to optimize performance, while removing the need for manual I/O tuning.

### **Automatic Storage Management Disk**

Storage is added and removed from Automatic Storage Management disk groups in units of Automatic Storage Management disks.

### **Automatic Storage Management File**

Oracle database file stored in an Automatic Storage Management disk group. When a file is created, certain file attributes are permanently set. Among these are its protection policy (parity, mirroring, or none) and its striping policy. Automatic Storage Management files are not visible from the operating system or its utilities, but they are visible to database instances, RMAN, and other Oracle-supplied tools.

### **Automatic Storage Management Instance**

An Oracle instance that mounts Automatic Storage Management disk groups and performs management functions necessary to make Automatic Storage Management files available to database instances. Automatic Storage Management instances do not mount databases.

### **Automatic Storage Management Template**

Collections of attributes used by Automatic Storage Management during file creation.

Templates simplify file creation by mapping complex file attribute specifications into a single name. A default template exists for each Oracle file type. Users can modify the attributes of the default templates or create new templates.

### **Automatic Undo Management Mode**

A mode of the database in which undo data is stored in a dedicated undo tablespace. Unlike manual undo management mode, the only undo management that we must perform is the creation of the undo tablespace. All other undo management is performed automatically.

### **Automatic Workload Repository (AWR)**

A built-in repository in every Oracle Database. At regular intervals, the Oracle Database makes a snapshot of all its vital statistics and workload information and stores them here.

## Background Process

Background processes consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. The background processes asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.

## Buffer Cache

The portion of the SGA that holds copies of Oracle data blocks. All user processes concurrently connected to the instance share access to the buffer cache. The buffers in the cache are organized in two lists: the dirty list and the least recently used (LRU) list. The dirty list holds dirty buffers, which contain data that has been modified but has not yet been written to disk. The least recently used (LRU) list holds free buffers (unmodified and available), pinned buffers (currently being accessed), and dirty buffers that have not yet been moved to the dirty list.

## Byte Semantics

The length of string is measured in bytes.

## Cache Recovery

The part of instance recovery where Oracle applies all committed and uncommitted changes in the redo log files to the affected data blocks. Also known as the rolling forward phase of instance recovery.

## Character Semantics

The length of string is measured in characters.

## 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.

## Client

In client/server architecture, the front-end database application, which interacts with a user through the keyboard, display, and pointing device such as a mouse. The client portion has no data access responsibilities. It concentrates on requesting, processing, and presenting data managed by the server portion.

## Client/Server architecture

Software architecture based on a separation of processing between two CPUs, one acting as the client in the transaction, requesting and receiving services, and the other as the server that provides services in a transaction.

## Cluster

Optional structure for storing table data. Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Because related rows are physically stored together, disk access time improves.

## Concurrency

Simultaneous access of the same data by many users. A multi-user database management system must provide adequate concurrency controls, so that data cannot be updated or changed improperly, compromising data integrity.

## Connection

Communication pathway between a user process and an Oracle instance.

**Database**

Collection of data that is treated as a unit. The purpose of a database is to store and retrieve related information.

**Database Buffer**

One of several types of memory structures that stores information within the system global area. Database buffers store the most recently used blocks of data.

**Database Buffer Cache**

Memory structure in the system global area that stores the most recently used blocks of data.

**Database Link**

A named schema object that describes a path from one database to another. Database links are implicitly used when a reference is made to a global object name in a distributed database.

**Data Block**

Smallest logical unit of data storage in an Oracle database. Also called logical blocks, Oracle blocks, or pages. One data block corresponds to a specific number of bytes of physical database space on disk.

**Data Integrity**

Business rules that dictate the standards for acceptable data. These rules are applied to a database by using integrity constraints and triggers to prevent the entry of invalid information into tables.

**Data Segment**

Each nonclustered table has a data segment. All of the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.

**Dedicated Server**

A database server configuration in which a server process handles requests for a single user process.

**Define Variables**

Variables defined (location, size, and datatype) to receive each fetched value.

**Disk Group**

One or more Automatic Storage Management disks managed as a logical unit. Automatic Storage Management disks can be added or dropped from a disk group while preserving the contents of the files in the group, and with only a minimal amount of automatically initiated I/O required to redistribute the data evenly. All I/O to a disk group is automatically spread across all the disks in the group.

**Distributed Processing**

Software architecture that uses more than one computer to divide the processing for a set of related jobs. Distributed processing reduces the processing load on a single computer.

**DDL**

Data definition language. Includes statements like CREATE/ALTER TABLE/INDEX, which define or change data structure.

**DML**

Data manipulation language. Includes statements like INSERT, UPDATE, and DELETE, which change data in tables.

**DOP**

The degree of parallelism of an operation.

**Enterprise Manager**

An Oracle system management tool that provides an integrated solution for centrally managing our heterogeneous environment. It combines a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools for managing Oracle products.

**Extent**

Second level of logical database storage. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information.

**Failure Group**

Administratively assigned sets of disks that share a common resource whose failure must be tolerated. Failure groups are used to determine which Automatic Storage Management disks to use for storing redundant copies of data.

**Indextype**

An object that registers a new indexing scheme by specifying the set of supported operators and routines that manage a domain index.

**Index Segment**

Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.

**Integrity Constraint**

Declarative method of defining a rule for a column of a table. Integrity constraints enforce the business rules associated with a database and prevent the entry of invalid information into tables.

**Logical Structures**

Logical structures of an Oracle database include tablespaces, schema objects, data blocks, extents, and segments. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

**LogMiner**

A utility that lets administrators use SQL to read, analyze, and interpret log files. It can view any redo log file, online or archived. The Oracle Enterprise Manager application Oracle LogMiner Viewer adds a GUI-based interface.

**Mean Time To Recover (MTTR)**

The desired time required to perform instance or media recovery on the database. For example, we may set 10 minutes as the goal for media recovery from a disk failure. A variety of factors influence MTTR for media recovery, including the speed of detection, the type of method used to perform media recovery, and the size of the database.

**Mounted Database**

An instance that is started and has the control file associated with the database open. We can mount a database without opening it; typically, we put the database in this state for maintenance or for restore and recovery operations.

**Object Type**

An object type consists of two parts: a spec and a body. The type body always depends on its type spec.

**Operator**

In memory management, the term operator refers to a data flow operator, such as a sort, hash join, or bitmap merge.

**Oracle XA**

The Oracle XA library is an external interface that allows global transactions to be coordinated by a transaction manager other than the Oracle database server.

**Partition**

A smaller and more manageable piece of a table or index.

**Priority Inversion**

Priority inversion occurs when a high priority job is run with lower amount of resources than a low priority job. Thus the expected priority is "inverted."

**Query Block**

A self-contained DML against a table. A query block can be a top-level DML or a subquery.

**Real Application Clusters (RAC)**

Option that allows multiple concurrent instances to share a single physical database.

**Recovery Manager (RMAN)**

A utility that backs up, restores, and recovers Oracle databases. We can use it with or without the central information repository called a recovery catalog. If we do not use a recovery catalog, RMAN uses the database's control file to store information necessary for backup and recovery operations. We can use RMAN in conjunction with a media manager to back up files to tertiary storage.

**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. If we run in an Oracle Real Application Clusters configuration, then we have multiple redo threads, one for each instance.

**Schema**

Collection of database objects, including logical structures such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. A schema has the name of the user who controls it.

**Segment**

Third level of logical database storage. A segment is a set of extents, each of which has been allocated for a specific data structure, and all of which are stored in the same tablespace.

**Sequence**

A sequence generates a serial list of unique numbers for numeric columns of a database's tables.

**Server**

In a client/server architecture, the computer that runs Oracle software and handles the functions required for concurrent, shared data access. The server receives and processes the SQL and PL/SQL statements that originate from client applications.

**Shared Server**

A database server configuration that allows many user processes to share a small number of server processes, minimizing the number of server processes and maximizing the use of available system resources.

**Standby Database**

A copy of a production database that we can use for disaster protection. We can update the standby database with archived redo logs from the production database in order to keep it current. If a disaster destroys the production database, we can activate the standby database and make it the new production database.

**Subtype**

In the hierarchy of user-defined datatypes, a subtype is always a dependent on its supertype.

**Synonym**

An alias for a table, view, materialized view, sequence, procedure, function, package, type, Java class schema object, user-defined object type, or another synonym.

**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 Global Area (SGA)**

A group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, then the data in the instance's SGA is shared among the users. Consequently, the SGA is sometimes referred to as the shared global area.

**Unicode**

A way of representing all the characters in all the languages in the world. Characters are defined as a sequence of codepoints, a base codepoint followed by any number of surrogates. There are 55K codepoints.

**Unicode column**

A column of type NCHAR, NVARCHAR2, or NCLOB guaranteed to hold Unicode.

**User process**

User processes execute the application or Oracle tool code.

**UTC**

Coordinated Universal Time, previously called Greenwich Mean Time, or GMT.

**View**

A view is a custom-tailored presentation of the data in one or more tables. A view can also be thought of as a "stored query." Views do not actually contain or store data; they derive their data from the tables on which they are based. Like tables, views can be queried, updated, inserted into, and deleted from, with some restrictions. All operations performed on a view affect its base tables.

**VOLUME -2: Multitenant Architecture**

Wilshire

## 87. Introduction to the Multitenant Architecture

Multitenancy refers to a principle in Software Architecture where a single instance of the software runs on a server, serving multiple tenants. A tenant is a group of users sharing the same view on the software they use. With a multitenant architecture a software application is designed to provide every tenant a dedicated share of the instance including its data, configuration, user management, tenant individual functionality and non-functional properties. Multitenancy contrasts with multi-instance architectures where separate software instances operate on behalf of different tenants.

Multitenancy is an important feature on cloud computing.

Large enterprises may use hundreds or thousands of databases. Often these databases run on different platforms on multiple physical servers. Because of improvements in hardware technology, especially the increase in the number of CPUs, servers are able to handle heavier workloads than before. A database may use only a fraction of the server hardware capacity. This approach wastes both hardware and human resources.

For example, 100 servers may have one database each, with each database using 10% of hardware resources and 10% of an administrator's time. A team of DBAs must manage the SGA, database files, accounts, security, and so on of each database separately, while system administrators must maintain 100 different computers.

A typical response to the management problem is to place multiple databases on each server. The problem is that the multiple database instances do not share background processes, system and process memory, or Oracle metadata. Another response is to logically separate the data into schemas or virtual private databases. The problem is that these virtual entities are difficult to manage, secure, and transport.

### Benefits

A new option for Oracle Database 12c, Oracle Multitenant delivers a new architecture that allows a multitenant container database to hold many pluggable databases. An existing database can simply be adopted with no application changes required. Oracle Multitenant fully complements other options, including Oracle Real Application Clusters and Oracle Active Data Guard.

The multitenant architecture enables an Oracle database to function as a multitenant container database (**CDB**) that includes zero, one, or many customer-created pluggable databases (**PDBs**). A PDB is a portable collection of schemas, schema objects, and non-schema objects that appears to an Oracle Net client as a non-CDB. All Oracle databases before Oracle Database 12c were non-CDBs.

The multitenant option represents one of the biggest architectural changes in the history of the Oracle database. The option introduced the concepts of the **Container Database (CDB)** and **Pluggable Database (PDB)**.

**Container Database (CDB)** : On the surface this seems very similar to a conventional Oracle database, as it contains most of the working parts you will be already familiar with (controlfiles, datafiles, undo, tempfiles, redo logs etc.). It also houses the data dictionary for those objects that are owned by the root container and those that are visible to all PDBs.

**Pluggable Database (PDB)** : Since the CDB contains most of the working parts for the database, the PDB only needs to contain information specific to itself. It does not need to worry about controlfiles, redo logs and undo etc. Instead it is just made up of datafiles and tempfiles to handle its own objects. This includes its own data dictionary, containing information about only those objects that are specific to the PDB.

### Database Consolidation

The process of consolidating data from multiple databases into one database on one computer is known as database consolidation.

Starting in Oracle Database 12c, the Oracle Multitenant option enables you to consolidate data and code without altering existing schemas or applications.

The PDB/non-CDB compatibility guarantee means that a PDB behaves the same as a non-CDB as seen from a client connecting with Oracle Net. The installation scheme for an application back end that runs against a non-CDB runs the same against a PDB and produces the same result. Also, the run-time behavior of client code that connects to the PDB containing the application back end is identical to the behavior of client code that connected to the non-CDB containing this back end.

Operations that act on an entire non-CDB act in the same way on an entire CDB, for example, when using Oracle Data Guard and database backup and recovery. Thus, the users, administrators, and developers of a non-CDB have substantially the same experience after the database has been consolidated.

### Manageability

The multitenant architecture has benefits beyond database consolidation. These benefits derive from storing the data *and* data dictionary metadata specific to a PDB in the PDB itself rather than storing all dictionary metadata in one place. By storing its own dictionary metadata, a PDB becomes easier to manage as a distinct unit, even when only one PDB resides in a CDB.

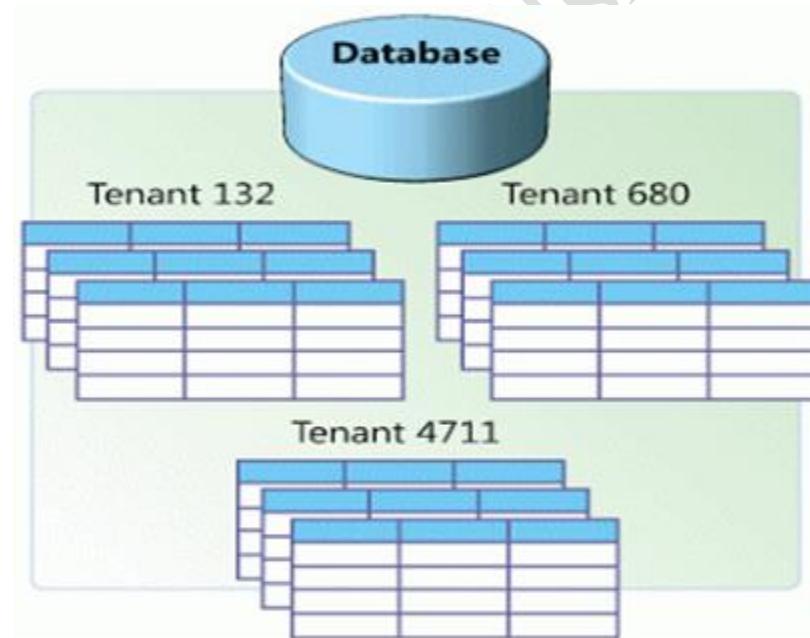
Benefits of data dictionary separation include the following:

Easier migration of data and code

For example, instead of upgrading a CDB from one database release to another, you can unplug a PDB from the existing CDB, and then plug it into a newly created CDB from a higher release.

Easier testing of applications

You can develop an application on a test PDB and, when it is ready for deployment, plug this PDB into the production CDB.



## 87.1 Containers in Multitenancy

A **CONTAINER** is a collection of schemas, objects, and related structures in a **multitenant container database (CDB)** that appears logically to an application as a separate database. Within a CDB, each container has a unique ID and name.

The root and every **pluggable database (PDB)** is considered a container. PDBs isolate data and operations so that from the perspective of a user or application, each PDB appears as if it were a traditional non-CDB.

**This section contains the following topics:**

The Root

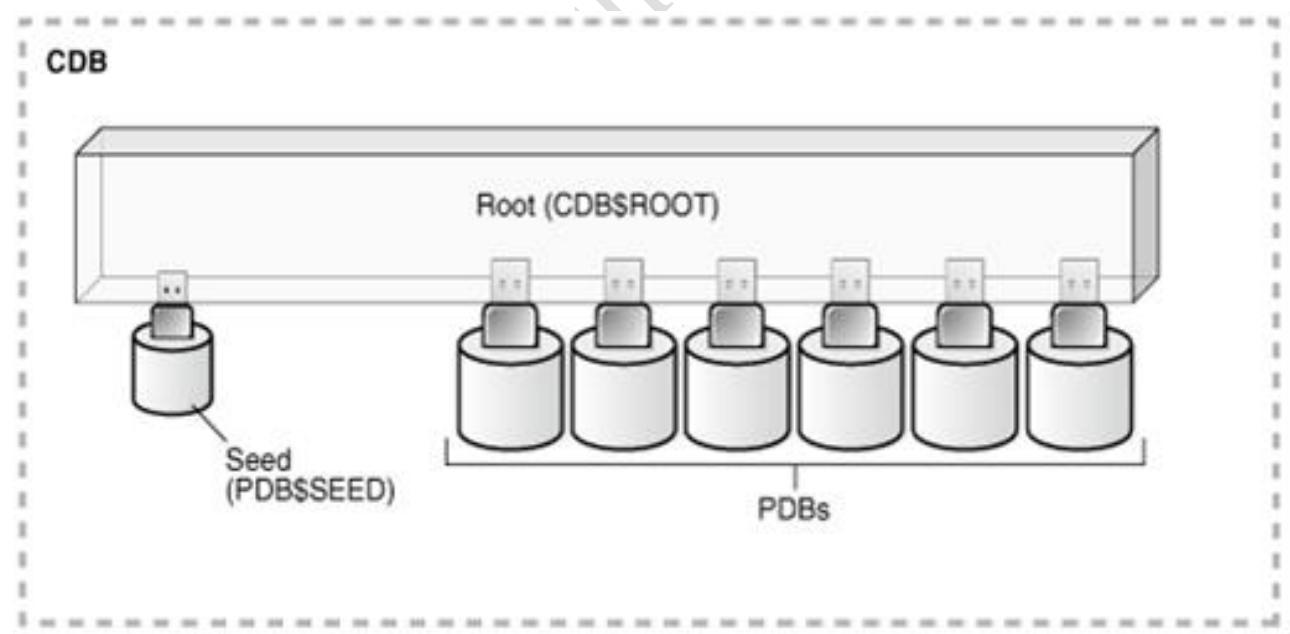
PDBs

Data Dictionary Architecture in a CDB

Current Container

Cross-Container Operations

CONTAINER



## 87.2 CONTAINER Database

### The ROOT

The **root** container, also called *the root*, is a collection of schemas, schema objects, and non-schema objects to which all PDBs belong. Every CDB has one and only one root container, which stores the system metadata required to manage PDBs. All PDBs belong to the root.

The name of the root is **CDB\$ROOT**.

The root does not store user data. Thus, you must not add user data to the root or modify system-supplied schemas in the root. However, you can create common users and roles for database administration

A common user with the necessary privileges can switch between PDBs.

### The SEED

The seed PDB is a system-supplied template that the CDB can use to create new PDBs.

The seed PDB is named **PDB\$SEED**. You cannot add or modify objects in **PDB\$SEED**.

### Pluggable Database

A Pluggable database (PDB) is a user-created entity that contains the data and code required for a specific set of features.

For example, a PDB can support a specific application, such as a human resources or sales application.

No PDBs exist at creation of the CDB. You add PDBs based on your requirement.

Currently Oracle 12c supports up to 253 Pluggable Databases in a single container.

### Commonality in a CDB

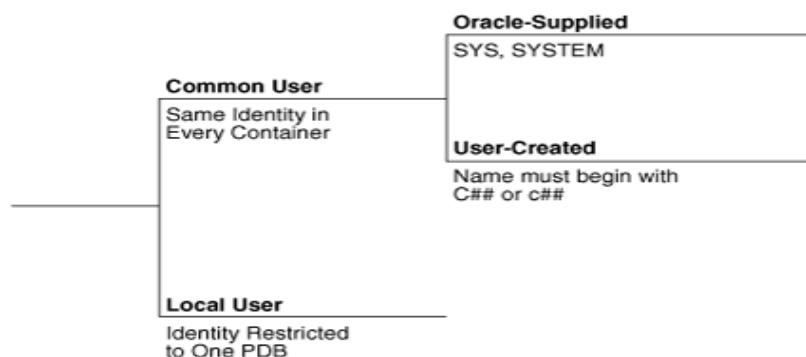
In a CDB, the basic principle of commonality is that a common phenomenon is the same in every existing and future container.

In a CDB, "common" means "common to all containers." In contrast, a local phenomenon is restricted to exactly one existing container.

A corollary to the principle of commonality is that only a common user can alter the existence of common phenomena.

Only a common user connected to the root can create, destroy, or modify CDB-wide attributes of a common user or role.

### Common and Local Users in a CDB



## Data Objects in a CDB

A **container data object** is a table or view containing data pertaining to multiple containers and possibly the CDB as a whole, along with mechanisms to restrict data visible to specific common users through such objects to one or more containers.

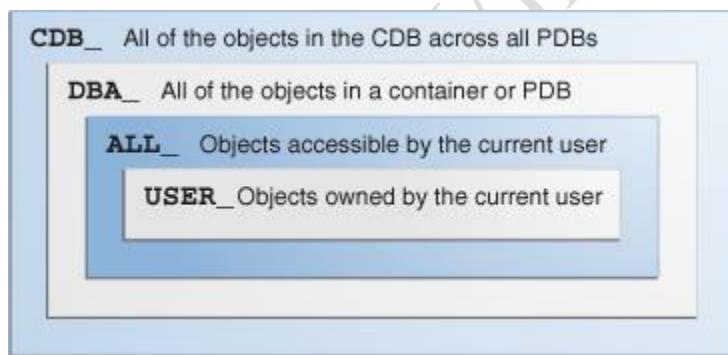
Examples of container data objects are Oracle-supplied views whose names begin with `V$` and `CDB_`.

All container data objects have a `CON_ID` column shows the meaning of the values for this column.

### Container ID Values

| Container ID | Rows pertain to             |
|--------------|-----------------------------|
| 0            | <b>Whole CDB or non-CDB</b> |
| 1            | <b>CDB\$ROOT</b>            |
| 2            | <b>PDB\$SEED</b>            |
| 3 – 257.     | <b>User Created PDBs</b>    |

In a CDB, for every `DBA_` view, a corresponding `CDB_` view exists. The owner of a `CDB_` view is the owner of the corresponding `DBA_` view. The following graphic shows the relationship among the different categories of dictionary views:



When the current container is the root, a common user can query `CDB_` views to see metadata for the root and for PDBs for which this user is privileged.

When the current container is a PDB, however, a user can view data dictionary information for the current PDB only.

To an application connected to a PDB, the data dictionary appears as it would for a non-CDB.

### Data Dictionary Storage in a CDB

The data dictionary that stores the metadata for the CDB as a whole is stored only in the **SYSTEM** tablespaces.

The data dictionary that stores the metadata for a specific PDB is stored in the self-contained tablespaces dedicated to this PDB.

The PDB tablespaces contain both the data and metadata for an application back end. Thus, each set of data dictionary tables is stored in its own dedicated set of tablespaces.

#### Current Container

For a given session, the current container is the one in which the session is running.

The current container can be the root (for common users only) or a PDB.

Each session has exactly one current container at any point in time.

As the data dictionary in each container is separate, Oracle Databases uses the data dictionary in the current container for name resolution and privilege authorization.

#### Cross-Container Operations

A cross-container operation is a DDL statement that affects any of the following:

The CDB itself

Multiple containers

Multiple entities such as common users or common roles that are represented in multiple containers

A container different from the one to which the user issuing the DDL statement is currently connected

Only a common user connected to the root can perform cross-container operations

Connections to Containers in a CDB

A CDB administrator with the appropriate privileges can connect to any container in the CDB.

The administrator can use either of the following techniques:

Use the **ALTER SESSION SET CONTAINER** statement, which is useful for both connection pooling and advanced CDB administration, to switch between containers.

SQL> ALTER SESSION SET CONTAINER = PDB\$SEED;

For example, a CDB administrator can connect to the root in one session, and then in the same session switch to a PDB.

In this case, the user requires the **SET CONTAINER** system privilege in the container.

Connect directly to a PDB.

In this case, the user requires the **CREATE SESSION** privilege in the container.

SQL> ALTER SESSION SET CONTAINER = PDB\$SEED;

## 87.3 Pluggable Databases

A pluggable database is a new construct whereby you can encapsulate a sub-set of Oracle data tables and indexes along with its associated metadata from the data dictionary. You start by creating a "root" instance database, called a container database (CDB). This container is used to hold many pluggable database "tenants" (PDB's), in a type of architectural separation.

The most important part of this separation is the "split" data dictionary, whereby everything needed for a PDB is self-contained in this sub-set of the metadata.

Unlike a traditional database where only one database can exist per instance, the Oracle 12c pluggable databases allow multiple databases within an instance. Oracle pluggable databases ease the movement into database consolidation because the data dictionary information (*obj\$*, *tab\$* and *source\$*) are independent of any container database. All PDB's within the container share a common LGWR process.

This container can then be populated with sub-sets of a schema and then be can be un-plugged from one instance and re-added to another instance quickly. Because the table definitions, index definitions, constraints and data all reside within the PDB, they can be easily copied and moved between instances and servers. Oracle has termed "multi-tenancy" to describe the process of creating a CDB that contains many "tenant" PDB's.

A **PDB** is a user-created set of schemas, objects, and related structures that appears logically to an application as a separate database.

Every PDB is owned by **SYSS**, which is a common user in the CDB regardless of which user created the PDB.

### Key Benefits:

Consolidate many PDBs onto a single platform.

Fast provisioning of a new PDB or of a clone of an existing PDB.

Fast redeployment, by unplug and plug, of an existing database to a new platform.

Patch or upgrade a single PDB by unplugging it and plugging it into a different CDB at a later version.

Separation of the content of one PDB from that of peer PDBs in the same CDB.

Separation of the duties of the application administrators of these PDBs.

### Key Capabilities:

You can have many PDBs inside a single container CDB.

PDB is backwards compatible with an ordinary pre-15.1 database.

PDB is transparent to applications – you don't change the client code or your database objects

A session sees only the single PDB it connects to

You can unplug a PDB from one CDB and plug it into another CDB

You can clone a PDB, both within the same CDB or from one CDB to another one

Resource Manager is extended with new PDB capabilities

The operation of PDBs as entities (creating, unplugging, plugging in, cloning, dropping and setting the Open\_Mode) are implemented as SQL statements.

Pluggable Database functionality is fully interoperable with all the database options.

All PDBs can be backed up at once, but recovered separately.

**Details:**

Each PDB has its own private data dictionary for customer-created database objects.

CDB as a whole has the data dictionary for the Oracle-supplied system each data dictionary defines its own namespace.

There is global Data Dictionary (CDB level) and local one (PDB level).

There is a new split data dictionary architecture that allows a PDB to be quickly unplugged from one CDB and plugged into a different CDB

Each PDB sees a read-only definition of the Oracle-supplied system.

There are global database initialization parameters on CDB level and local ones.

PDB parameters belong only to a particular PDB and will be persistent even after you unplug the PDB.

Database users can be global (CDB) or local (PDB only).

SYS and SYSTEM users exist in both DBs right at the beginning.

If you create a new user in CDB you will see it in PDB also.

If you create a new user on PDB level it will stay local.

Redo logs and Undo tablespace are only global (on CDB level).

Temporary tablespaces can be global or local.

Data Guard acts on the CDB as a whole.

RMAN scheduled backups are done for the CDB as a whole.

We can backup a selected PDB whenever we want to specific point-in-time.

An application connects, with no code changes, to a PDB.

The service named int the string specifies the destination PDB.

The system administrator connects to the CDB.

A PDB allows clear declarative definition of an application.

One PDB knows nothing of the other PDBs with the same CDB.

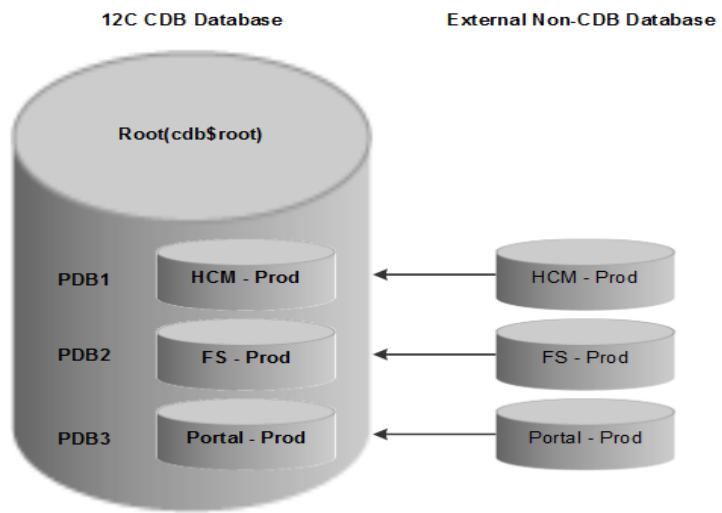
Each PDB is a hermetically sealed container which ensures new level of DB independence and robust security.

The service will be started as a side-effect of creating the PDB. The service has the same name as the PDB and its metadata is recorded inside the PDB.

Sessions will be typically created by authorizing as a user that cannot change the current container.

Each PDBS will denote the service within which it is defined as the initial current container. Use the normal methods to create, maintain, and drop additional services in a PDB.

A PDB's default service must not be dropped. The only way to establish a session whose initial container is a PDB is to specify a service.



## 88. User Management

The multitenant option introduced in Oracle Database 12c allows a single container database (CDB) to host multiple separate pluggable databases (PDB).

### 88.1 Introduction

When connected to a multitenant database the management of users and privileges is a little different to traditional Oracle environments.

### 88.2 Create Users

In multitenant environments there are two types of users.

#### Common User:

The user is present in all containers (root and all PDBs).

#### Local User:

The user is only present in a specific PDB.

The same username can be present in multiple PDBs, but they are unrelated.

#### 88.2.1 Create Common Users

The following requirements must be met before creating a common user.

The current container must be the root container.

You must be connected to a Common user with the **CREATE USER** privilege.

The username for the common user must be prefixed with "C##" or "c##" and contain only ASCII or EBCDIC characters.

The username must be unique across all containers.

You can either specify the **CONTAINER=ALL** clause, or omit it, as this is the default setting when the current container is the root.

#### 88.2.2 Create Local Users

When creating a local user the following requirements must all be met.

You must be connected to a user with the **CREATE USER** privilege.

The username for the local user must not be prefixed with "C##" or "c##".

The username must be unique within the PDB.

You can either specify the **CONTAINER=CURRENT** clause, or omit it, as this is the default setting when the current container is a PDB.

The following example shows how to create local users with and without the **CONTAINER** clause from the root container.

## 88.3 Create Roles

The multitenant option introduced in Oracle Database 12c allows creating roles in two types. They are:

Common Roles

Local Roles

### 88.3.1 Create Common Roles

All Oracle-supplied roles are common and therefore available in the root container and all PDBs. Common roles can be created, provided the following conditions are met.

You must be connected to a common user with **CREATE ROLE** and the **SET CONTAINER** privileges granted commonly.

The current container must be the root container.

The role name for the common role must be prefixed with "C##" or "c##" and contain only ASCII or EBCDIC characters.

The role name must be unique across all containers.

The role is created with the **CONTAINER=ALL** clause

The following example shows how to create a common role and grant it to a common and local user.

### 88.3.2 Create Local Roles

Local roles are created in a similar manner to pre-12c databases. Each PDB can have roles with matching names, since the scope of a local role is limited to the current PDB. The following conditions must be met.

You must be connected to a user with the **CREATE ROLE** privilege.

While connected to a common user, the container must be set to the local PDB.

The role name for the local role must not be prefixed with "C##" or "c##".

The role name must be unique within the PDB.

The following example shows how to create local a role and grant it to a common user and a local user.

## 89. Data Pump Enhancements in 12c

A multitenant container database (CDB) is an Oracle database that includes zero, one, or many user-created pluggable databases (PDBs). A PDB is a portable set of schemas, schema objects, and non-schema objects that appear to an Oracle Net client as a non-CDB. A non-CDB is an Oracle database that is not a CDB.

You can use Data Pump to migrate all, or portions of, a database from a non-CDB into a PDB, between PDBs within the same or different CDBs, and from a PDB into a non-CDB. In general, using Data Pump with PDBs is identical to using Data Pump with a non-CDB.

**Note:** In Oracle Database 12c Data Pump does not support any CDB-wide operations.

Data pump issues the following warning if you are connected to the root or seed database of a CDB.

ORA-39357: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database.

If you want to specify a particular PDB for the export/import operation, then on the Data Pump command line, you can supply a connect identifier in the connect string when you start Data Pump.

The following requirements when using Data Pump to move data into a CDB:

To administer a multitenant environment, you must have the `CDB_DBA` role.

Full database exports from Oracle Database 11.2.0.2 and earlier may be imported into Oracle Database 12c (CDB or non-CDB). However, Oracle recommends the source database first be upgraded to Oracle Database 11g release 2 (11.2.0.3 or later) so that information about registered options and components is included in the export.

When migrating Oracle Database 11g release 2 (11.2.0.3 or later) to a CDB (or to a non-CDB) using either full database export or full transportable database export, you must set the Data Pump Export parameter `VERSION=12` in order to generate a dump file that is ready for import into Oracle Database 12c. If you do not set `VERSION=12`, then the export file that is generated will not contain complete information about registered database options and components.

Network-based full transportable imports require use of `TRANSPORT_DATAFILES=datafile_name` and `TRANSPORTABLE=ALWAYS` parameters. When the source database is Oracle Database 11g release 11.2.0.3 or later, but earlier than Oracle Database 12c Release 1, the `VERSION=12` parameter is also required.

File-based full transportable imports only require use of `TRANSPORT_DATAFILES=datafile_name` parameter. Data Pump import infers the presence of `TRANSPORTABLE=ALWAYS` and `FULL=YES` parameters.

The default Data Pump directory object, `DATA_PUMP_DIR`, does not work with PDBs. You must define an explicit directory object within the PDB that you are exporting or importing.

## 90. RMAN Enhancements in 12c

Oracle Database 12c has new enhancements and additions in Recovery Manager (RMAN). The Recovery Manager continues to enhance and extend the reliability, efficiency, and availability of Oracle Database Backup and Recovery.

The multitenant container database (CDB) and pluggable databases (PDB) are introduced in Oracle 12c, and RMAN provides full support for backup and recovery. Using RMAN you can back up an entire container database or individual pluggable databases and also can perform point-in-time recovery. But it is recommended that you turn on control file auto backup. Otherwise point-in-time recovery for pluggable databases may not work efficiently when you need to undo data file additions or deletions.

The multitenant architecture manages many databases as one and retains the isolation, resource control of each database. This will help to manage both infrastructure and human resources effectively.

The multitenant option brings with it a number of changes to the way we approach backup and recovery. This section provides a top-level view of what is available for basic backup and recovery of CDBs and PDBs.

Let's take a look at some of the characteristics of a CDB

There is a separate **SYSTEM** and **SYSAUX** tablespace for the root container of the **CDB** and each **PDB**

There is only one **UNDO** tablespace for the entire **CDB**

There is only one set of **control files** and **online redo log files** for the entire **CDB**. Individual **PDB's** have their own data files (which contain the user data), but do not have distinct redo log or control files.

We can create on **default temporary tablespace** for the entire **CDB** or each **PDB** can have its own additional temporary tablespaces

There are single network administration files like **listener.ora**, **tnsnames.ora** and **sqlnet.ora** file for an entire **CDB**. All of the **PDBs** in the **CDB** use the same files.

The below are few important enhancements

SQL Interface Improvements

SYSBACKUP Privilege

Support for multitenant container and pluggable databases

DUPLICATE enhancements

SQL Interface Improvements

In Oracle 12c, you can run SQL commands in RMAN without preceding the command with the SQL keyword. You also no longer need to enclose the SQL command in quotes.

The RMAN DESCRIBE provides the same functionality of SQL\*Plus DESCRIBE:

You can run DDL/DML Commands from RMAN Command prompt, but note that in order to insert you need to use.

The user can SHUTDOWN/STARTUP the database and also can use ALTER commands.

With this new SQL Interface improvement's, users don't need to switch between RMAN and SQL\* Plus prompts during Backup & Recovery, administration...etc.

## 90.1 PDB subset Cloning

The 11.1.0.2 patch set introduced the concept of PDB subset cloning, which allows a subset of all the tablespaces in a PDB to be cloned. Excluding tablespaces can be useful when you want to build a PDB to test a specific piece of functionality, which doesn't require the whole database. It is also useful when splitting instances that were used for consolidation into their individual functional areas.

The **USER\_TABLESPACES** clause allows a user to specify which tablespaces need to be available in the new pluggable database (PDB).

An example of the application of this clause is a case where a customer is migrating from a non-multitenant container database (CDB) where schema-based consolidation was used to separate data belonging to multiple tenants to a CDB where data belonging to each tenant is kept in a separate PDB.

The **USER\_TABLESPACES** clause helps to create one PDB for each schema in the non-CDB.

This powerful clause helps convert cumbersome schema-based consolidations to more agile and efficient pluggable databases.

PDB subset cloning is made possible using the **USER\_TABLESPACES** clause, which allows you to specify the user-defined tablespaces to be included in the clone in one of several ways.

One or more named tablespaces in a comma separated list.

**NONE** : No user-defined tablespaces are included in the clone.

**ALL** : All user-defined tablespaces are included in the clone. This is the same as omitting the clause completely.

**ALL EXCEPT** : Exclude one or more named user-defined tablespaces as a comma separated list.

## 90.2 PDB Metadata Clone

The 11.1.0.2 patch set introduced the ability to do a metadata-only clone. Adding the **NO DATA** clause when cloning a PDB signifies that only the metadata for the user-created objects should be cloned, not the data in the tables and indexes.

An administrator can now create a clone of a pluggable database only with the data model definition. The dictionary data in the source is copied as is but all user-created table and index data from the source is discarded.

This feature enhances cloning functionality and facilitates rapid provisioning of development environments.

Perform a metadata-only clone of the PDB using the **NO DATA** clause.

### 90.2.1 Restrictions

The **NO DATA** clause is only valid if the source PDB doesn't contain any of the following.

Index-organized tables

Advanced Queue (AQ) tables

Clustered tables

Table clusters

If it does, you will get the following type of error.

ORA-56153: Unable to create pluggable database with no data

## 90.3 PDB Remote Clone

The prerequisites for cloning a remote PDB or non-CDB are very similar.

The user in the local database must have the `CREATE PLUGGABLE DATABASE` privilege in the root container.

The remote database (PDB or non-CDB) must be open in read-only mode.

The local database must have a database link to the remote database. If the remote database is a PDB, the database link can point to the remote CDB using a common user, or the PDB using a local or common user.

The user in the remote database that the database link connects to must have the `CREATE PLUGGABLE DATABASE` privilege.

The local and remote databases must have the same endian options installed and character sets.

When cloning from a non-CDB, both the local and remote databases must be using version 15.1.0.2 or higher.