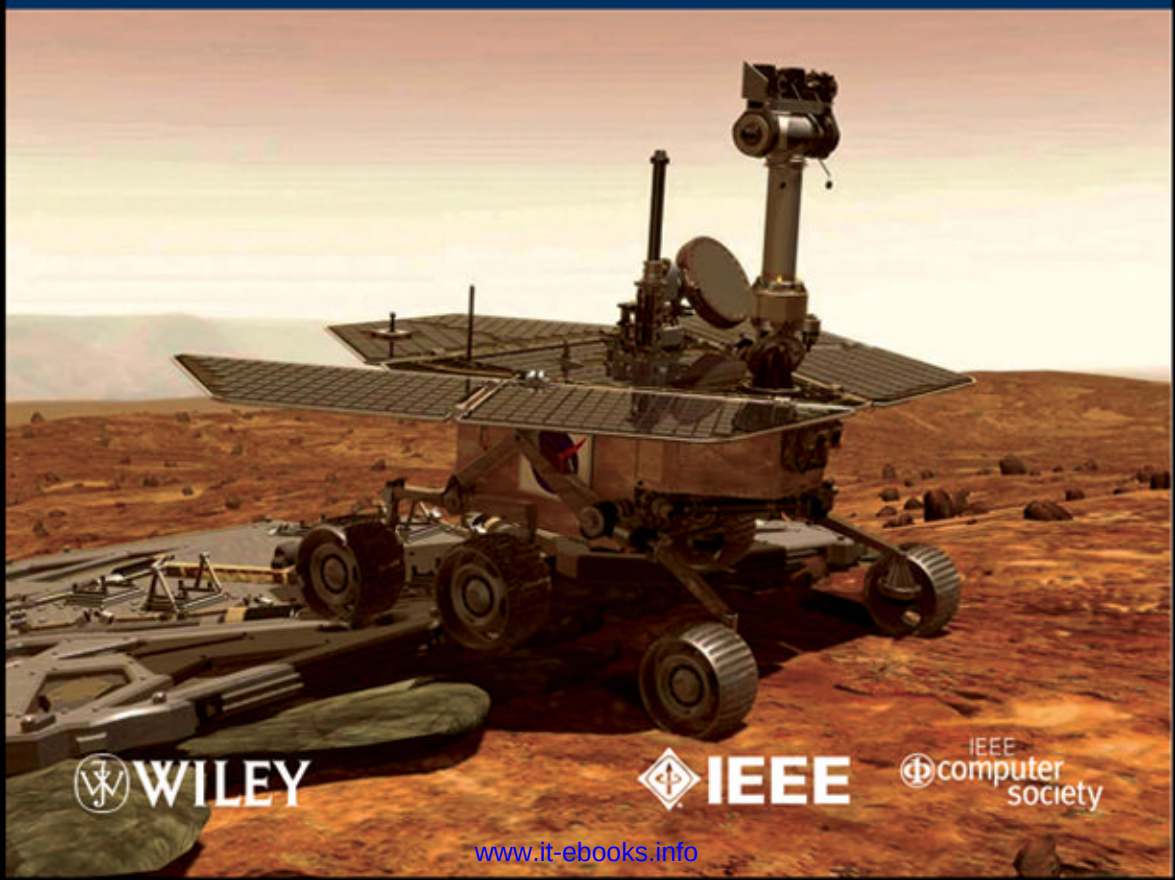


Quantitative Software Engineering Series • Lawrence Bernstein, Series Editor

Oracle Database Performance and Scalability

A Quantitative Approach

HENRY H. LIU



 **WILEY**

 **IEEE**

 IEEE
computer
society

www.it-ebooks.info

*Oracle Database
Performance
and Scalability*

Quantitative Software Engineering Series

The Quantitative Engineering Series focuses on the convergence of systems engineering with emphasis on quantitative engineering trade-off analysis. Each title brings the principles and theory of programming in-the-large and industrial strength software into focus.

This practical series helps software developers, software engineers, systems engineers, and graduate students understand and benefit from this convergence through the unique weaving of software engineering case histories, quantitative analysis, and technology into the project effort. You will find each publication reinforces the series goal of assisting the reader with producing useful, well-engineered software systems.

Series Editor: **Lawrence Bernstein**

Professor Bernstein is currently an Industry Research Professor at the Stevens Institute of Technology. He previously pursued a distinguished executive career at Bell Laboratories. He is a fellow of the IEEE and ACM.

Trustworthy Systems for Quantitative Software Engineering / Larry Bernstein and C.M. Yuhas

Software Measurement and Estimation: A Practical Approach / Linda M. Laird and M. Carol Brennan

World Wide Web Application Engineering and Implementation / Steven A. Gabarro

Software Performance and Scalability / Henry H. Liu

Managing the Development of Software-Intensive Systems / James McDonald

Trustworthy Compilers / Vladimir O. Safonov

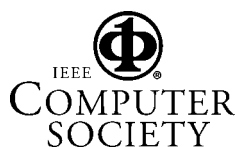
Oracle Database Performance and Scalability: A Quantitative Approach / Henry H. Liu

Enterprise Software Architecture and Design: Entities, Services and Resources / Dominic Duggan

Oracle Database Performance and Scalability

A Quantitative Approach

Henry H. Liu



A JOHN WILEY & SONS, INC., PUBLICATION

Copyright © 2012 by John Wiley & Sons, Inc. All rights reserved

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Liu, Henry H.

Oracle database performance and scalability : a quantitative approach /Henry H. Liu.

p. cm.

ISBN 978-1-118-05699-8 (cloth)

1. Oracle (Computer file) 2. Database management. I. Title.

QA76.9.D3L5945 2012

005.75'65--dc23

2011017552

Printed in the United States of America

oBook ISBN: 978-1-118-13553-2

ePDF ISBN: 978-1-118-13549-5

ePub ISBN: 978-1-118-13551-8

eMobi ISBN: 978-1-118-13550-1

10 9 8 7 6 5 4 3 2 1

To My Family

Contents

PREFACE **xxv**

Why This Book / xxv
Who This Book is For / xxvi
How This Book is Organized / xxvii
Software and Hardware / xxviii
How to Use This Book / xxix
How to Reach The Author / xxxi

ACKNOWLEDGMENTS **xxxiii**

INTRODUCTION **1**

Features of Oracle / 2
Objectives / 4
Conventions / 5
Performance versus Scalability / 6

PART 1 GETTING STARTED WITH ORACLE **7**

1 Basic Concepts **9**

- 1.1 Standard versus Flavored SQLS / 10
- 1.2 Relational versus Object-Oriented Databases / 11

1.3	An Instance versus a Database / 11	
1.4	Summary / 12	
	Recommended Reading / 12	
	Exercises / 12	
2	Installing Oracle Software	14
2.1	Installing Oracle 11g Server Software / 15	
2.2	Configuring a Listener / 18	
2.3	Creating an Oracle Database / 18	
2.4	Installing Oracle 11g Client Software / 28	
2.5	Oracle Grid Control versus DB Control / 31	
2.6	Summary / 33	
	Recommended Reading / 33	
	Exercises / 33	
3	Options for Accessing an Oracle Server	34
3.1	A Command Line Interface (CLI) versus a GUI-Based Console / 35	
3.2	The Oracle Enterprise Manager Java Console (OEMJC) / 37	
3.3	Using the SQL*Plus Tool / 40	
3.4	Oracle Enterprise Manager DBConsole / 42	
3.5	Other Tools for Developers / 43	
3.6	Case Study: Creating ER Diagrams with Visio via ODBC / 44	
3.7	Case Study: Accessing Oracle in Java via JDBC / 47	
3.8	Summary / 49	
	Recommended Reading / 50	
	Exercises / 50	
4	A Quick Tour of an Oracle Server	52
4.1	New Oracle Schemas Beyond “Scott” / 53	
4.2	Oracle Users versus Schemas / 54	
4.3	Tablespaces, Segments, Extents, and Data Blocks / 56	
4.4	Tables, Indexes and Index Types for Structured Data / 57	
4.5	Domain and LOB Index Types for Unstructured Data / 65	
4.6	Views, Materialized Views, and Synonyms / 68	
4.7	Stored Procedures, Functions, and Triggers / 68	
4.8	Referential Integrity with Foreign Keys / 71	
4.9	Summary / 73	

Recommended Reading / 73

Exercises / 74

PART 2 ORACLE ARCHITECTURE FROM PERFORMANCE AND SCALABILITY PERSPECTIVES 75

5 Understanding Oracle Architecture 79

5.1 The Version History of Oracle / 80

5.2 Oracle Processes / 82

5.3 Oracle Memory Areas / 87

5.4 Dedicated versus Shared Oracle Server Architecture / 89

5.5 Performance Sensitive Initialization Parameters / 91

5.6 Oracle Static Data Dictionary Views / 94

5.7 Oracle Dynamic Performance (V\$) Views / 95

5.8 Summary / 98

Recommended Reading / 98

Exercises / 99

6 Oracle 10g Memory Management 101

6.1 SGA Sub-Areas / 102

6.2 SGA Sizing: Automatic Shared Memory Management (ASMM) / 104

6.3 PGA Sizing: PGA_AGGREGATE_TARGET / 106

6.4 Summary / 108

Recommended Reading / 109

Exercises / 110

7 Oracle 11g Memory Management 111

7.1 Automatic Memory Management (AMM) / 112

7.2 Memory Sizing Options Configurable at Database Creation Time / 112

7.3 Checking Memory Management and Usage Distribution at Run Time / 113

7.4 Summary / 115

Recommended Reading / 115

Exercises / 115

8 Oracle Storage Structure 116

8.1 Overview / 117

8.2 Managing Tablespaces / 119

8.3	Managing Data Files / 122	
8.4	Managing Redo Logs / 124	
8.5	Summary / 125	
	Recommended Reading / 125	
	Exercises / 126	
9	Oracle Wait Interface (OWI)	127
9.1	Ratio-based versus OWI-based Oracle Performance Tuning Methodologies / 128	
9.2	Wait Event—The Core Concept of OWI / 130	
9.3	Classification of Wait Events from OWI / 131	
9.4	The Other Part (CPU Time) of the Equation Elapsed Time = CPU Time + Wait Time / 134	
9.5	AWR as a Compass to Tuning Oracle Performance and Scalability / 136	
9.6	Summary / 137	
	Recommended Reading / 137	
	Exercises / 138	
10	Oracle Data Consistency and Concurrency	139
10.1	Select ... for Update Statement / 140	
10.2	ACID Properties of Transactions / 141	
10.3	Read Phenomena and Data Inconsistencies / 143	
10.4	Oracle Isolation Levels / 145	
10.5	Multi-Version Concurrency Control (MVCC) and Read Consistency / 145	
10.6	Oracle Locks / 146	
10.7	Lock Escalations versus Conversions / 149	
10.8	Oracle Latches / 149	
10.9	Oracle Enqueues / 150	
10.10	Deadlocks / 150	
10.11	Taking Advantage of Oracle's Scalable Concurrency Model / 151	
10.12	Case Study: A JDBC Example / 152	
10.13	Summary / 158	
	Recommended Reading / 159	
	Exercises / 159	

11 Anatomy of an Oracle Automatic Workload Repository (AWR) Report	161
11.1 Importance of Performance Statistics /	162
11.2 AWR Report Header /	165
11.3 Report Summary /	166
11.3.1 Cache Sizes /	166
11.3.2 Load Profile /	167
11.3.3 Instance Efficiency Percentages (Target 100%) /	169
11.3.4 Shared Pool Statistics /	170
11.3.5 Top Five Timed Events /	170
11.4 Main Report /	171
11.5 Wait Events Statistics /	172
11.5.1 Time Model Statistics /	173
11.5.2 Wait Class /	174
11.5.3 Wait Events /	174
11.5.4 Background Wait Events /	176
11.5.5 Operating System Statistics /	176
11.5.6 Service Statistics /	177
11.5.7 Service Wait Class Stats /	178
11.6 SQL Statistics /	178
11.6.1 SQL ordered by Elapsed Time /	179
11.6.2 SQL ordered by CPU Time /	180
11.6.3 SQL ordered by Gets /	180
11.6.4 SQL ordered by Reads /	181
11.6.5 SQL ordered by Executions /	182
11.6.6 SQL ordered by Parse Calls /	183
11.6.7 SQL ordered by Sharable Memory /	183
11.6.8 SQL ordered by Version Count /	183
11.6.9 Complete List of SQL Text /	184
11.7 Instance Activity Statistics /	185
11.7.1 Instance Activity Stats /	185
11.7.2 Instance Activity Stats—Absolute Values /	196
11.7.3 Instance Activity Stats—Thread Activity /	197
11.8 IO Stats /	197
11.8.1 Tablespace IO Stats /	198
11.8.2 File IO Stats /	198
11.9 Buffer Pool Statistics /	199
11.10 Advisory Statistics /	199
11.10.1 Instance Recovery Stats /	200
11.10.2 Buffer Pool Advisory /	200
11.10.3 PGA Aggr Summary /	201
11.10.4 PGA Aggr Target Stats /	202

11.10.5	PGA Aggr Target Histogram / 202
11.10.6	PGA Memory Advisory / 203
11.10.7	Shared Pool Advisory / 204
11.10.8	SGA Target Advisory / 204
11.10.9	Streams Pool Advisory / 205
11.10.10	Java Pool Advisory / 205
11.11	Wait Statistics / 206
11.12	Undo Statistics / 207
11.13	Latch Statistics / 208
11.13.1	Latch Activity / 208
11.13.2	Latch Sleep Breakdown / 213
11.13.3	Latch Miss Sources / 214
11.13.4	Parent and Child Latch Statistics / 215
11.14	Segment Statistics / 215
11.14.1	Segments by Logical Reads / 215
11.14.2	Segments by Physical Reads / 216
11.14.3	Segments by Row Lock Waits / 217
11.14.4	Segments by ITL Waits / 217
11.14.5	Segments by Buffer Busy Waits / 217
11.15	Dictionary Cache Stats / 218
11.16	Library Cache Activity / 219
11.17	Memory Statistics / 219
11.17.1	Process Memory Summary / 219
11.17.2	SGA Memory Summary / 220
11.17.3	SGA Breakdown Difference / 221
11.18	Streams Statistics / 222
11.19	Resource Limit Stats / 224
11.20	init.ora Parameters / 224
11.21	Summary / 225
	Recommended Reading / 225
	Exercises / 226

12 Oracle Advanced Features and Options

227

12.1	Oracle 8i New Features / 227
12.1.1	Java / 228
12.1.2	Oracle interMedia, Spatial, Time Series, and Visual Image Retrieval / 229
12.1.3	Oracle Parallel Server / 230
12.1.4	Optimizer Plan Stability / 230
12.1.5	Locally Managed Tablespaces / 230
12.1.6	Online Index Creation and Rebuild / 231
12.1.7	Online Read-Only Tablespaces / 231

- 12.1.8 Temporary Tables / 231
- 12.1.9 Non-Blocking OCI (Oracle Call Interface) / 231
- 12.1.10 Function-Based Indexes / 232
- 12.1.11 Logical ROWIDs / 232
- 12.1.12 Enhanced Partitioning / 232
- 12.1.13 Connection Load Balancing / 233
- 12.1.14 Client Load Balancing / 233
- 12.1.15 Oracle Enterprise Manager / 233
- 12.2 Oracle 9i New Features / 233
 - 12.2.1 Real Application Clusters (RAC) / 234
 - 12.2.2 Data Guard / 236
 - 12.2.3 Performance Tuning Intelligent Advisors / 239
 - 12.2.4 Actual Operation-Level Query Statistics / 239
 - 12.2.5 Dynamic Sampling of Optimizer Statistics / 239
 - 12.2.6 Cloning Production Database with Oracle Enterprise Manager / 240
 - 12.2.7 Renaming Columns and Constraints / 241
 - 12.2.8 Dynamic Memory Pools / 241
 - 12.2.9 Flashback Query / 241
 - 12.2.10 List Partitioning / 241
- 12.3 Oracle 10g New Features / 241
 - 12.3.1 Automatic Storage Management (ASM) / 242
 - 12.3.2 Asynchronous Commit / 244
 - 12.3.3 Database Replay / 244
 - 12.3.4 Read Performance Statistics Directly from the SGA / 245
 - 12.3.5 Automatic Workload Repository (AWR) / 245
 - 12.3.6 Automatic Database Diagnostic Monitor (ADDM) / 245
 - 12.3.7 Automatic Shared Memory Tuning / 245
 - 12.3.8 Automatic Optimizer Statistics Gathering / 245
 - 12.3.9 SQL Tuning Features / 247
 - 12.3.10 Grid Computing / 247
- 12.4 Oracle 11g New Features / 248
 - 12.4.1 Automatic Memory Management / 249
 - 12.4.2 Intelligent Cursor Sharing / 249
 - 12.4.3 Database Resident Connection Pool (DRCP) / 249
 - 12.4.4 Server Result Cache / 250
 - 12.4.5 Database Smart Flash Cache / 251
 - 12.4.6 Database Replay SQL Performance Analyzer (SPA) Integration / 252
 - 12.4.7 I/O Calibration / 252
 - 12.4.8 Partitioning Enhancements / 252

12.4.9	SQL Plan Management / 253	
12.4.10	Zero-Size Unusable Indexes and Index Partitions / 254	
12.4.11	Invisible Indexes / 254	
12.4.12	Virtual Columns / 254	
12.5	Summary / 255	
	Recommended Reading / 255	
	Exercises / 255	
13	Top 10 Oracle Performance and Scalability Features	257
13.1	Real Application Clustering (RAC) / 258	
13.2	Dedicated versus Shared Server Models / 260	
13.3	Proven Transaction and Concurrency Models / 260	
13.4	A Highly Efficient SQL Optimization Engine / 261	
13.5	Efficient Parallel Processing with Modern Multi-Core CPUs / 261	
13.6	Partitioning / 262	
13.7	An All-Encompassing, Powerful Performance, and Scalability Troubleshooting Tool—AWR / 262	
13.8	The Most Comprehensive Set of Internal Performance Metrics / 263	
13.9	Database Resident Connection Pool / 263	
13.10	In-Memory Database Cache (IMDB) / 263	
13.11	Summary / 263	
	Recommended Reading / 264	
	Exercises / 264	
14	Oracle-Based Application Performance and Scalability by Design	266
14.1	Rapid Development Methodologies / 268	
14.2	Planning / 269	
14.2.1	Vision / 269	
14.2.2	Objectives / 270	
14.2.3	ROI Analysis / 270	
14.2.4	Feasibility Study / 271	
14.2.5	Project Team Formation / 271	
14.3	Requirements Gathering / 272	
14.3.1	Use Cases / 273	
14.3.2	User Views / 274	
14.3.3	Business Processes, Entities, and Business Rules / 274	

14.4	Conceptual Design via Data Modeling / 275
14.4.1	Entity-Relationship Diagramming / 276
14.4.2	The Information Engineering (IE) Format for ERDs / 278
14.4.3	UML Format for ERDs / 279
14.4.4	Relational Format for ERDs / 279
14.5	Logical Design via Normalization / 280
14.5.1	Operational Anomalies / 281
14.5.2	Review of Relation Theory / 282
14.5.3	Functional Dependencies and Lossless-Join Decompositions / 285
14.5.4	First Normal Form (1NF): Avoiding Multi-Valued Columns / 287
14.5.5	Second Normal Form (2NF): Eliminating Partial Dependencies / 288
14.5.6	Third Normal Form (3NF): Eliminating Transitive Dependencies: / 288
14.5.7	Boyce-Codd Normal Form (BCNF): Eliminating Key—Non-Key Dependencies / 289
14.5.8	Fourth Normal Form (4NF): Trivializing or Keying Multi-Valued Dependencies / 290
14.5.9	Fifth Normal Form (5NF): Trivializing or Keying Join Dependencies / 292
14.5.10	Which Level of Normalization to Settle Down? / 294
14.5.11	Denormalization? / 294
14.6	Physical Design / 295
14.6.1	Naming Conventions / 297
14.6.2	Creating Tablespaces / 298
14.6.3	Creating a Schema User with Proper Privileges / 299
14.6.4	Creating Application Schema Objects / 299
14.6.5	Changing Schema Objects / 308
14.6.6	Enforcing Business Rules and Data Integrity / 309
14.6.7	Adding Views / 312
14.6.8	Creating Sequences and Synonyms / 312
14.6.9	Adding Indexes / 313
14.6.10	Security / 314
14.7	Implementation / 315
14.7.1	Choosing an Effective and Efficient Coding Path / 315
14.7.2	Leveraging Proven Oracle Database Design Principles / 316

14.7.3	Leveraging Proven Application Design Patterns /	318
14.7.4	Enforcing with an Effective and Efficient Testing Process /	319
14.8	Release To Market (RTM) /	322
14.9	Continuous Improvements /	322
14.10	Summary /	323
	Recommended Reading /	324
	Exercises /	325

15	Project: Soba—A Secure Online Banking Application on Oracle	326

15.1	Getting SOBA Up and Running /	328
15.1.1	Prerequisite Software /	328
15.1.2	Initial Software Stack Setup /	329
15.1.3	Creating SOBA Database on Oracle /	330
15.1.4	Installing SOBA on Eclipse IDE /	330
15.1.5	Configuring SOBA to Work with Oracle /	331
15.1.6	Configuring SOBA to Work with Hibernate /	333
15.1.7	Building SOBA and Deploying SOBA with Ant to Run on Tomcat /	333
15.2	Overview of Spring Framework /	333
15.2.1	Background /	333
15.2.2	Spring for Building Flexible Applications Faster /	334
15.2.3	Spring Inversion of Control (IoC) and Dependency Injection /	335
15.2.4	Features of Spring 3.0 /	336
15.3	MVC Architecture /	337
15.3.1	MVC Architecture in General /	338
15.3.2	Spring MVC in Action with SOBA /	340
15.4	Spring MVC Framework Applied to SOBA /	342
15.4.1	Spring DispatcherServlet and WebApplicationContext /	343
15.4.2	Logic Flow of SOBA Defined in Spring MVC Framework /	347
15.4.3	A Web Entry Point Defined in a Spring MVC Web Form /	348
15.4.4	Handler Mapping /	350
15.4.5	Implementing Spring Controllers /	353
15.4.6	A Typical View Defined in a Spring MVC Web Form /	358

15.4.7	A Typical Form Success Controller and its Resultant View / 362	
15.4.8	POJOs Referenced in the CreateCustomerFormController / 364	
15.5	Hibernate Object-Relational Mapping (ORM) Applied to SOBA / 368	
15.5.1	Benefits of Using Hibernate / 369	
15.5.2	Metadata Mapping with Hibernate / 370	
15.5.3	Configuring Hibernate to Work with Oracle / 371	
15.5.4	Hibernate DAO / 373	
15.6	RESTful Web Services Applied to SOBA / 376	
15.6.1	Introduction to RESTful Web Services / 376	
15.6.2	RESTful Constraints / 377	
15.6.3	RESTful Interface Design Principles / 378	
15.6.4	Spring's Support for RESTful Web Services / 379	
15.6.5	Server Code / 380	
15.6.6	Client Code / 383	
15.7	Spring Security Applied to SOBA / 386	
15.7.1	Basic Concepts / 387	
15.7.2	Security Configured in web.xml / 387	
15.7.3	Security Configured in soba-security.xml / 388	
15.7.4	Implementing Spring Security in Views / 394	
15.8	Spring ACL Applied to SOBA / 394	
15.8.1	Creating ACL Tables in Oracle / 395	
15.8.2	Configuring Spring ACL / 395	
15.8.3	Maintaining ACLs for SOBA Domain Objects / 398	
15.8.4	Applying ACLs to Business Operations / 404	
15.8.5	Testing ACLs with SOBA / 406	
15.9	Summary / 413	
	Recommended Reading / 414	
	Exercises / 414	

PART 3 OPTIMIZING ORACLE PERFORMANCE AND SCALABILITY 415

16	Logistics of the Oracle Cost-Based Optimizer (CBO)	417
16.1	Life of a SQL Statement in Oracle / 418	
16.2	Oracle SQL Optimizer: Rule-Based versus Cost-Based / 420	

16.3	CBO Statistics / 421
16.4	Pivot Role of Gathering Database Statistics to CBO / 422
16.5	Methods of Gathering CBO Statistics / 424
16.6	Locking and Unlocking CBO Statistics / 425
16.7	Explain Plan—A Handle to CBO / 425
16.8	Data Access Methods—CBO's Footprints / 426
16.9	Looking Up CBO's Plan Hidden in V\$SQL_PLAN / 427
16.10	When CBO may Generate Suboptimum Execution Plans / 428
16.11	Summary / 429
	Recommended Reading / 429
	Exercises / 430

17 Oracle SQL Tuning 431

17.1	Tuning Joins / 432
17.2	Tuning Subqueries / 437
17.3	Case Study: Performance of SUBQUERY versus JOIN / 439
17.4	Case Study: Performance of IN versus EXISTS / 443
17.5	Case Study: A SQL Tuning Yielded a 12x Performance Gain / 444
17.6	Summary / 447
	Recommended Reading / 447
	Exercises / 448

18 Oracle Indexing 449

18.1	Rules of Thumb on Indexing / 450
18.2	Creating and Using Ubiquitous b-Tree Indexes / 451
18.3	Advanced Indexing Scheme I: Covering Indexes versus Index-Organized Tables / 452
18.4	Advanced Indexing Scheme II: Function-Based Indexes (FBIs) / 453
18.5	Unusual Indexing Scheme I: BITMAP Indexes / 454
18.6	Unusual Indexing Scheme II: Reverse Key Indexes / 455
18.7	Unusual Indexing Scheme III: Compressed Composite Indexes / 455
18.8	How To Create Oracle Indexes / 456
18.9	Summary / 457
	Recommended Reading / 458
	Exercises / 458

19 Auto_Tune Features 459

- 19.1 Oracle Automatic Database Diagnostic Monitor (ADDM) / 460
- 19.2 Automatic Undo Management / 462
- 19.3 Data Recovery Advisor / 462
- 19.4 Memory Advisors / 462
- 19.5 MTTR Advisor / 466
- 19.6 Segment Advisor / 466
- 19.7 SQL Advisors / 467
- 19.8 SQL Performance Analyzer / 469
- 19.9 Summary / 470

Recommended Reading / 471

Exercises / 471

PART 4 CASE STUDIES: ORACLE MEETING REAL WORLD PERFORMANCE AND SCALABILITY CHALLENGES 473**20 Case Study: Achieving High Throughput with Array Processing 477**

- 20.1 Context / 478
- 20.2 Performance Model / 479
- 20.3 Tests / 480
- 20.4 Solution / 480
- 20.5 Effects of Array Processing / 482
- 20.6 Summary / 484

Recommended Reading / 484

Exercises / 484

21 Case Study: Performance Comparison of Heap-Organized versus Index-Organized Tables 485

- 21.1 Context / 486
- 21.2 Conversion from Heap-Organized to Index-Organized / 487
- 21.3 Creating Indexes / 487
- 21.4 Creating Constraints / 488
- 21.5 EXPLAIN PLANs / 488
- 21.6 Oracle SQL Traces / 489
- 21.7 Summary / 490

Recommended Reading / 491

Exercises / 491

22	Case Study: SQL Tuning: “IN” versus “OR” versus Global Temporary Table	492
22.1	Context / 493	
22.2	Test Program / 494	
22.3	Observation 1: IN_CreateStatement is the Best Performer / 495	
22.4	Observation 2: Batch Insert Saves Time / 497	
22.5	Temptable Performed Better without an Index Hint than with an Index Hint / 498	
22.6	Effects of APPEND Hint for Populating Temptable / 499	
22.7	Effects of Number of Iterations / 499	
22.8	OR and IN without the Index Hint / 499	
22.9	Limitation on the Number of Literal Values and the Size of OR Statement / 501	
22.10	Dealing with More Than 1000 Literal Values for an IN Based SQL Query / 501	
22.11	A Recommendation for Dealing with 1000 Literal Value Limit in an IN Statement / 501	
22.12	Summary / 502	
	Recommended Reading / 503	
	Exercises / 503	
23	Case Study: Data Access Paths (Double Buffering)	504
23.1	Data Access Paths in General / 505	
23.1.1	Data Buffering / 507	
23.1.2	Inode Locking / 509	
23.1.3	Write-Sync Daemon / 510	
23.2	Test Environments / 511	
23.2.1	Solaris on Veritas / 511	
23.2.2	Solaris on UFS / 511	
23.2.3	Windows on NTFS / 512	
23.3	Test Results with Solaris on Veritas / 514	
23.3.1	Test Run #1—145 ms Average Read Time / 514	
23.3.2	Test Run #2—401 ms Average Read Time / 516	
23.3.3	Test Run #3—261 ms Average Read Time / 518	
23.3.4	Test Run #4—0.98 ms Average Read Time / 519	
23.3.5	Analysis / 521	
23.4	Test Results with Solaris on UFS / 522	
23.4.1	Test Run #1—447 ms Average Read Time / 522	
23.4.2	Test Run #2—10ms Average Read Time / 524	
23.4.3	Analysis / 525	

23.5	Test Results with Windows on NTFS / 526	
23.5.1	Test Run—8 ms Average Read Time / 526	
23.5.2	Analysis / 528	
23.6	Moral of the Case Study / 528	
	Recommended Reading / 529	
	Exercises / 530	
24	Case Study: Covering Index	531
24.1	Getting to Know the Application Architecture / 533	
24.2	Quantifying the Problems / 533	
24.3	Analyzing Bottlenecks / 533	
24.4	Applying Optimizations/Tunings / 535	
24.5	Verifying the Fixes / 535	
24.5.1	Report Summary / 537	
24.5.2	Wait Events Statistics / 538	
24.5.3	SQL Statistics / 541	
24.5.4	IO Stats / 544	
24.5.5	Buffer Pool Statistics / 544	
24.5.6	Wait Statistics / 544	
24.5.7	init.ora Parameters / 545	
24.6	Moral of the Case Study / 545	
	Recommended Reading / 546	
	Exercises / 546	
25	Case Study: CURSOR_SHARING	547
25.1	The Concept of a Bind Variable / 548	
25.2	Oracle CURSOR_SHARING Parameter / 549	
25.3	Getting to Know the Application Architecture / 550	
25.4	Quantifying Problems / 550	
25.5	Analyzing Bottlenecks / 551	
25.5.1	Report Summary / 552	
25.5.2	SQL Statistics / 556	
25.5.3	IO Stats / 557	
25.5.4	Wait Statistics / 558	
25.5.5	init.ora Parameters / 558	
25.6	Applying Tuning: CURSOR_SHARING = FORCE / 560	
25.6.1	Report Summary / 561	
25.6.2	Wait Events Statistics / 563	
25.7	Applying Tuning: CURSOR_SHARING = SIMILAR / 564	
25.7.1	Report Summary / 564	
25.7.2	Wait Events Statistics / 566	

25.8	Moral of the Case Study / 569	
	Recommended Reading / 569	
	Exercises / 570	
26	Case Study: Bulk Transactions	571
26.1	Application Architecture / 572	
26.2	Quantifying Problems / 572	
26.3	Identifying Performance and Scalability Optimization Opportunities / 573	
26.3.1	Report Summary / 573	
26.3.2	Wait Events Statistics / 575	
26.3.3	SQL Statistics / 577	
26.3.4	Wait Statistics / 579	
26.4	Effects of Bulk Transactions on Performance / 581	
26.4.1	Report Summary / 581	
26.4.2	Wait Events Statistics / 583	
26.4.3	SQL Statistics / 585	
26.4.4	Wait Statistics / 587	
26.5	Moral of the Case Study / 592	
	Recommended Reading / 593	
	Exercises / 593	
27	Case Study: Missing Statistics	594
27.1	Decaying Performance due to Missing Statistics / 595	
27.2	First Run with no Statistics / 597	
27.2.1	Report Summary / 598	
27.2.2	Wait Events Statistics / 599	
27.2.3	SQL Statistics / 601	
27.2.4	IO Stats / 602	
27.2.5	Wait Statistics / 602	
27.2.6	init.ora Parameters / 603	
27.3	Second Run with Missing Statistics / 604	
27.3.1	Report Summary / 605	
27.3.2	Wait Events Statistics / 606	
27.3.3	SQL Statistics / 607	
27.3.4	IO Stats / 609	
27.3.5	Wait Statistics / 609	
27.4	Third Run with Updated Statistics / 611	
27.4.1	Report Summary / 611	
27.4.2	Wait Events Statistics / 613	
27.4.3	Operating System Statistics / 614	
27.4.4	SQL Statistics / 614	

27.4.5	Wait Statistics / 616
27.5	Moral of the Case Study / 618
	Recommended Reading / 618
	Exercises / 618

28 Case Study: Misconfigured SAN Storage 620

28.1	Architecture of the Apple's Xserve RAID / 621
28.2	Problem Analysis / 622
28.2.1	Report Summary / 622
28.2.2	Wait Events Statistics / 624
28.2.3	IO Stats / 625
28.2.4	init.ora Parameters / 625
28.3	Reconfiguring the RAID and Verifying / 626
28.3.1	Report Summary / 626
28.3.2	Wait Events Statistics / 628
28.3.3	IO Stats / 629
28.4	Moral of the Case Study / 629
	Recommended Reading / 630
	Exercises / 630

APPENDIX A ORACLE PRODUCT DOCUMENTATIONS 633

A.1	Oracle Database Concepts / 633
A.2	Oracle Database Administrator's Guide / 633
A.3	Oracle Database Reference / 634
A.4	Oracle Database Performance Tuning Guide / 634
A.5	Oracle Database 2 Day + Performance Tuning Guide / 634
A.6	Oracle Database 2 Day DBA / 634
A.7	Oracle Database SQL Language Reference / 634
A.8	Oracle Database Sample Schemas / 635
A.9	Oracle Database PL/SQL Packages and Types Reference / 635
A.10	Oracle Database PL/SQL Language Reference / 635
A.11	Oracle Database JDBC Developer's Guide and References / 635

APPENDIX B USING SQL*PLUS WITH ORACLE 636

B.1	Installation / 636
B.2	SQL*Plus and tnsnames.ora File / 637
B.3	Basics of SQL*Plus / 638

	B.4	Common SQL*Plus Commands /	638
	B.5	Using SQL*Plus to Execute SQL Statements /	639
	B.6	Using SQL*Plus to Execute PL/SQL Blocks /	640
	B.7	Using SQL*Plus Autotrace to Obtain EXECUTION PLANS and Optimizer Statistics /	640
	B.8	Using SQL*Plus Timing Command /	641
	B.9	Exporting/Importing Oracle Databases with SQL*Plus /	642
	B.10	Creating AWR Reports with SQL*Plus /	643
	B.11	Checking Tablespace Usage with SQL*Plus /	644
	B.12	Creating EM DBConsole with SQL*Plus /	646
APPENDIX C	A COMPLETE LIST OF ALL WAIT EVENTS IN ORACLE 11g		648
APPENDIX D	A COMPLETE LIST OF ALL METRICS WITH THE V\$STATNAME VIEW		656
APPENDIX E	A COMPLETE LIST OF ALL STATISTICS WITH THE V\$SYSSTAT VIEW		667
INDEX			681

Preface

God created the integers, all else is the work of man.
—**Leopold Kronecker**

WHY THIS BOOK

This book stemmed from the author's other book—*Software Performance and Scalability: a Quantitative Approach*, published by Wiley in 2009. That book helps readers grasp the basic concepts, principles, methodologies, best practices, queueing theories, and profiling tools associated with optimizing software performance and scalability in general. Many quantitative, real world case studies have been used to support the notions and theories presented therein. The book has been positively received around the world. Some readers suggested I apply the same style and approach adopted in that book, namely, basing all concepts and theories on quantitative, real world case studies, to explore systematically the *art* and *science* of optimizing the performance and scalability of some common foundational software platforms, such as database and virtualization platforms, upon which various software applications are built and run.

After some deliberation on the suggestions described above, I decided to give it a try. It occurred naturally to me that I should do it with Oracle first for a few reasons. One reason is that I have been working in the trenches on Oracle-based enterprise application performance and scalability for more than a decade. I have studied, used, optimized and tuned Oracle a lot; and most importantly, I had the foresight to accumulate many good quantitative case studies based on my own first-hand real

experiences. I felt compelled to share all of my Oracle endeavors with a broader audience.

My second reason for writing this text is to offer an alternative, more effective and more efficient approach to learning Oracle and its performance and scalability features. One can certainly learn from the product documentations accompanying every release of Oracle. Yet, as of Oracle 11g, those documentations total over 10,000 pages! This apparently is not an effective approach to learning Oracle systematically in a reasonable timeframe. Then, what about so many other Oracle texts published over the years? Certainly, there are very excellent texts like Tom Kyte's, which contain comprehensive, reliable information; but there are also many Oracle texts that are full of opinions rather than facts supported with quality, quantitative, real world case studies. It is critical to distinguish between facts and opinions. Facts are supported by measured data, which is repeatable, whereas opinions are personal and not necessarily based on facts in general. Because of my physics research background, I always prefer the technical texts that are based on facts rather than opinions, and this text just falls into that category.

Along the way, I felt more and more obligated to make this text a concise, rigorous, and quantitative textbook so that university/college CS professors and their students could use it to supplement their database courses. I hope it will be useful not only in classrooms but also in the field for those professionals who strive to develop highly-performing, scalable enterprise software products based on Oracle. Incidentally, I am not on commission from Oracle. This has been totally my own choice that I feel is worth my time based on the intrinsic high performance and scalability of Oracle that I like and I know of.

WHO THIS BOOK IS FOR

One of the primary objectives of this text is to provide college professors who teach database courses at advanced undergraduate or post-graduate level with a much-needed, supplementary textbook. I took a database course at a U.S. college more than ten years ago when I was preparing for a career transition from physics research to computers. Retrospectively, I strongly feel that a database course should teach students not only database concepts and theories but also practical implementations in a real product like Oracle. It would be more ideal and beneficial if the concepts and theories could be corroborated with a real product like Oracle that is proven to be intrinsically high performing and scalable. Students could potentially have a much more rewarding future career if they were given a chance to have their classroom exercises enhanced with a solid, real database product.

The other equally weighted objective is to provide enterprise software professionals with a clearly-structured, reliable text that teaches how to build highly performing, scalable enterprise applications based on the world's most robust database product—Oracle. Although Oracle has been proven to be intrinsically high performing and scalable, software practitioners need to learn how to leverage the performance and scalability features engineered into Oracle to achieve the end result

of meeting performance and scalability requirements with their products as demanded by their customers.

This book has the right style and context both for college professors who teach database concepts and for enterprise software professionals who develop Oracle-based enterprise applications. It has been written carefully with the following considerations:

- practicality-based selection of all basic database concepts and architectural features designed specifically with Oracle from performance and scalability perspectives,
- precise step-by-step instructions about how to perform various Oracle specific tasks in the context of optimizing and tuning Oracle performance and scalability as convenient timesavers for all audiences,
- a full-scale secure online banking application (SOBA) built with the latest technologies such as Spring Framework, Hibernate, and RESTful Web services to demonstrate how an Oracle-based application can be developed with performance and scalability taken into account,
- quantitative case studies demonstrating Oracle meeting performance and scalability challenges in the real world.

These considerations are reflected in how this book is organized as discussed next.

HOW THIS BOOK IS ORGANIZED

This book is divided into the following four parts logically, in order to meet the main objectives described previously:

- Part 1, “Getting Started with Oracle,” consists of four chapters demonstrating how to set up a working Oracle environment with some of the major performance and scalability factors taken into account in the first place. A quick tour is provided to help illustrate all major database concepts in Oracle’s context. In summary, this part helps a reader get up to speed quickly with getting around an Oracle server. Based on my own experience, the best way to learn about a software product starts with learning how to install and set it up. This would serve as an effective stepping stone to learning more advanced concepts and features.
- Part 2, “Oracle Architecture from Performance and Scalability Perspectives,” covers all major database concepts and architectural features related to optimizing Oracle performance and scalability. The following subjects are covered:
 - overall Oracle architecture
 - memory management
 - storage structure

- Oracle wait interface (OWI)
- Oracle data consistency and concurrency
- Oracle automatic workload repository (AWR)
- Oracle advanced features and options
- Top 10 Oracle performance and scalability features
- Oracle-based application performance and scalability by design
- Project: SOBA—A Secure Online Banking Application on Oracle
- Part 3, “Optimizing Oracle Performance and Scalability,” teaches all about how to optimize and tune Oracle performance and scalability. The following subjects are selected:
 - Oracle cost-based optimizer (CBO)
 - Oracle SQL tuning
 - Oracle indexing
 - Oracle auto-tune features
- Part 4, “Case Studies: Oracle Meeting Real World Performance and Scalability Challenges,” provides quantitative case studies out of my own first-hand, real product based experiences to demonstrate how one can achieve high performance and scalability by applying various Oracle performance and scalability best practices. It sets a high standard on teaching Oracle performance and scalability by using quantitative, real world case studies rather than oversimplified, classroom-setting oriented, *Scottish* examples. Students and enterprise software professionals will be equipped with ready-to-apply techniques that can easily result in multifold or even orders-of-magnitude improvements on the performance and scalability of real products.

In addition to the main text, a few appendices are provided at the end of the book as handy references for performing various routine tasks in dealing with Oracle performance and scalability challenges.

SOFTWARE AND HARDWARE

To help make the most of this text, a hands-on approach is recommended. One can have an Oracle setting with the latest version of Oracle (11g release 2 as of this writing) installed on the following hardware systems (note that Oracle runs on a wide range of hardware and OS platforms).

- For college students, a typical Oracle setting might just be a laptop with the latest version of Oracle installed. For writing this text, I used two PCs:
 - Oracle Server PC with the following specs (HP Pavilion desktop p6620f):
 - OS: Windows 7 Home Premium 64-bit

- Processor: AMD Phenom II X4 quad-core 830 @ 2.8 GHz (2 MB L2 + 4 MB L3 Cache, 4 GHz System Bus)
- Memory: 6 GB DDR3 SDRAM (3 × 2 GB)
- Disk: SATA 1 TB (7200 RPM, 64 MB Cache)
- Network: 10/100 Ethernet LAN; Wireless LAN 802.11b/g/n
- Dedicated Oracle Client PC with the following specs (Toshiba Satellite laptop L655 – S5103):
 - OS: Windows 7 Home Premium 64-bit
 - Processor: Intel Core i3-370M 2 Cores/4 Hyper Threads @ 2.4 GHz (512 KB L2 + 3 MB L3 Cache, 2500 MHz Front Side Bus)
 - Memory: 4 GB DDR3 SDRAM
 - Disk: HDD 500 GB (5400 RPM, 8 MB Cache)
 - Network: 10/100 Ethernet LAN; Wireless LAN 802.11 b/g/n

Note that computer performance is not all about CPUs. A larger amount of memory and a faster disk are as equally important as fast CPUs.

- For enterprise software professionals, an Oracle setting could range from a laptop or a desktop comparable to the PCs as described above at a minimum or readily to an enterprise-class server system with much larger computing capacities.

After you have decided on a computer on which you are going to experiment with Oracle, next, I am going to suggest how you can use this book most effectively and efficiently to achieve the maximum benefits along the way.

HOW TO USE THIS BOOK

Depending on the background and interests of a reader, this text can be used in a variety of ways. Instead of suggesting what path a reader should take, I'd like to recommend a few learning elements from which a reader can build a learning path to suit his/her own interests. These learning elements include:

- *Setting up an Oracle Server Environment.* It is strongly recommended to have at least a one-time hands-on experience with getting an Oracle Server up and running by following the procedure given in this text. This kind of experience is more than just installing and creating an Oracle database. You will get a real feel for what components an Oracle Server has, and thus understand the architecture of a robust database server product better. Besides, you will get an early exposure to most of the major Oracle performance and scalability settings as well as initialization parameters that can help ease your subsequent consumption of the material presented in this text significantly. This is also the proven, most effective and efficient approach to learning the architecture of a software

product. By going through the process of setting up and configuring a product, combined with examining the relevant configuration files and navigating around on the product's admin console and user console if applicable, within a matter of hours, one could get a very solid feel about how things work together with the product. Note that learning by reading is merely a visual, brain exercise. Only actually getting your hands dirty can give you a real feel.

- *Understanding the Concepts First before Getting Your Hands Dirty.* An intellectual learning process always starts with grasping a basic set of concepts. It is always important to understand the concepts first, since they are the basic elements for associative and creative thinking. I have tried my best to explain various database concepts in Oracle's text as clearly as possible while meeting the goal of limiting the entire text to about 650 pages (in my opinion, wasting a reader's time with too verbose a text is kind of a *soft sin*). However, you may still feel that certain concepts might have not been explained in detail to the level that you can understand. If this turns out to be the case for you, please email me your comments and I'll try my best to address your questions. Or, you can refer to some other sources, for example:
 - Oracle's own documentation accompanying each release, which is not only freely available online but also as authentic as they can be. For the latest version of Oracle 11g R2 as of this writing, all documentation is accessible at <http://www.oracle.com/pls/db112/homepage>. I strongly recommend three texts here: (1) the *Concepts* document, which explains various concepts in depth; (2) the *Administrator's Guide*, which starts out by explaining clearly the Oracle architecture first before describing how to carry out various administrative tasks, and (3) the *Performance Tuning Guide*, which contains all Oracle performance tuning tips.
 - Tom Kyte's text, *Expert Oracle Database Architecture: 9i, 10g and 11g Programming Techniques and Solutions*, 2nd Edition, APress, New York, 2010. Tom knows Oracle inside out, and his data-based, scientific approach to exploring every Oracle issue has made him an asset to the Oracle community.
- *Experimenting with the Secure Online Banking Application (SOBA).* This is an Oracle-based sample application intended for demonstrating all major Oracle performance and scalability features. I developed it with an end-to-end, piece-by-piece approach. This sample application will not only help demonstrate the full development life cycle of an Oracle-based application but also serve as a valuable educational and experimental tool for exploring Oracle performance and scalability features. I have decided to take this sample application—SOBA—much further than just another Oracle sample schema or a standalone backend tier: the application tier and Web tier were coded with one of the most widely used Java development platforms, *Spring Source* (version 3.0), in conjunction with some of the standard Web technologies available today. It is a very exciting and highly challenging project, and I will even bring you into some of the very hot new software technologies such as RESTful Web services. This is a fully fledged

project, and I hope it will open a new path to learning about Oracle performance and scalability from a developer's perspective.

- *Quantitative Case Studies.* About one third of this text is dedicated to quantitative case studies out of my own first-hand, real product based experiences since Oracle 8i. This is one of the aspects that this text possesses, which differentiates itself from many other Oracle texts, outdated or new, on the market. Because of my background as a physicist, whenever I pursue a software performance and scalability issue, I always strive to set everything down to a firm ground as if it were a scientific experiment. I have a strong belief in *data*, and I hope those quantitative case studies can help open up horizons for you to learn how Oracle meets real world performance and scalability challenges.

Finally, to be fair and responsible, I have to make it clear that this is not a *how-to* text for full-time Oracle DBAs, although the Oracle performance and scalability troubleshooting methodologies and those quantitative case studies based on real products and customer engagements might be useful for such a group of Oracle professionals. For the same reason, I only cover those administrative tasks that are necessary for carrying out Oracle performance and scalability work in a broad sense.

HOW TO REACH THE AUTHOR

All errors in the text are the author's responsibility. You are more than welcome to email your questions and comments to me at henry_h_liu@perfmath.com. Your valuable feedback will be taken seriously and acknowledged in the next version of this text. For downloads and updates, please visit the book's Web site at <http://www.perfmath.com>.

HENRY H. LIU, PH.D.

*Folsom, California
Spring, 2011*

Acknowledgments

First, I would like to thank both my current employer, BMC Software, Inc., and my previous employers of Amdocs and Intel for all Oracle related opportunities through my employment during the past ten years, as without such great places to work, I could have not accumulated so much first-hand experience in Oracle and written this book. I would also like to thank Dr. Larry Bernstein for his great vision of *quantitative computing* with Wiley as well as his continuous support throughout the entire process of getting this text published. I am very grateful to Simone Taylor, the director of the editorial development at Wiley, for her making this publishing experience as smooth as possible. Many thanks to those anonymous reviewers for their professional opinions and generous comments, which greatly influenced my decisions on how this book should be organized and written to better serve the potential audiences of college students and enterprise developers. At last, my gratitude extends to both my wife Sarah and our son William for their invaluable support and patience. Hopefully I won't do it more than once again.

I have been deeply impressed by the splendid endeavor of NASA's launching of the rover Spirit to Mars. The cover design of this book is a best annotation of what performance and scalability really mean—how fast and how far one can go! Thanks for NASA's permission for using the Spirit illustration on the cover design to express the theme of this book in an indirect context.

Introduction

*Give a man a fish and you feed him for a day. Teach a man to fish and you feed him
for a lifetime.*
—A Chinese Proverb

As one of the top few largest software companies in the world, Oracle Corporation offers many products in the categories of enterprise servers and applications. Although Oracle has expanded enormously during the past decade with aggressive acquisitions, Oracle database platform remains its strongest flagship product out of its entire portfolios. Rigorously speaking, we should always be specific about which Oracle server we are referring exactly, for example, Oracle database server, Oracle application server, and so on. For convenience, however, in this text, we use the term Oracle server (or simply Oracle) to mean Oracle database server implicitly, without always carrying the term database or database server explicitly when it's contextually obvious.

The Oracle database management system (DBMS) is one of the earliest, most widely used, and most robust database management systems for managing enterprise data. It has numerous features both in terms of functionality and in terms of performance and scalability, from most basic to very advanced ones. The magnitude and complexity of all those Oracle features can be evidenced with over 10,000 page documents accompanying every release of the product. Not every customer uses every feature, though. In reality, most customers use only a subset of all those features.

Next, let's explore some of the main features Oracle has to offer.

FEATURES OF ORACLE

Oracle can work standalone or form a cluster of Oracle servers. The clustered fashion is commonly known as Oracle Real Application Clusters (RACs). With an Oracle RAC, a single database is run across a cluster of servers, with the benefits of scalable performance in addition to fault tolerance. Since an Oracle RAC shares all fundamental performance and scalability peculiarities inherent with a single instance of an Oracle server, we will focus on Oracle as a standalone server only throughout this book. Besides, from an application's perspective, the concept of an Oracle RAC is application-transparent, meaning that the same application can be deployed either on a standalone Oracle server or on an Oracle RAC with no application changes necessary.

Oracle has divided its features based on different editions of the same release. In addition to functionality features, different editions have different hardware specs to match expected performance and scalability. For example, with Oracle 11g, there are three editions: Standard Edition One, Standard Edition, and Enterprise Edition. The Standard Edition One and Standard Edition can take up to 2 and 4 processor sockets at most, respectively, while the Enterprise Edition has no limit to the number of sockets (Note: a socket is a complete package of a processor, which may include multiple cores or logic threads. Both cores and logic processor threads might be called CPUs, but performance-wise, one should only count physical cores as CPUs.) In terms of the amount of physical memory on an Oracle database server, the limit is the maximum the underlying OS can support except the 32-bit versions that are subject to the 4-GB limit, which is extendable to some degree. The database size essentially has no limit as long as the hosting server hardware can support. The OS platforms cover Windows, Linux and UNIX. All three editions support 64-bit mode.

In addition to the common features such as caching at the server and client levels, backup, recovery, data protection, auditing, security, and clustering, and so on, Oracle has more advanced features built in as listed below. Note that most of the following advanced features are available with the Enterprise Edition only:

- *In-Memory Database Cache.* This is a technology for storing the entire database or a subset of it directly in the memory space of an application so that the network communication latency between the application and the Oracle server is eliminated completely. In addition, this approach can offload the backend execution tasks significantly so that the overall application performance is improved. The most fundamental piece of this technology is called TimesTen, which essentially is a memory-optimized relational database.
- *Java, PL/SQL Native Compilation.* Stored procedures deployed in the database can be written in Java or PL/SQL, which is Oracle's proprietary SQL. Such stored procedures can be compiled in native mode to eliminate performance penalty associated with non-native compilations.
- *Automatic Memory Management.* With this feature, Oracle DBAs are released from managing Oracle memory manually. Oracle has introduced a flexible,

dynamic, and adaptive memory management scheme to enable this feature. However, Oracle only automatically manages the amount of memory it is assigned to, so sizing an application and coming up with an accurate estimate of how much physical memory Oracle needs to have still lie with the application stakeholders.

- *Automatic Storage Management.* This is a feature in Oracle 10g/11g that allows all Oracle files such as non-structured data files (binaries, external files and text files, etc.) as well as structured data files to be managed automatically.
- *Partitioning.* This feature enables tables and indexes to be split into smaller, more manageable parts to enhance performance, without requiring changes to the applications.
- *Data Mining.* This feature relates to supporting business intelligence (BI) applications by enabling efficient information extraction from large databases.
- *Advanced Queuing.* This feature is similar to a messaging bus that allows message exchanges with applications based on the well-known publish-subscribe messaging protocol.
- *XML DB.* This feature relates to navigating and querying XML data. Oracle has introduced numerous enhancements to improve performance and scalability of various XML DB tasks.
- *Text.* This feature is called Oracle Text, which allows text-based searching, for example, searching based on keywords, context, pattern matching, HTML/XML section and so on.
- *Spatial.* Oracle designed this feature to meet the needs of advanced geographic information system (GIS) applications.

No matter which subset of Oracle features are used by a customer, there exist a common set of performance and scalability challenges that every customer has to deal with. Oracle specific optimizations and tunings have always been a significant part of developing an enterprise software product that uses Oracle as the backend. Depending on the experience levels of an organization's developers, some organizations are able to deal with those challenges better than others.

On the other hand, many colleges offer database courses to educate students about some basic database concepts in classrooms. Oracle has been far more than an academic research topic. It has been helping countless organizations solve real world day-to-day operational problems for several decades. It's very desirable to find a solid, common base for those abstract, theoretical database concepts taught in classrooms, and in my opinion, that base should come from Oracle.

Whether it's for software practitioners or college students, there is a common question that needs to be answered, that is, what subjects should be covered about Oracle performance and scalability? Just a condensed version of as much as 10% of those 10,000 page Oracle product documents with every feature a little bit or another text that is as generic as it can be and that leaves the reader to experiment trial by error? According to my experience in learning a new software product, it seems that it's not

hard to pick up a text, copy a few simple code examples, and make them work. However, it would be much more effective and efficient if a text is available to show not only how it works but also how *well* it works *quantitatively*. This text has been written with those specific criteria in the author's mind. The objectives are summarized in the next section.

OBJECTIVES

This text encourages the reader to think rather than just read and complete the exercises mechanically. It is designed to help the reader achieve the following objectives at a minimum:

- Getting an opportunity to see how various abstract database concepts are implemented in a multi-billion dollar, leading commercial product. This helps close the gap between the theoretical concepts and real world applications in the field of database systems.
- Acquiring the skill set needed in installing or getting an Oracle database up and running as part of the requirements for conducting your performance and scalability tests. Production databases are managed by experienced, full-time DBAs. Databases in development and testing environments, however, are managed by developers or testing engineers themselves. Therefore, being able to install and configure a database is the first necessary skill for developers and performance testing engineers.
- Getting a good understanding of how Oracle works as one of the most typical database backend systems that you are most likely to encounter or you have been using with the product you are developing. Computers execute the instructions they are given to execute. Software professionals, however, will be able to perform better if they are more proficient in their areas, because their ability to solve a problem is strongly predicated on the knowledge and experience they have. This text, written in a concise, self-consistent, coherent and accurate manner, can help accelerate your process of acquiring knowledge and experience in the context of Oracle performance and scalability.
- Being comfortable with taking care of some of the most basic tasks in maintaining an Oracle database for your application development and performance and scalability tests. Some examples include logging into your Oracle database server from a client interface (GUI or command line), checking and making changes to some of the Oracle initialization parameters, checking and adding extra storage for those heavily used tablespaces when needed, and so on.
- Knowing most of the Oracle tuning knobs that are performance and scalability sensitive. Oracle has evolved over time, and it has a lot of tuning knobs built in to enable flexible, application-specific tuning. Knowing what tuning options are available is the first step toward solving Oracle performance and scalability issues associated with your applications.

- Being able to interpret and trouble shoot Oracle performance and scalability bottlenecks related to your applications. This will involve some more advanced concepts and skill sets. To help sharpen your skills within a manageable time frame, I'll focus on a set of highly reusable and effective techniques that can help you resolve most of your Oracle specific performance and scalability issues.

Note that this text is not intended to be a comprehensive coverage of all aspects about Oracle. Instead, it focuses on the performance and scalability aspects of Oracle to help you become efficient in building performance and scalability into your applications that use an Oracle database as the backend. If you are a computer science student, this text provides you with plenty of opportunities for you to see that a commercial quality database product like Oracle has been built with so many concepts you learn in classrooms. The Oracle performance and scalability skill set targeted for software practitioners should be valuable as well in helping precondition a student for a brighter future career.

Next, let's clarify some conventions used throughout the text.

CONVENTIONS

Our first convention is that whenever you see a line starting with `cmd>`, that means an operating system command line prompt without an specific directory path given explicitly. A SQL*Plus command line prompt is indicated with `SQL>`.

If you see anything like `<. . >` in a command or in a code snippet, that means you need to replace it with your own entry to suit your needs. For example, in order to execute the following SQL*Plus command, you need to use your username, password, and connect identifier:

```
cmd>sqlplus <yourUsername>/<yourPassword>@<yourConnectIdentifier>
```

Another convention is that whenever something special in Oracle's context appears in a sentence, it is printed in a distinctive font. For example, Oracle has added new schemas like `HR`, `OE`, and so on, in addition to the original `Scott/tiger` schema. In this case, `HR`, `OE`, and `Scott/tiger` have been typed in a different font from the main text.

In addition, when an important concept needs to be emphasized, it is typed in *italics*. This should be self-evident contextually.

Finally, I would like to mention that we'll make a distinction between an Oracle *server* and an Oracle *Server*. The former with the lower case 's' in the word '*server*' implies the host system on which Oracle is installed, whereas the upper case 'S' in the word '*Server*' implies all the Oracle related run-time components that constitute an Oracle database system on a specific host system. So, an Oracle *Server* runs on an Oracle *server*.

Next, we clarify the subtle differences between *performance* and *scalability*.

PERFORMANCE VERSUS SCALABILITY

Since this book is about Oracle database performance and scalability, it's necessary to clarify what it means by *performance* and *scalability* first. Simply speaking, *performance* is a measurement of the time taken to complete a user action or a computing task. With an online transaction processing (OLTP) application that involves real time interactions between a user and a system, the performance of the application is typically measured in terms of the response time in seconds. With a batch-job application that processes computing tasks in background, a different metric of throughput is used to measure the performance of the application, which measures the number of objects processed within a certain period of time. With a mixed application that has both active users and batch-jobs to service, both metrics of response time and throughput are used to quantify the performance characteristics of the application.

Scalability captures the performance of an application system in a variable scale. From the application perspective, *scaling-up* means supporting more loads of more users and/or more batch jobs, whereas *scaling-down* means supporting reduced loads of fewer users and/or fewer batch jobs. From the hardware perspective, adding more resources (e.g., adding more CPUs) or replacing slower resources with faster resources (e.g., faster CPUs) on the same hardware system is termed *scaling-up* (or *vertical scaling*), whereas adding more identical systems is termed *scaling-out* (or *horizontal scaling*).

For now, it's sufficient to understand conceptually what it means by performance and scalability in general. We'll understand these concepts better later when we discuss various Oracle performance and scalability features with quantitative case studies in the main text. Let's begin with Part I next to help you get a smooth warm-up on Oracle.

Part One

Getting Started With Oracle

Rome was not built in one day.
—John Heywood, 1497–1580, *English Playwright and Poet*

Building and deploying business software systems across an enterprise requires a high-performance, scalable, and reliable database platform. Oracle as a powerful database platform has been fulfilling this role since decades ago. However, Oracle has more and more features built-in as a result of evolutions from generation to generation. In order to be able to cope with Oracle's complexity effectively, we need to learn its peculiarities and strengths in performance and scalability so that we would be able to make the most of it.

Based on my experiences, the best way to learn about a software product is to actually put it up and test-drive it. As soon as you become familiar with it, apply some load and increase load intensity gradually. Then watch how it behaves and even how it breaks. Then find out why it is broken. Figure out how you can make it undergo larger and larger loads by making adjustments to its various tunable parameters. After a few rounds of test drives like this, you could quickly become an expert.

But we need to take baby steps one at a time. In this part, I'll help you achieve the first goal of being able to get Oracle up and running and get around it freely.

This part consists of the following chapters:

Chapter 1, "Basic Concepts," introduces some basic concepts so that we will all be on the same page when we move along to more advanced subjects.

Next, in Chapter 2, I'll show you how to install Oracle software and create an Oracle database without you having to comb through an installation guide and try it

out on your own, repeatedly hitting and missing. It could become very frustrating some times when you think it should be a simple matter but actually it doesn't work no matter how hard you try. In addition, how well an Oracle Server eventually performs and scales is pre-determined by how it is configured at the installation time to some extent. Therefore, consider an Oracle installation experience a valuable opportunity to learn about how to set up an Oracle environment for a high potential of high performance and scalability down the road rather than merely a boring task of getting it up and running.

Chapter 3 gives you a complete overview of all the options you have to access your Oracle database server. Knowing about those options up-front can save you a lot of guessing work and frustrations.

Chapter 4 walks you through all major aspects of an Oracle server with a tour of an Oracle setup I installed on one of my systems.

Let's start with introducing some basic preparatory concepts in Chapter 1 next.

1

Basic Concepts

*Physical concepts are free creations of the human mind, and are not, however it may seem,
uniquely determined by the external world.*

—ALBERT EINSTEIN, *The Evolution of Physics*

Before we begin our exposition of Oracle performance and scalability, we need to consider a few preliminaries. This would include a brief introduction to what SQL is, and then a comparison between relational and object-oriented databases. We'll also clarify the differences between the concept of an Oracle *instance* and that of an Oracle *database* (these two concepts will come up frequently throughout this text). This is a necessary preparation before we take off on optimizing and tuning Oracle performance and scalability.

To be specific, this chapter consists of the following main sections:

- Standard versus Flavored SQLs
- Relational versus Object-Oriented Databases
- An Instance versus a Database

Let's start with a brief overview of standard SQLs versus flavored SQLs next.

1.1 STANDARD VERSUS FLAVORED SQLS

SQL stands for Structured Query Language, which is pronounced as “sequel” or “ess cue ell.” It was said that at Oracle the former has been the norm. SQL was originally invented in 1974 by Donald Chamberlin and Raymond Boyce at IBM. The concept of SQL was further refined by Edgar F. Codd in his influential paper “*A Relational Model of Data for Large Shared Data Banks*,” published in 1970. SQL is a language for querying data, which is structured in relations, with both data and relations stored in a data store or database. SQL was later taken over and standardized by ANSI/ISO. The latest version of standard SQL is SQL: 2008, which is version 6 since the first version of SQL-86 released in 1986.

Most college students as well as computer and software professionals have had some exposure to SQL, and some of them are experts in this area. Our purpose here is not to delve deeply into SQL, but rather to review the components of SQLs divided into a series of subsets as described below:

- *DML (Data Manipulation Language) SQLs.* This subset of SQLs includes the SQL statements of SELECT, INSERT, UPDATE, DELETE, MERGE, and so on. Such SQLs allow users to query and manipulate data stored in a database.
- *DDL (Data Definition Language) SQLs.* This subset of SQLs includes the SQL statements of CREATE, ALTER, DROP, TRUNCATE, and so on. Such SQLs are used to create and modify tables, views, and indexes, and so on.
- *DCL (Data Control Language) SQLs.* This subset of SQLs includes GRANT, REVOKE, and so on. Such SQLs are used for controlling user access to data in a database, for example, granting/revoking certain privileges to/from certain users.
- *TCL (Transaction Control Language).* This subset of SQLs includes COMMIT, ROLLBACK, SET TRANSACTION, and so on. Such SQLs are useful for defining and manipulating database transactions.

Since a SQL standard is a common specification, it's subject to implementations by any interested parties. That leads to various flavors of SQL database products such as IBM's DB2, Microsoft's SQL Server, Oracle's Oracle, and the open source MySQL, PostgreSQL, and so on. MySQL was acquired by Oracle in 2009 as part of the Sun Microsystems acquisition.

Each of those products mentioned above has its own specific procedural language for writing programs and scripts that can run on its database server, for example:

- IBM's SQL is termed SQL PL
- Microsoft's SQL is termed T-SQL
- MySQL's SQL is termed MySQL
- Oracle's SQL is termed: PL/SQL
- PostgreSQL's SQL is termed PL/pgSQL

Mostly, those various flavors of SQLs differ in data types, especially in time and date, as well as in schemas and stored procedures. One needs to learn the proper SQL with a given database product.

Next, let's briefly discuss the subject of relational database management systems (RDBMS) versus object-oriented database management systems (ODBMS).

1.2 RELATIONAL VERSUS OBJECT-ORIENTED DATABASES

Real database-centric enterprise applications are rarely coded in SQL directly or entirely. Instead, they are coded in object-oriented programming languages such as C++, Java, or Microsoft C#, and so on. This has created a disparity between the language used for coding object-oriented application logic and SQL for operating on relational data in a relational database. Thus, the need for storing objects rather than relational tables in a database arose accordingly. It is believed that by supporting data as objects in a database, the overhead of converting between objects and relations can be avoided, resulting in higher development efficiency and better performance as well.

Most major database products, including Oracle, started supporting objects a few years ago. However, it's beyond the scope of this text at this time. We will concentrate on the relational side of Oracle only, mainly because the relational model will remain the mainstream for many years to come. Standard technologies such as Hibernate in the Java camp exist today to take care of object to relational table mapping. Besides, most application development frameworks support issuing SQLs directly to relational databases using technologies such as JDBC (Java database connectivity), which has proven to be effective in alleviating the object-relational gap.

Next, let's clarify the distinctions between an instance and a database in Oracle's context.

1.3 AN INSTANCE VERSUS A DATABASE

Conceptually, an Oracle database and an Oracle instance are two different, complementary entities. An Oracle Server consists of an Oracle instance and an Oracle database. An Oracle database is a logical entity from the data perspective. For example, the constituents of a database include schemas, tables, indexes, views, triggers, stored procedures, dictionaries, as well as users, and so on. Nevertheless, an Oracle instance is more of a physical entity from the system resource perspective with such constituents as processes that perform various tasks, memory areas that hold various types of data, and data files residing on physical disks, etc. An instance can operate on one database only, whereas a database can be operated upon by more than one instance in a clustered environment for high-availability. We will elaborate more on the concepts of database and instance later when we discuss Oracle architecture.

To help you get to know about Oracle quickly, in the next few chapters, I'll walk you through on how to set up and get around an Oracle database server using the latest version of Oracle 11g. Then I'll guide you through a tour of an Oracle Server to help you learn various basic concepts and building blocks of an Oracle Server. There is no better way in learning a software product than actually getting your hands dirty and experimenting with the product with the guidance of the well-written product documents or a more pertinent text such as this book.

1.4 SUMMARY

This chapter briefly introduced some basic concepts such as standard versus flavored SQLs, relational versus object-oriented databases, and an instance versus a database in Oracle's context. The purpose is to help you see the *forests* before seeing the *trees* to which the remainder of this book will be devoted. If you are interested in knowing more about those subjects, refer to the recommended sources listed next.

RECOMMENDED READING

The ISO's Web site (<http://www.iso.org>) has the following SQL-related documents available, which are not free, however:

- SQL - Part 1: Framework (SQL/Framework)
- SQL - Part 2: Foundation (SQL/Foundation)
- SQL - Part 3: Call-Level Interface (SQL/CLI)
- SQL - Part 4: Persistent Stored Modules (SQL/PSM)
- SQL multimedia and application packages - Part 5: Still image
- SQL multimedia and application packages - Part 6: Data mining
- SQL - Part 9: Management of External Data (SQL/MED)
- SQL - Part 10: Object Language Bindings (SQL/OLB)
- SQL - Part 11: Information and Definition Schemas (SQL/Schemata)
- SQL - Part 13: SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
- SQL - Part 14: XML-Related Specifications (SQL/XML)

Since Oracle's object-oriented features are far less widely used than its relational features, there are not many texts about them. If you are interested in learning more about object-oriented features of Oracle, refer to the following text:

W. Rahayu, *Object-Oriented Oracle*, IRM Press, Hershey, 2005.

EXERCISES

- 1.1 If you happen to have had experiences with all or some of those major database products, which product will you recommend for a new enterprise application that is to be settled down on a specific database product? Justify.

- 1.2 In general, which type of database will you be mostly likely to recommend for a new enterprise application: object-oriented or relational? Justify.
- 1.3 What are the criteria for distinguishing between physical and logical entities? Use examples to explain.
- 1.4 What's the major difference between an Oracle *instance* and an Oracle *database* conceptually?

2

Installing Oracle Software

Poets do not go mad; but chess-players do. Mathematicians go mad, and cashiers; but creative artists very seldom.

*—Gilbert Keith Chesterton, **Orthodoxy***

Being able to install and configure an Oracle Server is a required skill for many software developers and test engineers. In many cases, it's unlikely that a dedicated DBA will be assigned to performing all those tasks for you—you have to take care of all the routine tasks of installing, configuring, monitoring, and tuning your Oracle database for yourself. On the other hand, it's a valuable learning experience to install a software product. With such a hands-on exercise, you would know what components and features got installed, what settings went into your setup, and so on. Such first-hand experience and knowledge would be helpful for knowing how to troubleshoot your system later for best possible performance and scalability.

In a typical development environment or performance and scalability test environment, there are two scenarios with setting up a fresh instance of database for your application. One is that you install the database server first, and then launch a `setup.exe` file to run various scripts to deploy your application, including creating the schemas for your application. The other scenario is that setting up a database is an integral part of an entire application installation process, and by the time the application setup is completed successfully, you already have a database set up as well. This second scenario does not require that a database server has been set up *a priori*. In this case, the database is most likely an embedded one that can only exist on the same system as the application server.

Oracle Database Performance and Scalability: A Quantitative Approach, First Edition. Henry H. Liu.
© 2012 John Wiley & Sons, Inc. Published 2012 by John Wiley & Sons, Inc.

Installing an Oracle server could be a smooth or a very frustrating process, depending on how much one knows about the general installation procedure and what the application needs exactly. One needs to understand that the system hardware specs and configuration settings in general have huge impacts on the performance and scalability of the Oracle server which has to meet the performance and scalability requirements of the application. We will expand into these areas in later chapters.

Assuming that you have sized your Oracle server hardware resource requirements properly or you are setting up an Oracle environment just for self-training or exploration, this chapter shows you the following basic tasks involved in setting up a working Oracle database environment:

- Installing the Oracle server software without creating a database
- Creating an Oracle listener for receiving client connections
- Creating an Oracle database
- Installing an Oracle client

Let's begin with elaborating the procedure for installing Oracle server software using the latest version of Oracle 11g R2 next.

2.1 INSTALLING ORACLE 11g SERVER SOFTWARE

Oracle typically comes with two separate installation files for an operating system platform it supports: one for the server software and the other for the client software. Obtain the server installation file either from your organization available internally with a valid license or from Oracle's Web site for evaluation purposes. The version you obtain should match the specs of your system, including the OS type and version as well as the hardware architecture. For instance, Oracle 11g R2 supports the following platforms:

- Microsoft Windows (32-bit & x64 64-bit)
- Linux x86 (32-bit)& x86-64 (64-bit)
- Solaris (SPARC) (64-bit)
- Solaris (x86-64)
- HP-UX Itanium
- HP-UX PA-RISC (64-bit)
- AIX (PPC64).

Assuming that you are installing the Oracle server software on a Windows system, you can start your installation by clicking on the `setup.exe` file. That will start up the Oracle Universal Installer (OUI). The OUI is used for installing and uninstalling all Oracle products. It guides you through the entire installation process with a series of dialogs. What dialogs you will see depends on the version of Oracle and options you

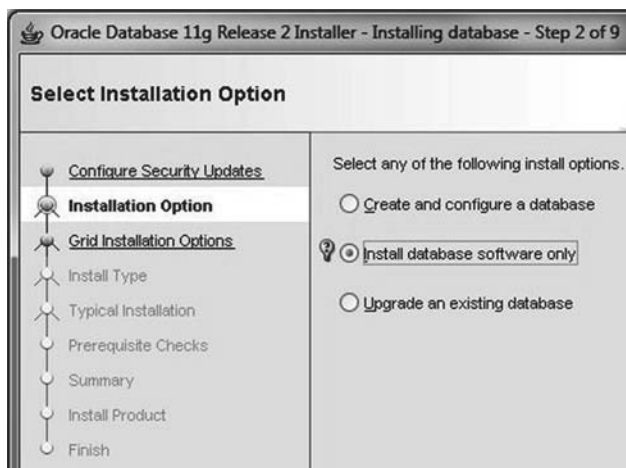


Figure 2.1 Three installation options of Oracle 11g R2.

choose. The below procedure applies to Oracle 11g R2 Enterprise Edition on Windows 7 Home Premium 64-bit. To avoid ambiguities, the installation process is augmented with the proper screenshots.

Since the Oracle 11g R2 server download is divided into two separate zip files, it's necessary to unzip those two zip files into the same directory. Then, to kick off the installation, locate and click the `setup.exe` file in the database directory. After passing the first step of Configure Security Updates, the second step of Installation Option presents three install options: (1) Create and configure a database, (2) Install database software only, and (3) Upgrade an existing database. See Figure 2.1. Select Install database software only and click **Next**.

The third step specifies the option between a single instance database and a Real Application Cluster (RAC). See Figure 2.2. Select Single instance and click **Next**.

The fourth step specifies the language in which your Oracle will run. Select English and continue.

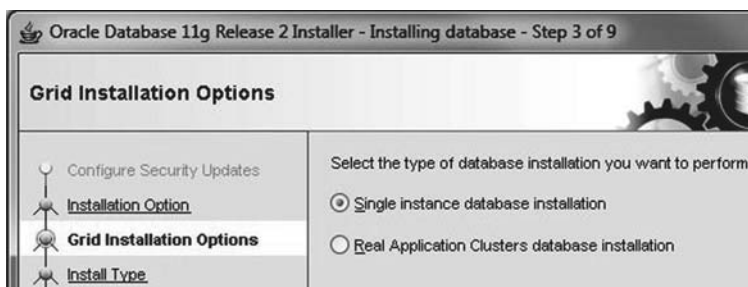


Figure 2.2 Oracle 11g R2 grid installation options: a single instance versus a RAC.

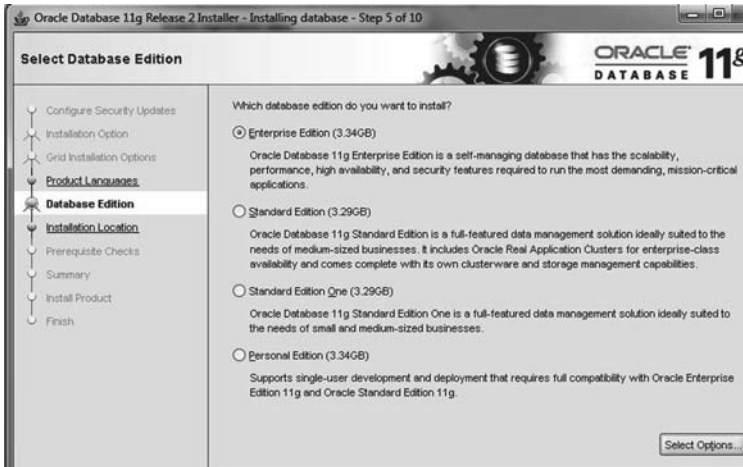


Figure 2.3 Oracle 11g R2 editions: Enterprise Edition, Standard Edition, Standard One Edition, and Personal Edition.

The fifth step specifies the database edition as is shown in Figure 2.3. Note that at the lower right corner, there is a Select Options button, which contains the options as shown in Figure 2.4 by default. For this exercise, select Oracle Partitioning option only with the Enterprise Edition option (note: you might want to review the description for each edition if you are interested in knowing the differences among those editions). Click Next to continue.

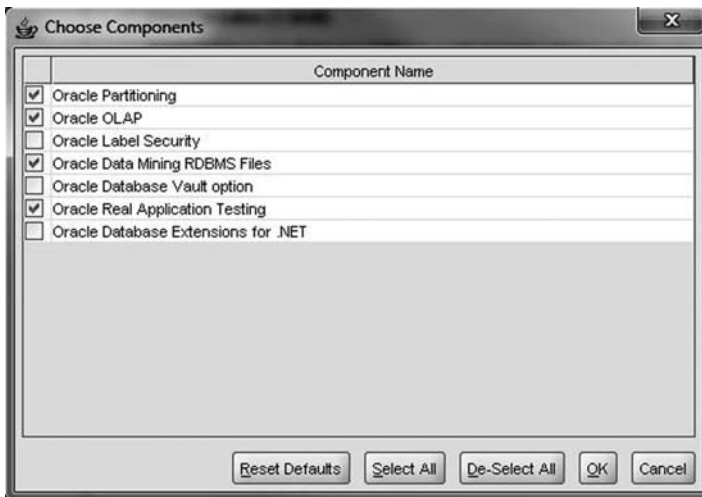


Figure 2.4 Oracle 11g R2 enterprise features (Note: do not uncheck the OLAP box otherwise you will encounter errors when you export/import your database later).

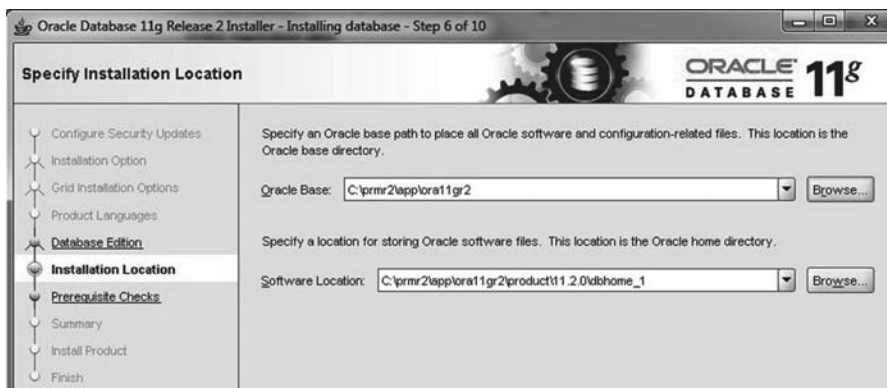


Figure 2.5 Oracle 11g R2 Base and Software Locations. Note that the second entry defines your Oracle home directory.

The sixth step lets you specify the location of Oracle Base and Software Location as shown in Figure 2.5. On Windows, it's more convenient to specify a directory with no spaces in its path name. The subsequent steps just guide you through the rest of the installation process and you should get a message stating your installation is successful at the end of your installation.

Next, you need to configure an Oracle listener to enable clients to communicate with an Oracle server. This step must be performed prior to creating a database, since the information created during this step will be needed when creating a database.

2.2 CONFIGURING A LISTENER

Installing a Listener is easy. After installing Oracle software, go to Start -> All Programs -> Oracle-OraDb11g_home1 -> Configuration and Migration Tools -> Net Configuration Assistant. Select all default settings and keep clicking Next until you are done. Note that the listener is configured to run on TCP port 1521 by default.

The last step is to create a database, as will be demonstrated in the next section.

2.3 CREATING AN ORACLE DATABASE

After installing Oracle server software and configuring a listener as described in the previous sections, now you can create an Oracle database with the Database Configuration Assistant (DCA). To access DCA, go to Start -> All Programs -> Oracle-OraDb11g_home1 -> Configuration and Migration Tools -> Database Configuration Assistant. Click Next on the Welcome dialog, and proceed using the following step-by-step procedure to create your database. A screenshot will not be provided unless it contains information that is interesting and

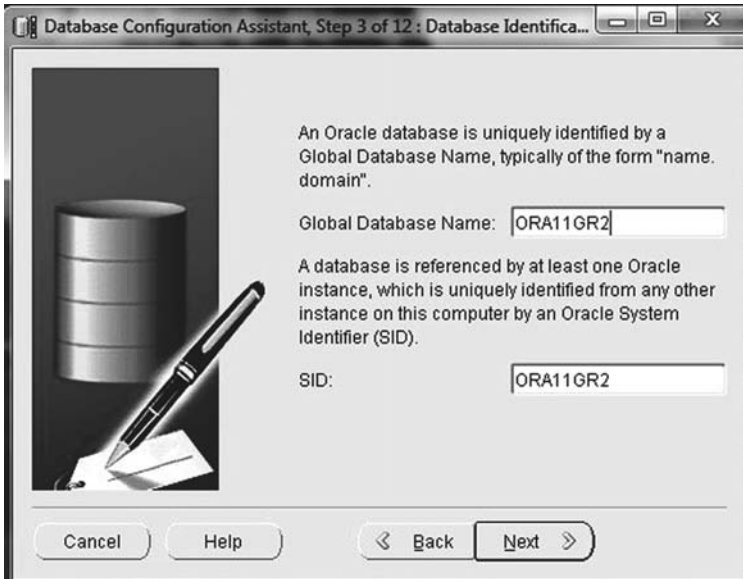


Figure 2.6 Oracle 11g R2: Global Database Name versus SID.

will be referenced later. Also, if this is your first time to install Oracle, you are strongly encouraged to review the descriptions on each installation dialog box which actually explain many database concepts and features well.

1. *Select an Operation.* Select Create a Database and proceed.
2. *Select a Template.* Select General Purpose or Transaction Processing and proceed.
3. *Global Database Name and SID.* Enter Global Database Name and Oracle System Identifier (SID), as shown in Figure 2.6. Although these two entries can have the same value, they represent different concepts of an Oracle database versus an Oracle instance, as we discussed in Chapter 1. You may want to review the description for each entry as shown above each text box. Note that a SID cannot be changed after installation. Click Next.
4. *Enterprise Manager and Database Control.* The information contained in this dialog box, as shown in Figure 2.7, is interesting. First, check the Configure Database Control box to include the HTTP-based EM DB Console in your installation. Note that the Register with Grid Control and Management Service... are grayed out. Leave the other two check boxes unchecked unless you are installing for a production environment. One of these two options enables alert notifications for raising alerts, and the other enables daily disk backup to recovery area. Figure 2.8 also shows the tab of Automatic Maintenance Tasks. Keep these



Figure 2.7 Oracle 11g R2: Management Options to be set at the installation time.

maintenance tasks in mind as they may affect your performance and scalability tests depending on when they are scheduled to run. Click **Next**.

5. **Passwords.** Specify different passwords for different built-in accounts, or use the same administrator password for all built-in accounts. Refer to Figure 2.9. I typically use `system` for all accounts for convenience, but follow your company's policy if you are installing for production use.

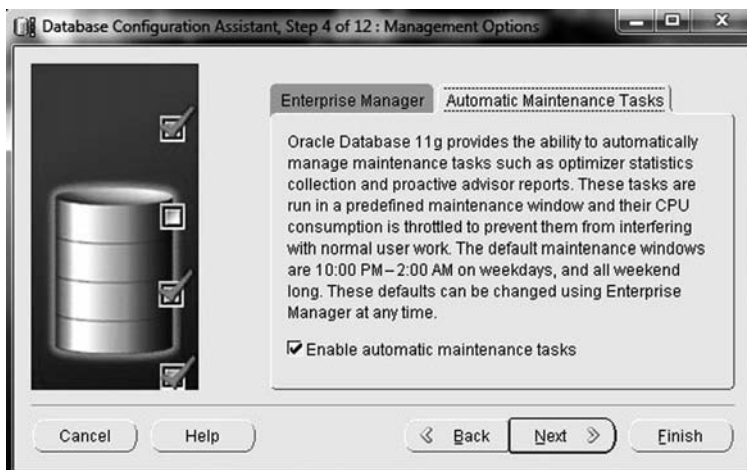


Figure 2.8 Oracle 11g R2: Default automatic management tasks. Note the time windows set by default.



Figure 2.9 Options for setting passwords.

6. *Database File Locations.* As shown in Figure 2.10, this step specifies the locations for Oracle database files. If you have an internal RAID or external SAN to use for your data storage, change to the proper drive accordingly by checking the radio button labeled *Use Common Location for All Database Files*. Otherwise, use the default location on your system if this is only for your own development and are not concerned with I/O performance. Also note that at the lower right corner, there is a button labeled *File Location Variables ...*. See Figure 2.11 for the entries of this dialog box.
7. *Recovery Options.* This step sets the recovery and archiving options as shown in Figure 2.12. Deselect both in a test environment or select both in a production environment.
8. *Sample Schemas.* Check the Sample Schemas box as shown in Figure 2.13 for working on the exercises in the later sections of this text. Otherwise uncheck it if you know that you do not need these sample schemas. We'll discuss more about those sample schemas in a chapter later.
9. *Initialization Parameters.* Now we have come to the fun part of creating an Oracle database. Go over the four tabs of *Memory*, *Sizing*, *Character Sets* and *Connection Mode*, which should look similar to Figure 2.14 parts (a) to (d). We will refer back to these screenshots later. Note that there is a button labeled *All Initialization Parameters* at the bottom, which is where you can view and change the default settings for the initialization parameters that you are particularly interested in. Any changes made here will

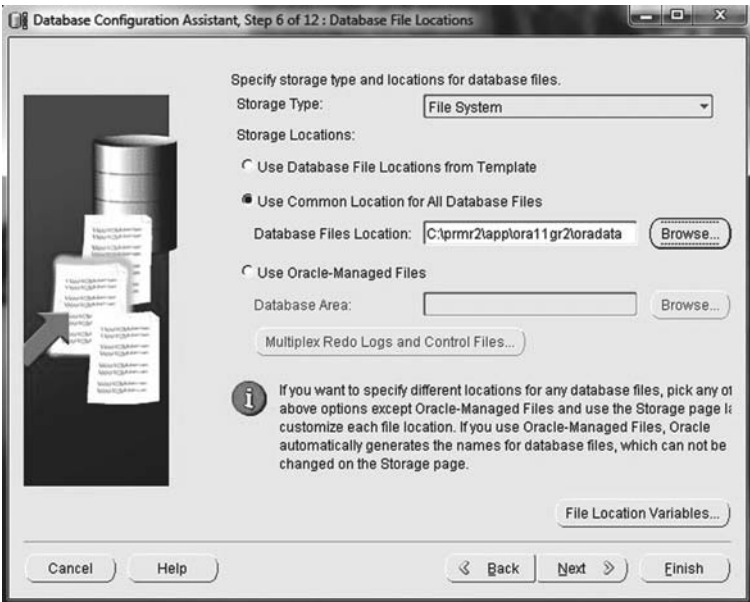


Figure 2.10 Oracle 11g R2: Specifying Database File Locations.

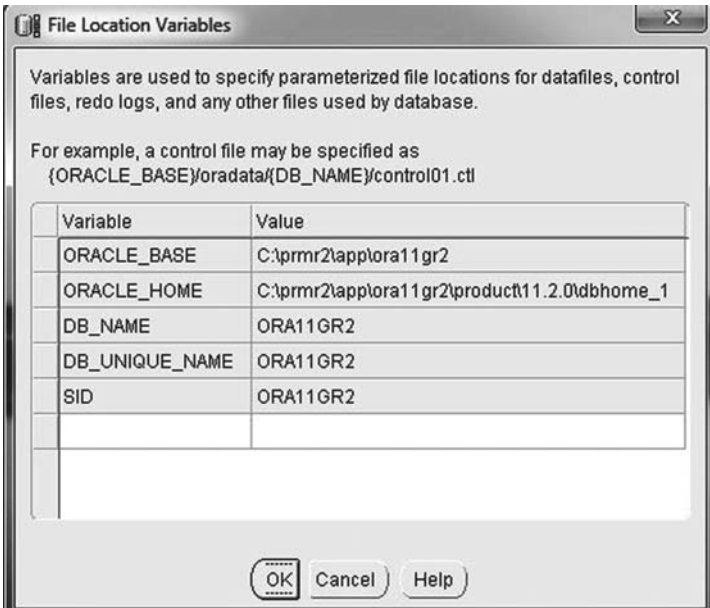


Figure 2.11 Oracle 11g R2: File Location Variables set at the installation time.

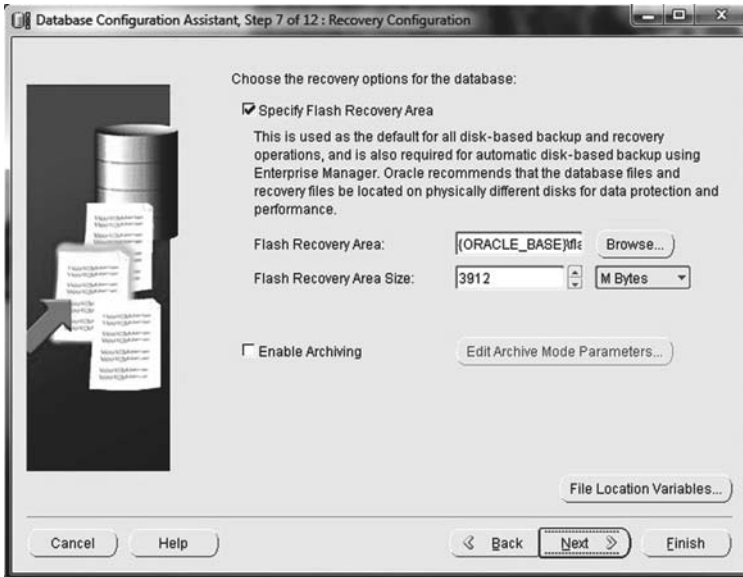


Figure 2.12 Oracle 11g R2: Recovery Options.

be persisted (see Figure 2.14[e]). Also note the Show Advanced Parameters and Show Description buttons at the bottom. The first button brings up more advanced parameters and the second button shows a brief interpretation on a parameter selected.

10. *Database Storage.* Make sure the file locations are what you intend to use and proceed. Also note the Redo Log Groups shown on the left frame as shown in Figure 2.15. Select all default settings and proceed.

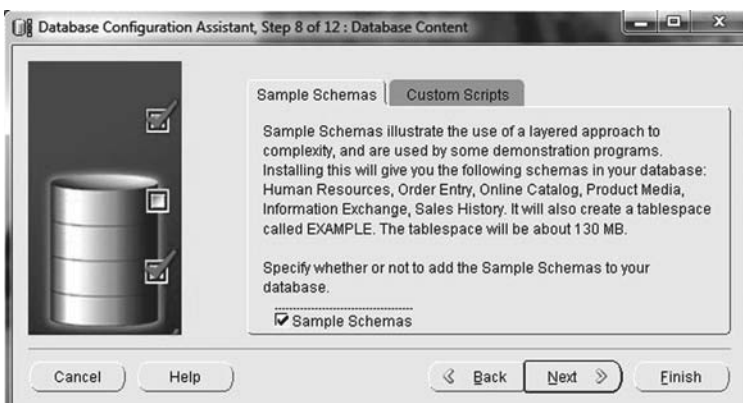


Figure 2.13 Oracle 11g R2: Option for including Sample Schemas at the installation time.

11. *Database Creation Options.* Check the Create Database box and proceed.
12. *Confirmation.* Summary of all create database options. Note that at the lower right corner, there is a button labeled “Save as an HTML file.” Click this button if you want to save a copy of all your install options at this point.

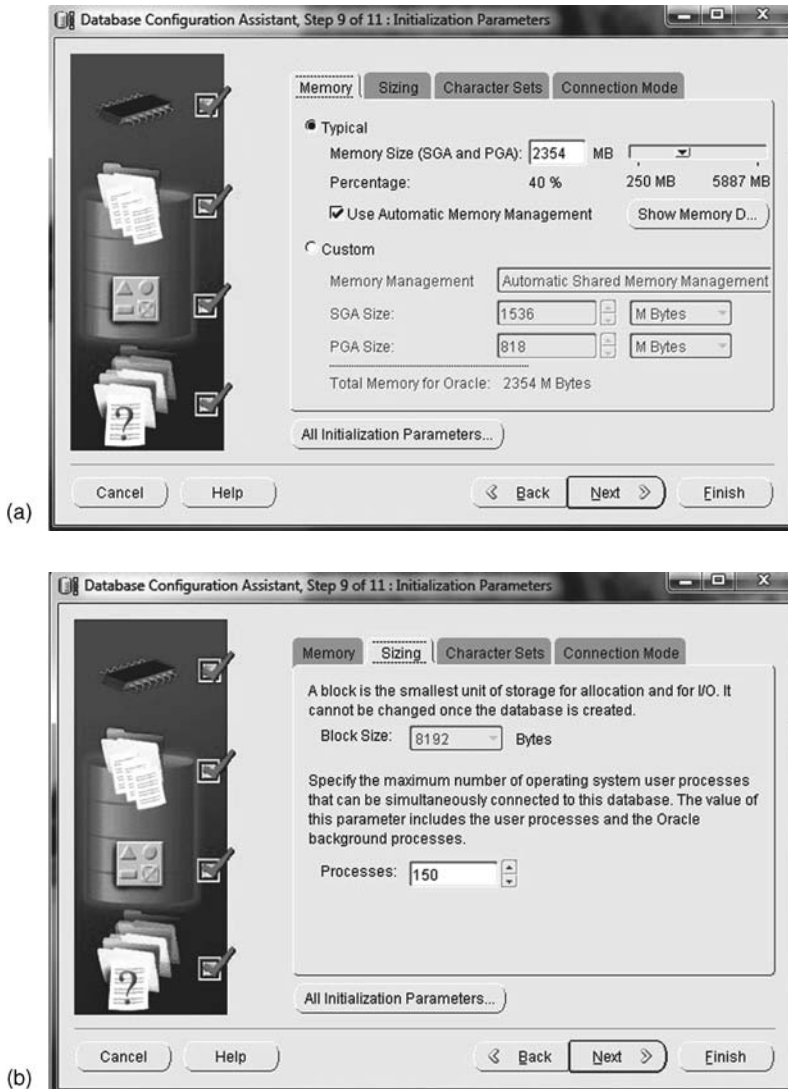


Figure 2.14 Oracle 11g R2: Performance and Scalability Sensitive Parameters: (a) Automatic Memory Management (AMM), (b) Block Size and number of processes for parallelism, (c) Character Sets (note the option of Use Unicode), (d) Connection Mode: Dedicated versus Shared Server Mode, and (e) Some of All Initialization Parameters.

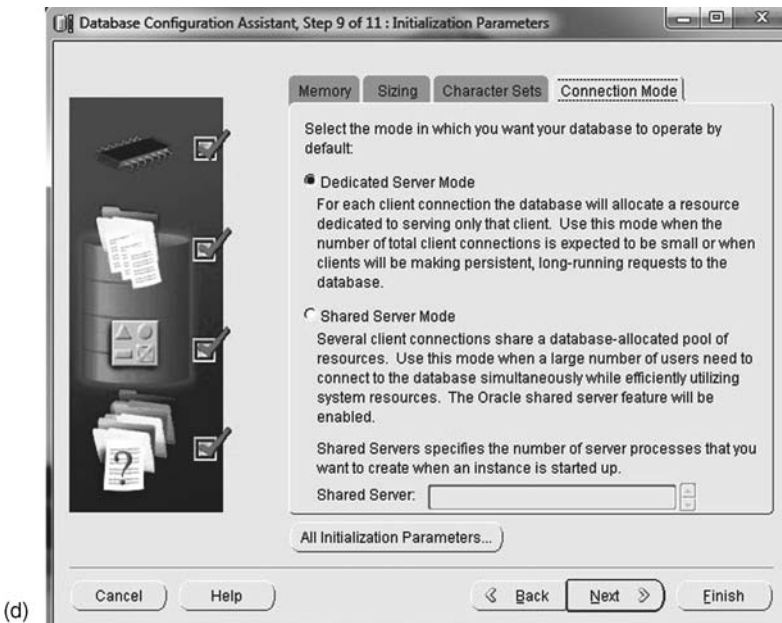
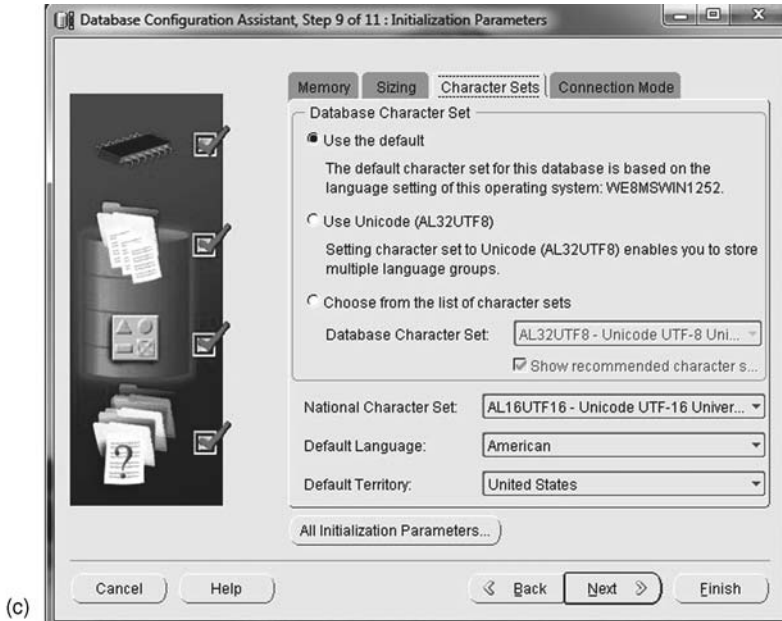


Figure 2.14 (Continued)

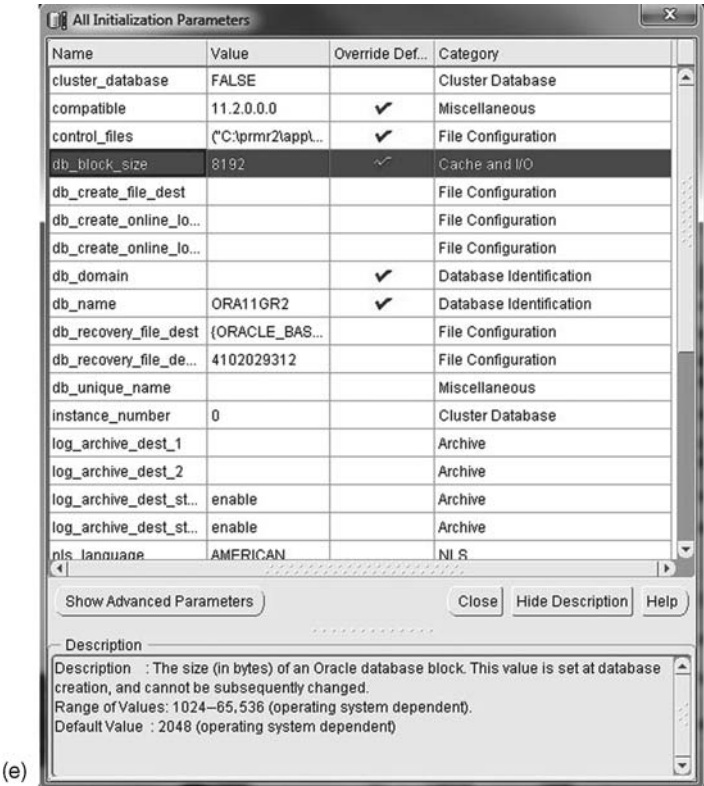


Figure 2.14 (Continued)

- 13. The installation begins. It took 7 minutes on my system.
- 14. A final dialog confirming that database creation is complete. Note a few important items on this dialog: your SID, Server Parameter File Name, and the DB Console URL. Note that the DB console will be accessed with HTTPS instead of HTTP. You can change it to using HTTP, as we will describe later. As suggested, you should also back up the DB console encryption key file of `emkey.ora` in case it gets corrupted over time. Click `Exit` and you are done with creating your Oracle database.

At this point, you should see Oracle services as shown in Figure 2.16 taken from the Windows Services Management snap-in. Now go to the Services Management Console (Start -> All Programs -> Control Panel -> System and Security -> Services) and check out the following services:

- *OracleServiceORA11GR2*. This is the Oracle server process. It must be running in order to make your Oracle database server available to its clients, users and applications. Note that the SID `ORA11GR2` is appended to the name of the



Figure 2.15 Oracle 11g R2: Database Storage (note the redo log groups).

Oracle service on Windows. Also, you can start/stop your Oracle database from here by right-clicking the service entry and bringing up the start/stop dialog.

- *OracleOraDb11g_home1TNSListener*. This is the listener process that must be running in order to enable client connections.
- *OracleDBConsoleORA11GR2*. This is the HTTP-based Oracle Enterprise Manager (EM) DBConsole for accessing and managing the Oracle server. First, make sure that it is started up. Then try it out using your Web browser by accessing it with the URL `https://<your-server-host-name>:1158/em`. Notice that by default the HTTPS protocol rather than the non-secure HTTP protocol is used. With strengthened security on Windows 7, you might get an error of “There is a

Oracle ORA11GR2 VSS Writer Service	Started	Manual
OracleDBConsoleORA11GR2	Started	Automatic
OracleJobSchedulerORA11GR2		Disabled
OracleMTSRecoveryService	Started	Automatic
OracleOraDb11g_home1ClrAgent		Manual
OracleOraDb11g_home1TNSListener	Started	Automatic
OracleRemExecService	Started	Disabled
OracleServiceORA11GR2	Started	Automatic

Figure 2.16 Oracle 11g R2 services after a database is created successfully.

problem with this website's security certificate." If this happens, click "Continue to this website..." Then you should be able to log on with your system account and the password specified during your installation.

It's necessary to keep in mind that the Oracle listener must be running in order for all users and clients to be able to connect to an Oracle server. However, it's not necessary to restart the listener prior to or after restarting the Oracle Server.

Note that those new sample schemas were created with their accounts locked and passwords expired by default. You need to unlock each account and specify a new password as well. A convenient convention is to set the password to be the same as the username, for example, `hr/hr` for the username/password of the HR schema. Also note that the schema names and user names are all stored internally in upper case letters.

In case you want to unlock those sample accounts now, here is the procedure. To unlock an account and assign a new password, execute the following commands on your Oracle server for every user that you want to unlock and assign a new password:

```
MS-DOS>/bin/sqlplus "/as sysdba"
SQL>alter user account unlock identified by ;
SQL>commit;
```

Then you should get a confirmation that "User altered" after executing the first command. To check the account status of a user, execute the following command with your schema user name entered all in upper case letters:

```
SQL>select account_status from dba_users where username =
'<your_user_name>';
```

The above command should return OPEN or EXPIRED & LOCKED, depending on the account status.

This Oracle database server with the SID of ORA11GR2 will be used throughout the remainder of this text whenever applicable to help explain various concepts in the context of Oracle performance and scalability tunings. But before we get there, let's see how to install Oracle client software in the next section.

2.4 INSTALLING ORACLE 11g CLIENT SOFTWARE

The Oracle client software is a package separate from the server package. The client software might be forward compatible within a few releases in general, namely, you can use an older version of Oracle client to access the same or newer versions of Oracle server. But some applications require that the version of the client must match with that of the server. Also, very often, the server might be 64-bit, whereas the client can be 32-bit or 64-bit from the same version of Oracle.

Installing Oracle client software is much simpler than installing the Oracle server software. In this section, we give a brief introduction about the Oracle client so that you would know when you would need to install an Oracle client and what type of client you should install based on your needs.

An Oracle client is installed by first locating and clicking the `setup.exe` file from the Oracle client software package, and then the OUI appears. Ignore the first welcome dialog and specify the destination name and path for the client on the second dialog box. The third dialog box will display the four installation types available, each of which is explained as follows:

- *InstantClient*. This type is for applications that need to load the shared libraries for Oracle Call Interface (OCI) to access an Oracle database, for example, in a typical *n-tier* enterprise architecture that the application tier interacts with the database tier to access data stored in the database. The InstantClient is installed on the same system as the application. As soon as the installation is complete, the application can connect to and access the Oracle database without requiring further configuration.
- *Administrator*. This installation type installs the Oracle Enterprise Manager Standalone Console, Oracle networking services, and client software for clients to connect to an Oracle server. It also installs development tools such as SQL Developer and SQL Plus for developing applications. There are quite a few things to clarify with this installation type:
 - Prior to Oracle 11g, there used to be an Oracle Enterprise Manager Java Console (OEMJC), which served as the admin console for managing Oracle databases. Beginning with Oracle 11g, the OEMJC has been excluded from the Oracle client software. A new HTTP-based admin console called Oracle Enterprise Manager (EM) or simply DBConsole has been introduced since Oracle 10g to replace the OEMJC.
 - As the new EM DBConsole can be accessed using an Internet browser via HTTP, the need for installing an Oracle client has been diminished significantly from the administration perspective. However, some Oracle installations exclude the EM DBConsole, and in this case, you still need to install the client for remotely managing an Oracle database using the command line interface (CLI) SQL*Plus.
 - From the management perspective, you don't need to install an Oracle client if: (1) the EM DBConsole is available on the Oracle server and you are not interested in managing the Oracle database remotely using SQL*Plus; (2) you can log onto the Oracle system directly. For example, you have installed an Oracle database server on your desktop or laptop or a server system that you have full control over and since in this case, you can use the SQL*Plus installed in the server's bin directory.
 - For those who have been using OEMJC, the good news is that you can still use the OEMJC available from the 10g client software to access your 11g server.

- *Runtime.* This type of installation enables applications to connect to an Oracle database on the local or a remote system. It installs the same components as the Administrator option except that the Oracle Enterprise Manager Stand-alone Console will not be installed.
- *Custom.* This type gives the flexibility of choosing features customized to meet certain specific requirements.

After installing the Oracle client, check your PATH environment variable to make sure that the proper Oracle client path is set. In addition, create an ORACLE_HOME environment variable at the global system level by following the access path of Control Panel -> System and Security -> Advanced system settings -> Environment Variables -> System variables/New.... This is especially important when installing an application that requires all necessary Oracle server connection information to be entered. I once had an experience that I set ORACLE_HOME environment variable at the MS-DOS command prompt but not at the global system level as described above. The application installer was launched at the same MS-DOS command prompt and was able to connect to the Oracle server but kept complaining at a later step that the client library does not have proper privilege or the client version is wrong. After hours of troubleshooting, I found out that the application installer only checked at the global system level but not at the MS-DOS command prompt level for the definition of the ORACLE_HOME environment variable. After manually setting the ORACLE_HOME environment variable at the global system level, the application installation went through successfully without complaining about the client library or version.

To enable communicating with an Oracle server, it's necessary to create a `tnsnames.ora` file in the `<ORACLE_HOME>/NETWORK/ADMIN` directory. The `tnsnames.ora` file should contain the TNS descriptor for the Oracle server to be accessed. The next chapter provides more details about this. For now you can just copy the `tnsnames.ora` file from your Oracle server to the client machine. The following excerpt shows a typical entry named with a connect string of `ORA11GR2_CS` in a `tnsnames.ora` file that enables a client to connect to an Oracle server installed on the host `p6620f` with the SID `ORA11GR2`:

```
ORA11GR2_CS =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP) (HOST = p6620f) (PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = ORA11GR2)
  )
)
```

Note that if you are unable to connect to your Oracle server installed separately from your client machine, make sure that the firewall policy is set properly on your Oracle server machine. (In my case I had to disable Windows firewall on the private home

network so that the client and server installed on two separate Windows 7 machines could communicate with each other). You can follow the following procedure to troubleshoot your connection problem:

- Run “ping <host>” command to make sure the Oracle server is reachable from the client machine.
- Run “tnsping <connect_string>” to check your connect setting in the file.
- Run “sqlplus <user>/<password>@<connect_string>” to make sure you can actually connect to your Oracle server with the SQL*Plus tool.

With both Oracle Server and client installed, you might be anxious to do a test-drive. I’ll provide you with some assistance by showing you all common options to access an Oracle server outside an application in the next chapter. How to make an Oracle server available to an application will be discussed in the next chapter as well with two interesting case studies. But before getting there, let’s have a brief discussion on Oracle Grid Control next.

2.5 ORACLE GRID CONTROL VERSUS DB CONTROL

Although the Oracle Grid Control will not be used throughout this book, I am tempted to provide a brief coverage about it for two reasons. First, the letter “g” in the name of Oracle 10g and 11g implies the targeted new computing paradigm shift to *grid computing*. Knowing how Grid Control works gives us a perception about what grid computing is about. Secondly, it’s interesting to know that the Oracle Grid Control manages a large number of systems that constitute a grid, whereas the DB Control that we will mention a lot throughout this book is just a miniaturized Grid Control, which manages only one Oracle instance or an Oracle RAC. Knowing the difference between the Oracle Grid Control and Oracle DB Control is academically interesting.

First, let’s explain what grid computing is about. Grid computing is a kind of large-scale, distributed computing with a large number of nodes or systems that form a computational grid in order to solve a highly complex problem that requires unprecedented amount of resources. Apparently, grid computing requires the specially purposed software that can divide and farm out pieces of a task to various nodes and then collect and synthesize the results from those nodes.

On a smaller scale, grid computing could be applied to large, commercial applications architected with multiple tiers or layers such as the front tier, application or middle tier, and backend tier. Managing such a highly complex system is no easy task. The Oracle Grid Control is designed just for simplifying the management tasks associated with an entire application stack with Oracle database as the backend. A generic Oracle Grid Control architecture is shown in Figure 2.17. At the center is the Grid Control, while multiple agents are installed on the target systems on which multiple tiers are hosted. Agents monitor and collect the state of each target system, while the Grid Control gathers and analyzes such state information, making it

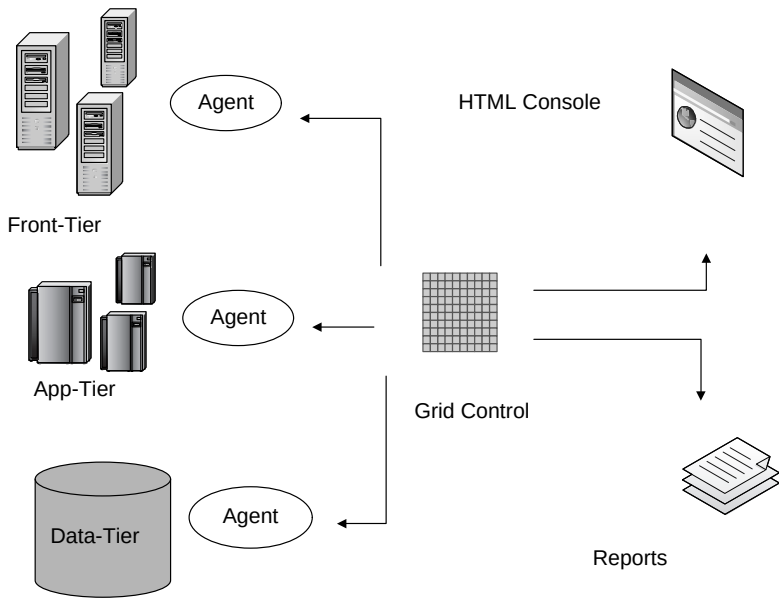


Figure 2.17 Oracle 11g Grid Control.

available to system administrators and analysts via the HTML console or Reports GUIs. Note that whether it's a Grid Control or a DB Control, the mechanism is the same. The only major difference is that the Grid Control requires installing agents on target systems and manages the entire application stack, whereas a DB Control manages an Oracle instance and database without using an agent, or in other words, the database server only. See Figure 2.18 for the differences between a Grid Control and a DB Control. This concludes our discussion on Grid Control versus DB Control.

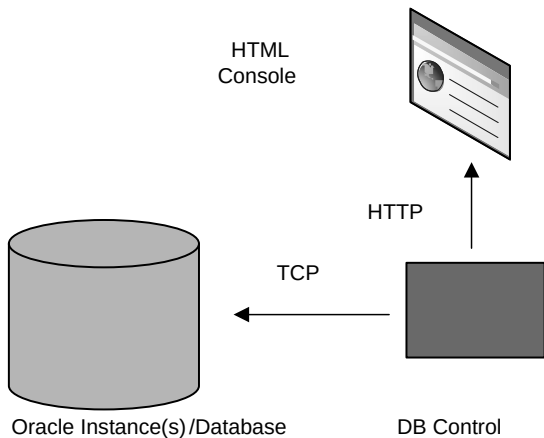


Figure 2.18 Oracle 11g DB Control.

2.6 SUMMARY

In this chapter, we covered how to install Oracle server software, how to create a listener, how to create a database, and how to install Oracle client software, and so on. These are some basic tasks that developers and test engineers may have to perform from time to time. Based on your application, you might need to select more or fewer features, but the procedure should be similar. The steps listed here should provide you with a working reference in case you run into problems with your Oracle installations.

For college students who use this book as a main text or a supplementary text for your database course, going through such an Oracle installation process will enhance your understanding of database architecture significantly. By setting up a working Oracle environment, you learn what components a typical database system like Oracle has and how those components collaborate to function as a database management system.

It's very likely that you may encounter some problems with your installation, since your system may not be exactly the same as my system. Refer to the recommended sources below if you encounter problems related to your Oracle installation.

RECOMMENDED READING

Oracle accompanies each release with a set of documents that total over 10,000 pages. For this chapter, the following document is very pertinent:

Oracle Corp., *Oracle Database Administrator's Guide*, 11g Release 2 (11.2) E17120-06 (1062 pages), October 2010, available at: http://download.oracle.com/docs/cd/E11882_01/server.112/e17120.pdf

For Oracle Grid Control versus DB Control, refer to the relevant Oracle document, for example:

Oracle Corp., *Oracle Enterprise Manager Concepts*, 11g Release (11.1.0.1) E11982-02 (260 pages), April 2010, available at: http://download.oracle.com/docs/cd/E11857_01/em.111/e11982.pdf

EXERCISES

- 2.1 Go through the first two chapters of the above Oracle Administrator's Guide to solidify the concepts you learned in this chapter.
- 2.2 If you have experiences with other non-Oracle database products, compare their installation procedure with that of Oracle.
- 2.3 Explain why sometimes it is insufficient to set the ORACLE_HOME environment variable at the command prompt level. What's the advantage of setting ORACLE_HOME environment variable at the global level over setting at the session level?

3

Options for Accessing an Oracle Server

Creativity is allowing yourself to make mistakes. Art is knowing which ones to keep.
—*Scott Adams*

This chapter discusses the options for accessing an Oracle Server. From an administrative perspective, an administrator may access an Oracle Server from the following three different approaches:

- From the Oracle Enterprise Management Java Console (OEMJC)
- From the command line interface (CLI) program SQL*Plus
- From the newer HTTP-based Enterprise Manager DB Console

From a programmer's perspective, an application can access an Oracle Server through a proper driver that defines a proper interface. For example, Oracle provides Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) drivers for non-Java and Java applications to access an Oracle Server, respectively. We will provide two case studies at the end of this chapter to demonstrate the uses of the ODBC and JDBC drivers by their respective applications. In addition, developers may access an Oracle Server using development-oriented tools such as the Oracle SQL Developer and Oracle J Developer. See Figure 3.1 for all these options of accessing an Oracle Server.

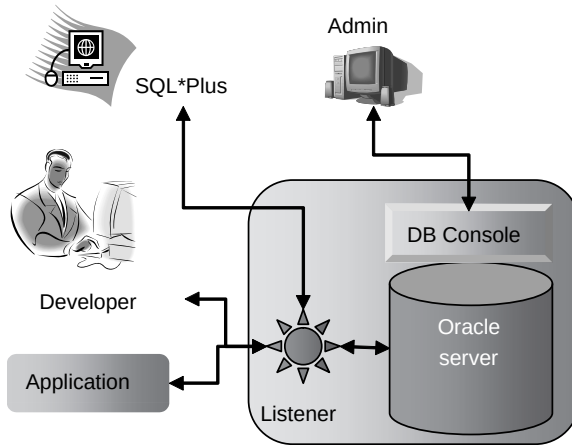


Figure 3.1 Various types of Oracle clients accessing an Oracle server.

This chapter consists of the following main sections:

- A Command Line Interface (CLI) versus a GUI-Based Console
- Oracle Enterprise Manager Java Console (OEMJC)
- Using the SQL*Plus Tool
- Oracle Enterprise Manager DB Console
- Other Tools for Developers
- Case Study: Creating ER Diagrams with Visio via ODBC
- Case Study: Accessing Oracle in Java via JDBC

Since an Oracle Server can be accessed via either a command line program or a UI-based console, let's first clarify the pros and cons of a CLI versus a GUI-based admin console in general.

3.1 A COMMAND LINE INTERFACE (CLI) VERSUS A GUI-BASED CONSOLE

Whether using a CLI or an admin console to access a server is not just a personal preference. It may have different ramifications. There is a common notion that a CLI is for full-time administrators while an admin console is for non-administrators such as developers and test engineers. My opinion is that for majority of us who are non-administrators, we need a mixed skill set in both. Here is a summary of pros and cons of both approaches:

- *Intuitiveness versus Efficiency.* A GUI-based admin console is intuitive and can be learned easily just by navigating through various menu paths if one

understands all basic concepts about how the server works. However, a CLI is much more efficient than a GUI-based admin console for many tasks. First, a CLI doesn't require you to log into a server admin console and click many times to get where you want to be. Secondly, it doesn't incur the overhead of a GUI, especially when a graphically intensive admin console is accessed remotely.

- *Availability.* Note that one can't use an admin console if it is installed on the server side but not up and running, or if it's even not installed on the server side, for example, in some production environments. A CLI does not have this limitation, because it consists of programs and scripts that are always inactive when they are not used.
- *Versatility.* There are circumstances that certain tasks can only be performed with a CLI but not with a GUI-based admin console. With a GUI-based admin console, one can only perform the tasks that the designers have put there. It's hard to image that the console designers could take every user's needs into account when they decided what to put there. With a CLI, the only limitation is the limitation of a user's own needs and creativity.
- *Rich Built-in Features.* A GUI-based admin console may have a very rich set of built-in features that are hard to reproduce with manual scripts that can only run with a CLI. One example lies with those auto-tune features built into Oracle 10g and 11g DB Consoles. I guess it takes an entire Oracle team to make them work correctly and robustly, and it's hard for any individual to write his/her own scripts to duplicate what those auto-tune features can accomplish because of the enormous complexity with the logic behind those features.
- *Simplicity and Power with a CLI.* A CLI is a very appropriate approach for repetitive, logically very simple tasks that need to be executed on a regular basis. For instance, with one of my applications, I needed to obtain the hourly rate at which a specific type of object was inserted into a very large table in every second. Since this is specific to my application, no console has a corresponding feature for accomplishing it. In this case, I wrote a simple SQL script myself, and every time when I needed to execute it, it's just one click away.
- *Risks with a CLI.* I appreciate the power and flexibility of a CLI, but I am often intimidated by its complex syntax with the fear that it's error-prone, while a GUI-based console shields away some risks inherent with any command-line driven admin tools, which accept whatever commands a user gives, but take no responsibilities for any potential consequences. An inadvertent command may cause unthought-of damages to the system, or even wipe out part of or an entire database in seconds.

Note that all those pros and cons of a CLI-based versus a GUI-based admin console discussed above apply to any client-server type of applications in general. It's necessary to be able to make a proper judgment on when to use a GUI-based admin console and when to use a CLI to accomplish a task.

Next, let's introduce the standalone Oracle Enterprise Management Java Console for accessing and managing Oracle database servers. The OEMJC is my favorite, although it's no longer included in the Oracle 11g client software and is being phased out in favor of the HTTP-based EM DBConsole.

3.2 THE ORACLE ENTERPRISE MANAGER JAVA CONSOLE (OEMJC)

The OEMJC is a standalone Java console for managing an Oracle database server. It is more convenient and intuitive than the command line `SQL*Plus` tool and less cluttered than the newer HTTP-based EM DBConsole. It was included in the Oracle 10g client software and backwards but has been excluded in the Oracle 11g client software. However, we still cover it here for two reasons. First, you can use the OEMJC to access all latest versions of Oracle database server including 11g. Secondly, the OEMJC does a much better job in showing what an Oracle database is composed of, because of its tree structure that sorts out objects in a very natural way.

To install the OEMJC from the Oracle 10g client software, choose the `Administrator` option out of the four options of `Administrator`, `InstantClient`, `Runtime` and `Custom`. This installation process is similar to the Oracle 11g client installation we described in the previous chapter. You can complete the installation by following all default options.

After installed, the OEMJC can be accessed from the navigation path of `Start->All Programs -> <Oracle Client Home> -> Enterprise Manager Console` on a Windows system. Then, expand the `Network` node, right-click the `Databases` node and select `Add Database To Tree ...` to start adding your Oracle database server to the OEMJC. Enter the `Hostname`, `Port Number` - if different from the default value of 1521, and `SID`, and then click `OK` to complete it. After this step, you can find a `connect descriptor` defined for your database in a file named `tnsnames.ora` in the folder of `<client_install_dir>\network\admin`. The standard format for a `connect descriptor` is as follows (change the values in brackets to suit your environment):

```
<connect_identifier> =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP) (Host = <hostname>) (Port = <port>))
  )
(CONNECT_DATA =
  (SERVICE_NAME = <service_name>)
)
)
```

For your reference, the following is my `tnsnames.ora` file on one of my systems I installed Oracle 11g:

```
ORA11GR1 =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP) (HOST = blue_sky) (PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = Oracle11GR1)
  )
)
```

Note that the first entry `ORA11GR1` is a connect identifier (or connect string) that resolves to a connect descriptor in a `tnsnames.ora` file. The last entry `SERVICE_NAME` is an alias for the underlying database that it represents as a service. You might find that it's confusing to have those multiple terms just to refer to an Oracle database, for example, Global Database Name, `SID`, and `SERVICE_NAME`. If you have just one instance for an Oracle database, all these terms are interchangeable. For example, I changed `SERVICE_NAME` to `SID` in the above TNS descriptor to make it look like `SID = Oracle11GR1` and I could still use the same connect identifier to connect to my database. However, if you have multiple instances or nodes for a single database, then each instance has a different `SID` value, while the Global Database Name remains a single unique entry, and the `SERVICE_NAME` remains a logical alias at the database level, but not at the instance level.

As we discussed in the previous chapter, you can try the command `tnsping <connect_identifier>` at an MS-DOS command prompt to test the connection to your database. If you get a reply of "OK (xx msec) ," that means your database is reachable; otherwise, you would get error messages indicating the errors. Of course, you can test the connection with the SQL*Plus command of `sqlplus <user_name>/<password>@<your_connect_identifier>` as well.

The OEMJC depends on a connection descriptor entry in the `tnsnames.ora` file to find all the information it needs to connect to your Oracle database, and so does your application. Check this file first whenever you cannot connect to your database irrespective of which client tool is used.

To connect to your Oracle Server with the OEMJC, double-click on your Oracle Server under the Databases node, and log in with your user name and password, for example, with your `system` account. Then you should see a display similar to Figure 3.2, which was taken with one of my 11g installs. Note from the right pane that the connect identifier `ORA11GR1` is also the name of the database. So a more consistent picture seems to be that an Oracle `SID` is an instance identifier, and a service name is a database identifier associated with a given `SID`, whereas a connect identifier is a database identifier as well but from a local point of view. One can't change the `SID` and service name after installation, but can change the connect identifier at will as long as it's unique in the local `tnsnames.ora` file.

In addition to the top and left-most tool bars shown in Figure 3.2, the most valuable feature of this tool is that it presents an entire database, as shown in the left pane, under

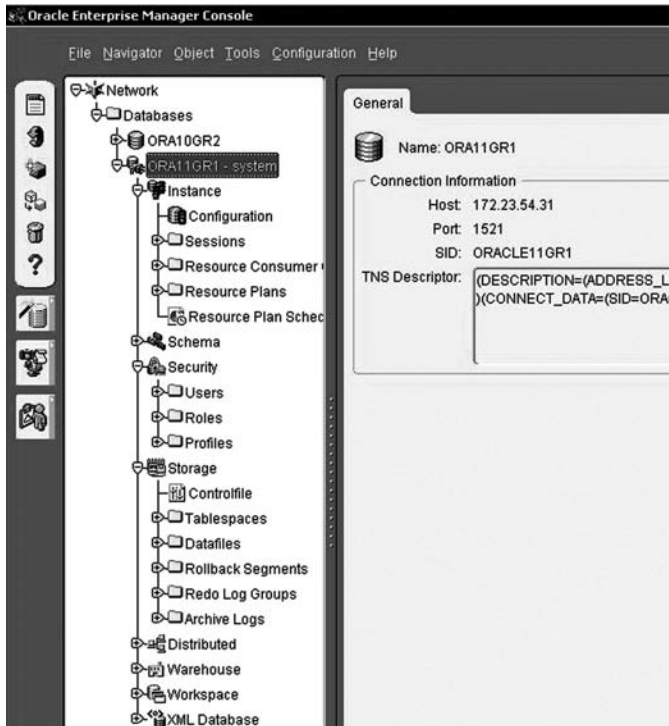


Figure 3.2 Oracle Enterprise Manager Java Console (OEMJC) connected to an Oracle 11g server, showing important constituent elements of Instance, Schema, Security, and Storage, and so on.

four simple nodes: Instance, Schema, Security, and Storage. These are the most basic elements of an Oracle database. The child nodes of each node help drill down into more detailed levels as follows (later we will elaborate more on the child nodes introduced briefly here):

- **Instance.** Expand this node and click Configuration. Then you'll see a button at the bottom in the right pane named All Initialization Parameters... This is where you can view all database instance initialization parameters and make changes with proper privileges of the logged-on user.
- **Schema.** This is where you can browse and manage the logical database objects such as tables, indexes, and views, and so on, associated with your application. Note that schema objects are organized by users, and users are organized under the Security node.
- **Security.** This is where you can manage users, roles, and profiles, and so on. Note that roles and profiles are assigned to users to help define the privileges for a user.
- **Storage.** This is where you can view and manage data files for a database. For example, further expand the Tablespaces node under Storage, and you can view the disk space usage for each tablespace. You may need to check this place

frequently to make sure no tablespace is approaching 100% full. When a tablespace becomes full and auto-expand is not enabled on the data files, your application may stop functioning normally—sometimes even without giving a warning.

However, the OEMJC doesn't have the necessary logic built in for carrying out the tasks that it was not designed for. For such tasks, it's more convenient to use the command-line driven `SQL*Plus` tool, as we will see next. The OEMJC is convenient when you want to browse various objects visually and decide what to do accordingly. The command-line tool `SQL*Plus` is more convenient for some tasks that are performed repeatedly, for example, exporting and importing a database, taking database snapshots, and creating AWR reports, and so on. This is the subject of the next section.

3.3 USING THE `SQL*PLUS` TOOL

Every type of software server is often supplemented with both a command-line tool and a GUI-based admin console for accessing and managing the server. For an Oracle database server, the command line driven tool is `SQL*Plus`. For those who are more familiar with other types of database servers on the market, `SQL*Plus` for Oracle is equivalent to:

- “db2” for IBM DB2
- “sqlcmd” for Microsoft SQL Server
- “mysql” for MySQL
- “psql” for PostgreSQL
- “isql” for Sybase

`SQL*Plus` comes with both the Oracle client software and server software. On the database server side, it is available from the `<install_dir>/product/11.1.0/db_1/BIN` directory, while on the client side, it is available from the `<client_install_dir>/bin` directory. One might run into difficulties with the use of `SQL*Plus` if both an Oracle client and an Oracle Server are installed on the same system. In that case, you need to check which one takes the precedence in the `PATH` environment variable. If you simply open up an MS-DOS command prompt and run `sqlplus` without preceding it with the path you intended to use, you might encounter errors, simply because you are not invoking `SQL*Plus` from the right path. Therefore, before using this tool, make sure the tool you are going to use is from the correct path out of the two options of the server path and client path. In fact, the `SQL*Plus` programs from both paths are the same—the difference is their respective `tnsnames.ora` file. You need to know which `tnsnames.ora` file contains the connect identifier you intend to use.

`SQL*Plus` can be run both locally on the Oracle server and remotely from a client machine, depending on the tasks you perform. For example, to take database

snapshots and create AWR reports, it's better to run SQL*Plus remotely on your desktop so that the reports will be stored on your desktop system for better bookkeeping. To export and import a database, however, it's better to run SQL*Plus on the database server so that the database export files can be stored and accessed locally on the database server.

For some management tasks, for example, unlocking an account, you must run SQL*Plus locally on the database server. In this case, make sure the environment variables ORACLE_BASE, ORACLE_HOME, and SID are set properly. If these environment variables are not set, they can be set by executing the following commands at a Windows MS-DOS command prompt, assuming that your Oracle Database Server is installed at <install_dir>/product/11.1.0/db_1, for example:

```
cmd>set ORACLE_BASE=<install_dir>
cmd>set ORACLE_HOME=%ORACLE_BASE%/product/11.1.0/db_1
cmd>set ORACLE_SID=<your SID>
```

To verify if an environment variable is set, for example, ORACLE_SID, execute the command

```
cmd>echo %ORACLE_SID%
```

Note that <install_dir> is the installation location specified when the Oracle database server was installed. In my case, the path c:\app\henry was specified, so the command to set the ORACLE_BASE environment variable would look like:

```
cmd>set ORACLE_BASE=C:/app/henry
```

After all the environment variables are set, you can execute the following two commands to unlock an account of the user named `username` (for example, if you need to unlock `system` account, replace `username` with `system` verbatim):

```
%ORACLE_HOME%/bin/sqlplus "/as sysdba"
SQL>ALTER USER ACCOUNT UNLOCK;
```

To check the status of all user accounts, execute the following command:

```
SQL>SELECT username, account_status from DBA_USERS;
```

You can consider you are logging in with the root privilege with the above command. This is the most powerful account of all Oracle users, and you should use it with caution. The next two accounts that have admin privileges are `sys` and `system`. The `sys` and `system` accounts have the default passwords of `change_on_install` and `manager`, respectively. You can change the password of a user

account by first logging in with `sqlplus "/as sysdba"` and then executing the command:

```
SQL>ALTER USER <username> IDENTIFIED BY <new_password>;
```

In the above command you should replace `<username>` and `<new_password>` with your entries verbatim. Note that you should exercise caution when changing the password of the user account that your applications use to connect to Oracle while the application is running. After you change the password of such an account, your application may stop working.

For non-account-administrative tasks, you should use the following command, instead:

```
cmd>sqlplus <username>/<password>@<your_connect_identifier>
```

Pay attention to the privilege of the user account you use to log in. For `sys` account, you are required to add `"as sysdba"` to the above command. The `system` account does not have this requirement. Also refer to the previous section about the `connect identifier` definition in a `tnsnames.ora` file.

There are many powerful and convenient SQL*Plus commands for querying and managing an Oracle database. One of such commands is the `DESC[RIBE]` command, which allows you to explore the definitions of some objects in Oracle. It applies to Oracle tables, views, synonyms, functions, packages, and so on. Its syntax is `DESC <table>`, for example, if your object to be queried is an Oracle table. It gives a quick view of all attributes or columns as well as the corresponding types for the database object you are interested in. Since this is a SQL*Plus command instead of a SQL statement, you don't need to terminate it with a semicolon. Of course, you need to log in with the proper user account before you can execute the `DESC` command.

To know more about how to use SQL*Plus with Oracle, refer to Appendix B, "Using SQL*Plus with Oracle." Next, let's introduce the HTTP-based EM DBConsole that can be installed when an Oracle database is created.

3.4 ORACLE ENTERPRISE MANAGER DBConsole

The Oracle EM DBConsole is the Oracle database server admin tool recommended by Oracle for accessing and managing Oracle 10g, 11g, and maybe future releases for some time. It's an HTTP-based UI that can be accessed via `https://<hostname>:1158/em` where 1158 is the default TCP port number. In addition, it is configured to be accessed with `https` instead of `http`. Depending on the configuration of your system, you might encounter the following problems:

1. *The service OracleDBConsole<SID> cannot be started up from the Windows Service snap-in.* The EM DBConsole depends on this service to

function properly. If you encounter the similar problem as I did, try to start it up from the command line with the command `%ORACLE_HOME%/bin/emctl start dbconsole` so that you can see the error messages explicitly. In my case, I had to enter an <IP> Hostname entry in the `hosts` file located at `C:\WINDOWS\system32\drivers\etc`. You may or may not need to do this, depending on the network configuration of your system.

2. *Use HTTP protocol.* If you do not want to run your EM DBConsole with the `https` protocol, for example, when your systems are already in a secured internal corporate LAN and you want faster responses, you can change it to use the plain HTTP protocol by executing the command `%ORACLE_HOME%/bin/emctl unsecure dbconsole`.

Although the OEMJC is better organized and more streamlined than the new EM DBConsole, the latter has a lot more new features than the former and also stays up to date with the newer Oracle releases. The preference is between a flat structure (HTML pages with EM DBConsole) and a tree structure (OEMJC). The EM DBConsole seems to be more cluttered with its flat structure, but you'll get used to it after becoming familiar with its ins and outs.

The next section discusses other Oracle tools for developers.

3.5 OTHER TOOLS FOR DEVELOPERS

In addition to the client tools introduced above, some other tools are available to facilitate SQL and application development with Oracle, including:

- *Oracle SQL Developer.* This is a graphical tool that lets you view, create, edit, and delete (drop) database objects, edit and debug PL/SQL code, run SQL statements and scripts, manipulate and export data, and create and view reports. You can also use this tool to connect to schemas of non-Oracle databases, such as Microsoft SQL server and MySQL, and so on.
- *Oracle JDeveloper.* This is an IDE (integrated development environment) supporting developing applications in Java, Web Services, and SQL. It can be used for executing and tuning SQL statements as well. It has a visual schema diagramming tool as a database modeler.

In addition, the Oracle Call Interface (OCI) and Oracle pre-compilers allow standard SQL statements to be embedded within a procedural programming language. A detailed coverage of those tools is beyond the scope of this text.

Accessing an Oracle database with the OEMJC, SQL*Plus or EM DBConsole is visually intuitive. However, accessing an Oracle database from a client program via ODBC or JDBC may not be so intuitive unless you have some basic programming experience. The next two sections provide two case studies to help illustrate how an Oracle database can be accessed from a client program via ODBC and JDBC,

respectively. These two examples are practical as well. The first one shows how you can create Entity-Relational diagrams (ERDs) for Oracle schemas using Microsoft Visio. This might save you some time when you want to create an ER diagram for an Oracle schema but you don't have a database modeling software program other than Visio installed on your system. The second example demonstrates how to connect to your Oracle database in Java and execute various SQL statements against your Oracle database.

Let's begin with the ODBC connection case study first in the next section.

3.6 CASE STUDY: CREATING ER DIAGRAMS WITH VISIO VIA ODBC

First, let's explain what an ER diagram is. An ER diagram in the context of a relational database is an abstract and conceptual representation of database tables. This will become clear after we generate an ER diagram with one of the sample schemas, HR, that comes with Oracle 11g.

Then what is ODBC? ODBC stands for Open Database Connectivity. It's an open interface that allows programs written in .Net, C, C++, Perl, PHP, Python, and so on, to access major types of databases on the market. To enable a client program to access a specific type of database, one needs to have the proper ODBC driver available to the client application.

Before you start, make sure that you have Oracle client software installed on the same system as your Visio is installed. This ensures that the Oracle ODBC driver will be available to Visio. Also verify with SQL*Plus that you can connect to your Oracle database from which you want to create ER diagrams. You may need to create a `tnsnames.ora` file in your Oracle client's `%ORACLE_HOME%\NETWORK\ADMIN` directory with the TNS descriptor copied over from your Oracle server.

After starting up Visio (in my case Visio 2007), follow the below procedure to create an ER diagram for the Oracle sample schema HR:

1. Select from File: New → Software and Database → Database Model Diagram. You may want to uncheck Grid view if it's on.
2. Then navigate through Database → Options → Drivers.... Select Oracle Server, click Setup and you should see your Oracle 11g client installed and checked as shown in Figure 3.3. Now click Cancel until you return to the normal drawing pane.
3. Then select → Database Reverse Engineer...to bring up the Reverse Engineer Wizard, as shown in Figure 3.4. Now you should see your connect identifier listed under Data sources. If yes, click Next, and you should see the Connect Data Source dialog as shown at the bottom of Figure 3.4. If not, click on New and the wizard will guide you through the process of creating a data source.

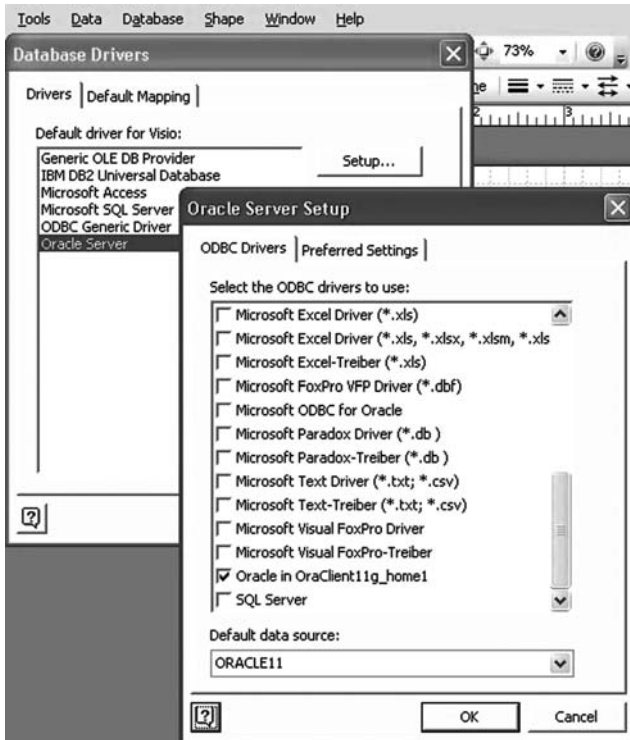


Figure 3.3 Generating an Oracle ER diagram with Visio via ODBC: database drivers and oracle server setup.

4. Enter the password for the HR schema and click **Next**. You should see the **Select object types to reverse engineer** dialog as shown in Figure 3.5. Make sure you select only the check boxes for **Tables**, **PK's** and **FK's**.
5. Click **Next** and you should see the **Select tables and/or views to reverse engineer** dialog as shown in Figure 3.6. Select all seven tables and click **Next**.
6. The **Reverse Engineer Wizard** now asks if you want to add the shapes to the current page. Select **Yes** and click **Next**.
7. Review the selections of the tables and catalog info and click **Finish**.
8. After a few minutes, you should see an ER diagram as shown in Figure 3.7 for the HR schema. You can save and exit Visio now.

You can follow the same procedure and create ERDs for other schemas. This concludes our case study of creating an Oracle ERD with Visio via ODBC. Next, we'll go through the steps of connecting to an Oracle database via JDBC in a Java client program.

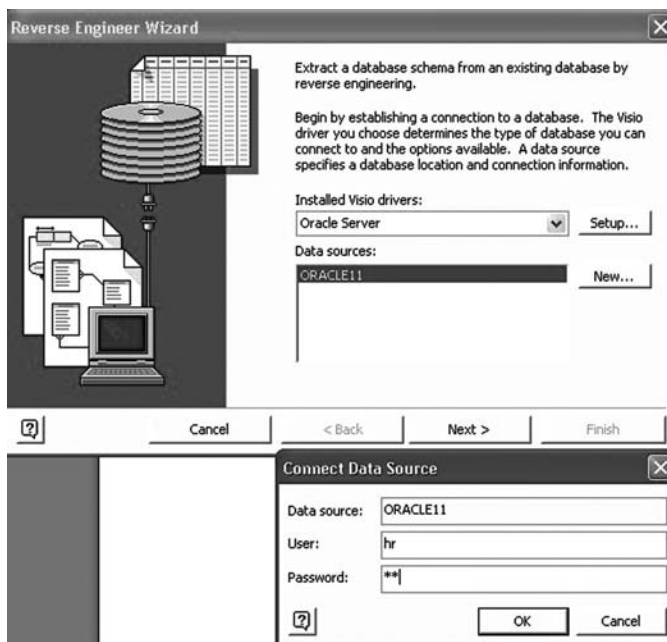


Figure 3.4 Generating an Oracle ER diagram with Visio via ODBC: reverse engineer wizard and connect data source.

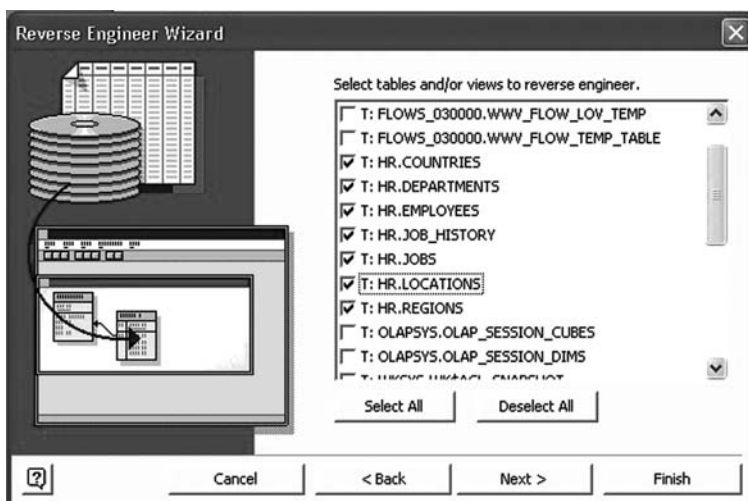


Figure 3.5 Generating an Oracle ER diagram with Visio via ODBC: select object types to reverse engineer.

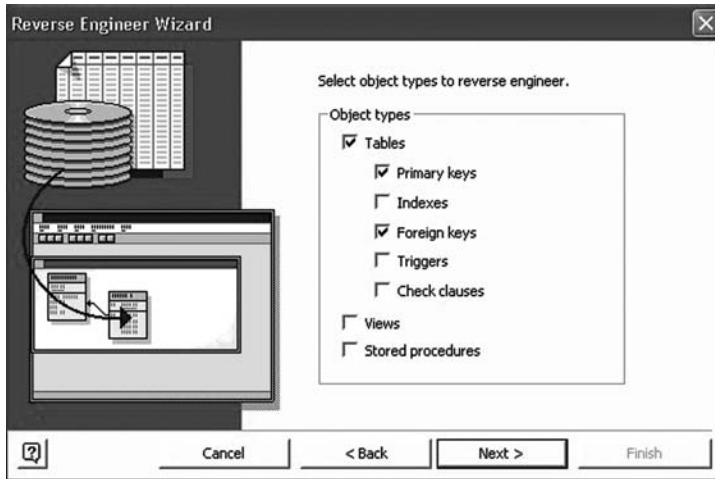


Figure 3.6 Generating an Oracle ER diagram with Visio via ODBC: select tables and/or views to reverse engineer.

3.7 CASE STUDY: ACCESSING ORACLE IN JAVA VIA JDBC

The following code snippet demonstrates how to establish a connection to an Oracle database via JDBC. The code logic is as follows:

- First, an Oracle data source object is created with the statement `ods = new OracleDataSource () ;`.
- Then the URL, user (or schema), and password information is provided and a connection is returned to the `Connection` object `conn`.
- With the created connection to the Oracle database, a SQL query is issued and the result set is printed out.

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import oracle.jdbc.pool.OracleDataSource;

class JdbcTest {
    public static void main(String args[]) throws SQLException {
        OracleDataSource ods = null;
        Connection conn = null;
        Statement stmt = null;
        ResultSet rset = null;
        // Create DataSource and connect to the remote database
        ods = new OracleDataSource ();
        ods.setURL(
            "jdbc:oracle:thin:@//172.23.41.86:1521/Ora11GR1");
```

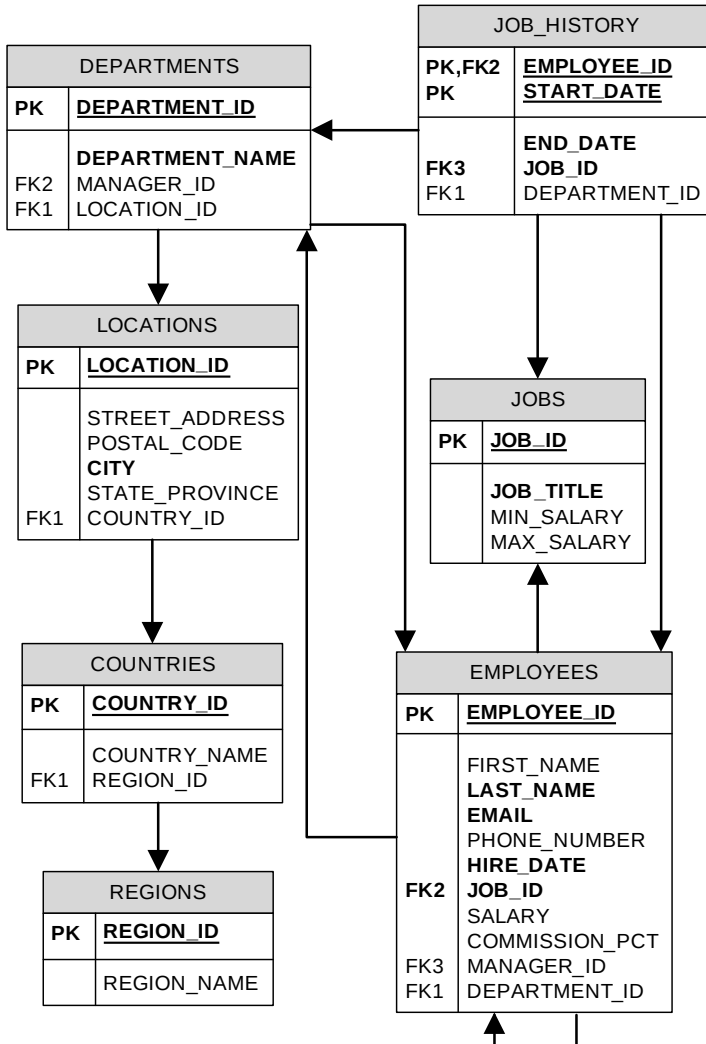


Figure 3.7 Generating an Oracle ER diagram with Visio via ODBC: generated ERD of the HR schema.

```

ods.setUser("HR");
ods.setPassword("hr");
conn = ods.getConnection();
try {
    // Query the employee last names
    stmt = conn.createStatement();
    rset = stmt.executeQuery
        ("SELECT last_name FROM employees");
    // Print the name out
    while (rset.next())
  
```

```

        System.out.println(rset.getString(1));
    }
    // Close the result set, statement, and the connection
    finally {
        if (rset != null)
            rset.close();
        if (stmt != null)
            stmt.close();
        if (conn != null)
            conn.close();
    }
}
}

```

Perhaps the most interesting line in this code example is the parameter of "jdbc:oracle:thin:@//172.23.41.86:1521/Ora11GR1" passed to the `ods.setURL` method. Note the part of `jdbc:oracle:thin` tells the program to use the Oracle thin driver which in this case is a file named `ojdbc6.jar` downloadable from Oracle's Web site. The part of `172.23.41.86:1521` indicates the IP and TCP port of the Oracle server, separated by the colon sign. The last part of `Ora11GR1` is simply the SID of the Oracle Server.

To compile and run the above program, add `-cp .;./ojdbc6.jar` to the `javac.exe` and `java.exe` programs as follows:

```

cmd> javac -cp .;./ojdbc6.jar JdbcTest.java
cmd> java -cp .;./ojdbc6.jar JdbcTest

```

Then you should see that all of the last names of the Employees table are printed out. Note that after a JDBC connection is established successfully, one can issue various SQL statements as needed. This concludes our case study of accessing an Oracle server via JDBC.

3.8 SUMMARY

In this chapter, we reviewed all the options of accessing an Oracle Server from the client side. A few major points are listed below:

- The old Oracle Enterprise Manager Java Console has been excluded starting with Oracle 11g, giving its throne to the newly introduced HTTP-based EM DBConsole. The OEMJC has a better tree-like structure, while the new EM DBConsole has far more features built in and should be relied upon for accessing and managing newer Oracle releases such as 10g and 11g and going forward.
- When using SQL*Plus to access and manage your Oracle database, make sure that it's invoked from the proper path. Also make sure you have a proper

connection descriptor set in your `tnsnames.ora` file. Whenever encountering problems with your SQL*Plus tool, first of all, check the connectivity to your Oracle server with the command `tnsping <your_connection_string>`. If you can't `tnsping` your database, then you can't connect to your database with SQL*Plus. You need to troubleshoot and resolve your problem before you can connect to your database via SQL*Plus.

- You may need to set an `ORACLE_HOME` environment variable at the global level rather than at the command proper level. This is especially important when you install an application that needs to connect to your Oracle database.

The next chapter offers a quick tour of an Oracle Server using an Oracle 11g installation. Some of the Oracle server access options introduced in this chapter will be used. Specifically, I'll give you a tour of an Oracle Server using both the OEMJC and the EM DBConsole to help you get a good understanding of Oracle from multiple perspectives. At this point, you might have an EM DBConsole running as well with your Oracle installation to follow through the illustrations in the next chapter. If you don't have an OEMJC installed, don't worry—just use the screenshots given in the next chapter as a convenient means for illustrating various Oracle concepts and components.

RECOMMENDED READING

To learn more about SQL*Plus, refer to the relevant official Oracle documentation released with each version of Oracle. With 11g, for example, refer to the following document from Oracle:

Oracle Corp., *SQL*Plus User's Guide and Reference*, Release 11.1 B31189-01 (428 pages)
http://download.oracle.com/docs/cd/B28359_01/server.111/b31189.pdf

There is also a SQL*Plus Quick Reference from Oracle, as listed below, for example, with 11g:

Oracle Corp., *SQL*Plus Quick Reference*, Release 11.1 B31190-01 (18 pages)
http://download.oracle.com/docs/cd/B28359_01/server.111/b31190.pdf

EXERCISES

- 3.1** Use the `tnsping` command to check the connectivity to your database. Then test `sqlplus` with your database to make sure you can connect to your database.
- 3.2** Log onto your database with the EM DBConsole. Navigate through the various tabs at the top and identify the links that are most interesting to you.
- 3.3** Try out the commands provided in the text to lock and unlock the sample account "Scott."
- 3.4** Try the SQL*Plus command `DESC SCOTT.EMP` and notice what information you can get on this object.

- 3.5** How do you look up the connect identifiers on a system with Oracle installed?
- 3.6** If your EM DBConsole is configured to use HTTPS protocol, how do you change it to using the plain HTTP protocol?
- 3.7** Explain the concepts of a TNS descriptor, a connect identifier, a service name, and a SID. What are the major differences among a connect identifier, a service name, and a SID, conceptually?

4

A Quick Tour of an Oracle Server

*Painting is poetry that is seen rather than felt, and poetry is painting that is felt
rather than seen.*
—Leonardo da Vinci

The purpose for this quick tour is to walk you through all major elements of an Oracle Server. At the end of the tour, you will gain not only an inside-out understanding of what an Oracle Server consists of but also the ability to correlate various Oracle objects with various data items of your application. For example, if your application manages the orders of your customers, you will understand how orders, customer information, and so on, are represented and stored in an Oracle Server.

Before starting the tour, let's get familiar with the new sample schemas installed with Oracle 11g. Those new sample schemas to be introduced in the next section are far more serious than the original schema named "Scott," which still is included with Oracle 11g. We will rely on those sample schemas to explain various Oracle concepts throughout this chapter.

This chapter consists of the following main sections:

- New Oracle Schemas beyond "Scott"
- Oracle Users versus Schemas
- Tablespaces, Segments, Extents, and Data Blocks

- Tables, Indexes, and Index Types for Structured Data
- Domain and LOB Index Types for Unstructured Data
- Views, Materialized Views, and Synonyms
- Stored Procedures, Functions, and Triggers
- Referential Integrity with Foreign Keys

Let’s start with introducing new Oracle schemas beyond “Scott” next.

4.1 NEW ORACLE SCHEMAS BEYOND “SCOTT”

Several new sample schemas are available in Oracle 11g for demo purposes. This set of interlinked schemas encompasses a lot of Oracle concepts that are necessary not only for understanding Oracle architecture in general but also for being able to carry out actual Oracle performance and scalability work more efficiently. Each of those schemas is briefly introduced as follows (note that some terminologies might sound unfamiliar to you here, but don’t worry about them, as they will be covered soon):

- *The Human Resources (HR) Schema.* This is the simplest schema of all six sample schemas. It’s handy for explaining some very basic concepts like users, tables, indexes, views, and so on. It also includes more complicated schema object types, such as procedures, sequences, LOBs, synonyms, triggers, and so on.
- *The Order Entry (OE) Schema.* This schema is more complex than the HR schema. It includes the object types of function, index, LOB, sequence, synonym, table, trigger, type, type body, and view.
- *The Online Catalog (OC) Schema.* This is a subschema with a collection of object-relational database objects built inside the OE schema.
- *The Product Media (PM) Schema.* This is dedicated to multimedia types using LOBs.
- *The Information Exchange (IX) Schema.* This is a main schema that has a set of member schemas, mainly for demonstrating Oracle Advanced Queuing component. The schema object types of this schema include evaluation context, index, LOB, table, queue, sequence, rule set, and so on.
- *The Sales History (SH) Schema.* This is a data warehouse type of schema with more data. The schema object types of this schema include dimension, index, index partition, LOB, materialized view, table, table partition, view, and so on.

You can verify the schema object types by executing the following command with <SCHEMA> replaced with a schema name such as HR, OE, OC, IX, SH, and so on:

```
SQL>select distinct object_type from dba_objects
      where owner = '<SCHEMA>' order by object_type asc;
```

For example, executing the above command with the IX sample schema resulted in the following output for the object types contained in this schema:

```
OBJECT_TYPE
-----
EVALUATION CONTEXT
INDEX
LOB
QUEUE
RULE SET
SEQUENCE
TABLE
TYPE
VIEW
```

9 rows selected.

Next, let's introduce the two most fundamental concepts for Oracle: users and schemas.

4.2 ORACLE USERS VERSUS SCHEMAS

An Oracle instance needs a SID to identify itself. Similarly, any user who wants to gain access to Oracle needs an ID and password. In the OEMJC, all users are listed under the Users node. Figure 4.1 shows the user HR for the HR sample schema installed with an Oracle 11g server. Screenshot (a) shows the authentication and tablespace associated with the user HR, whereas screenshot (b) shows the available Roles and the roles assigned to the user HR. The other tabs such as System and Object have similar settings, namely, granted privileges out of the privileges defined globally. Roles and privileges define the access boundaries for a user.

A schema defines all the database objects for a user. There is a 1:1 relationship between a user and a schema, and thus we sometimes just say a *schema user*. Users connect to the Oracle database server with proper credentials and privileges, and then gain various levels of accesses to their respective schemas as well as granted accesses to the schemas of other users. For example, the sample schema HR has both the user and schema named "HR." I can log into my Oracle 11g database with the user HR and read/modify the HR schema, while the user HR's access to the schemas of other users might be limited to read-only unless more privileges have been granted explicitly.

A schema is exclusively defined by its objects grouped by various object types. With the OEMJC, click a schema node in the left pane, and all the associated objects appear in the right pane along with the object types for various objects. This can be verified by examining Figure 4.2, which shows the HR Schema with its objects displayed in the right pane. Under the HR Schema node, some of its table and index

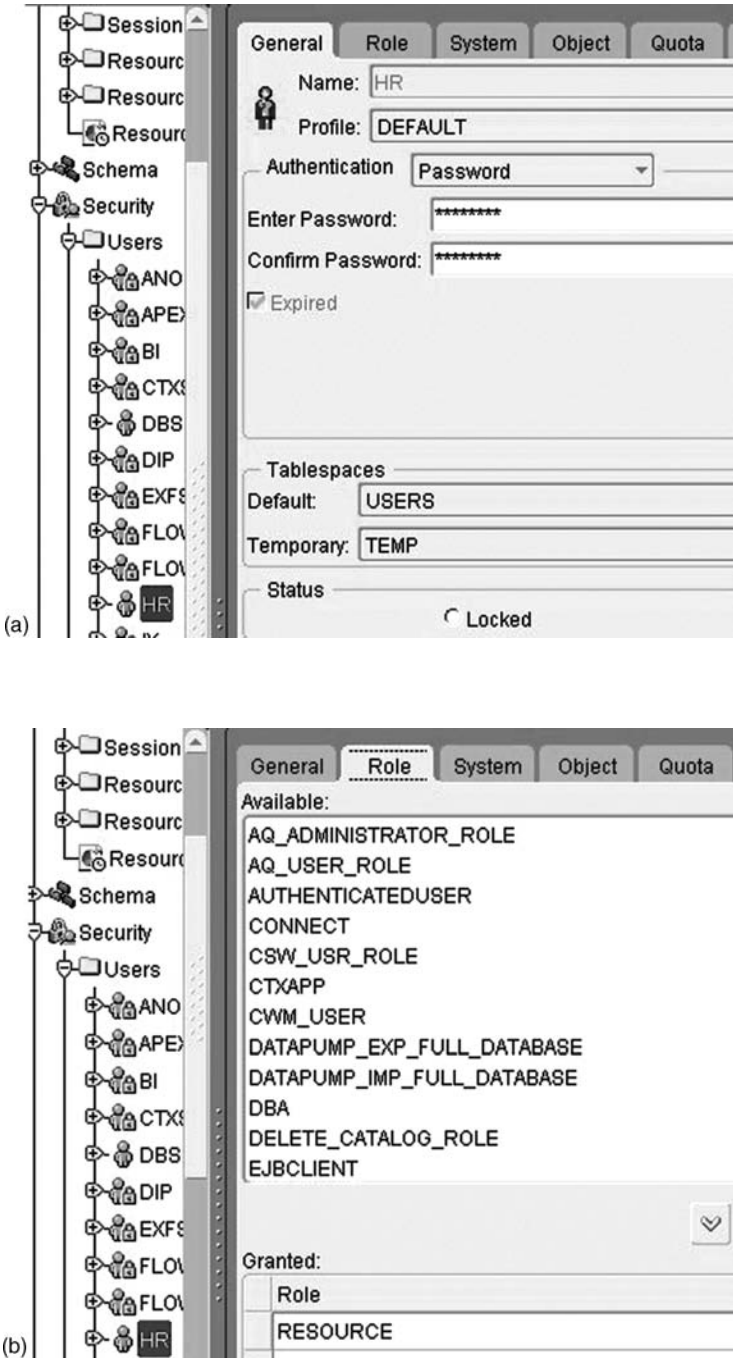
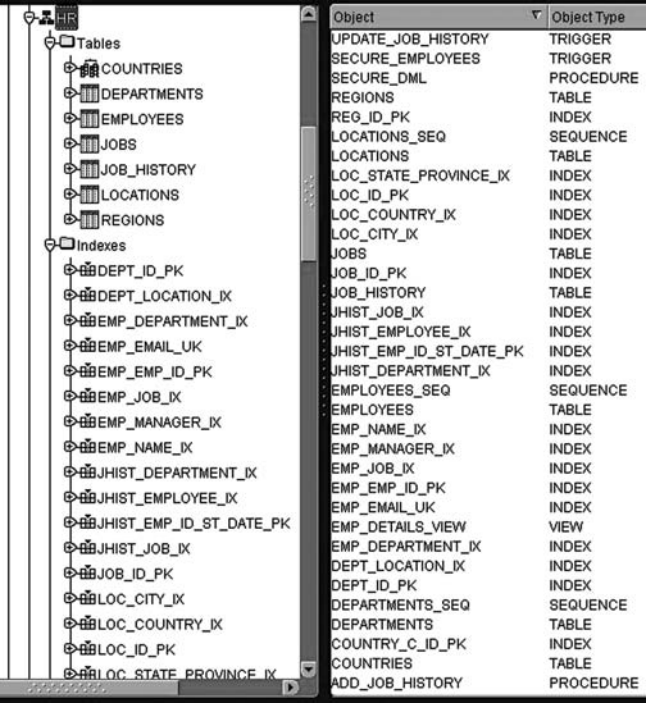


Figure 4.1 Oracle 11g HR sample schema: (a) authentication and tablespace assigned to user HR; (b) available roles and roles assigned to user HR.



Object	Object Type
UPDATE_JOB_HISTORY	TRIGGER
SECURE_EMPLOYEES	TRIGGER
SECURE_DML	PROCEDURE
REGIONS	TABLE
REG_ID_PK	INDEX
LOCATIONS_SEQ	SEQUENCE
LOCATIONS	TABLE
LOC_STATE_PROVINCE_IX	INDEX
LOC_ID_PK	INDEX
LOC_COUNTRY_IX	INDEX
LOC_CITY_IX	INDEX
JOBS	TABLE
JOB_ID_PK	INDEX
JOB_HISTORY	TABLE
JHIST_JOB_IX	INDEX
JHIST_EMPLOYEE_IX	INDEX
JHIST_EMP_ID_ST_DATE_PK	INDEX
JHIST_DEPARTMENT_IX	INDEX
EMPLOYEES_SEQ	SEQUENCE
EMPLOYEES	TABLE
EMP_NAME_IX	INDEX
EMP_MANAGER_IX	INDEX
EMP_JOB_IX	INDEX
EMP_EMP_ID_PK	INDEX
EMP_EMAIL_UK	INDEX
EMP_DETAILS_VIEW	VIEW
EMP_DEPARTMENT_IX	INDEX
DEPT_LOCATION_IX	INDEX
DEPT_ID_PK	INDEX
DEPARTMENTS_SEQ	SEQUENCE
DEPARTMENTS	TABLE
COUNTRY_C_ID_PK	INDEX
COUNTRIES	TABLE
ADD_JOB_HISTORY	PROCEDURE

Figure 4.2 Oracle 11g HR sample schema: left frame: tables, indexes, and so on; right frame: objects, and object types (trigger, procedure, table, index, and sequence, etc.).

objects are expanded in the left pane to give you a concrete feel about the schema objects. Most commonly used Oracle schema object types include: table, index, view, trigger, procedure, synonym, sequence, cluster, LOB, dimension, and so on. We'll explore more about those object types after discussing in the next section how Oracle actually stores data for the various schema object types listed above.

4.3 TABLESPACES, SEGMENTS, EXTENTS, AND DATA BLOCKS

A database is essentially a data store. No database stores data randomly. Instead, each database product has its own elaborate data storage schemes to maximize performance and scalability. Let's see how Oracle stores data.

We can take either a bottom-up view or a top-down view into Oracle's hierarchical data storage structure. If we take a top-down view, it goes by Tablespaces → Segments → Extents → Data Blocks. Let's explore these concepts next.

Tablespaces are the top level logical storage units in Oracle for storing and organizing all system and user data, while the data blocks are the lowest-level data storage units in Oracle (also called *logical blocks*, *Oracle blocks*, or *pages*). Thus a tablespace is measured by its total storage capacity, while a data block is measured typically in the units of kilobytes or kB. Dividing a tablespace's

capacity by a data block size gives the total number of blocks contained in that tablespace. The size of a data block is controlled by the initialization parameter `DB_BLOCK_SIZE`. The default value for `DB_BLOCK_SIZE` is OS-dependent, typically 8192 bytes or 8 kB on Windows. It can vary from 2048 bytes—32768 bytes or from 2 kB to 32 kB.

A tablespace is divided into multiple segments. These are data segment, index segment, rollback segment, and so on, partially corresponding to various schema object types, and partially corresponding to various Oracle transaction tasks.

A segment is further divided into a set of extents, each of which is a contiguous area on disk for storing data blocks. Note that an extent is a contiguous storage area, whereas a segment may not be contiguous, as it may grab extents from non-contiguous areas.

The setting of a data block size may affect performance and scalability. Oracle must read persisted data off a disk device, and often write new data to the disk device as well. Let's assume the `DB_BLOCK_SIZE` parameter is set to 8 kB. Then a transaction that must write 16 kB data to a disk would need to perform 2 physical writes, while a transaction that must write 32 kB data to the same disk would need to perform 4 physical writes. If the `DB_BLOCK_SIZE` were set to 4 kB, the above two transactions would need to perform 4 and 8 physical writes, respectively. However, setting `DB_BLOCK_SIZE` to a much larger value than 8 kB may waste storage space if the average transaction data size is much smaller. As you can imagine, OLTP applications favor smaller data block sizes in the range of 2 kB to 4 kB, while batch jobs favor larger data block sizes in the range of 16 kB to 32 kB. So the default setting of 8 kB data block size is a compromise for the two drastically different types of applications.

Since different applications have different portions of batch jobs and online users, there is no one data block size that fits all. The best strategy is to come up with a representative workload, conduct tests with various data block sizes, and then decide on the optimum data block size.

You have already learned that schema objects are logical entities, and their contents need to be stored on physical disk devices with their storage areas divided into tablespaces, segments, extents, and data blocks hierarchically. In the next few sections, we'll discuss various Oracle schema object types. We'll go by structured and unstructured data.

4.4 TABLES, INDEXES AND INDEX TYPES FOR STRUCTURED DATA

As we briefly mentioned earlier, an Oracle schema encompasses various schema objects as instances of various schema object types. For all types of schema objects, refer to Figure 4.3 taken from the EM DBConsole with one of my Oracle 11g setups. As is seen, there are many types of schema objects. In this section, we'll only take a closer look at some of the commonly encountered schema object types: tables and indexes. Other more advanced schema object types will be discussed later in this chapter.

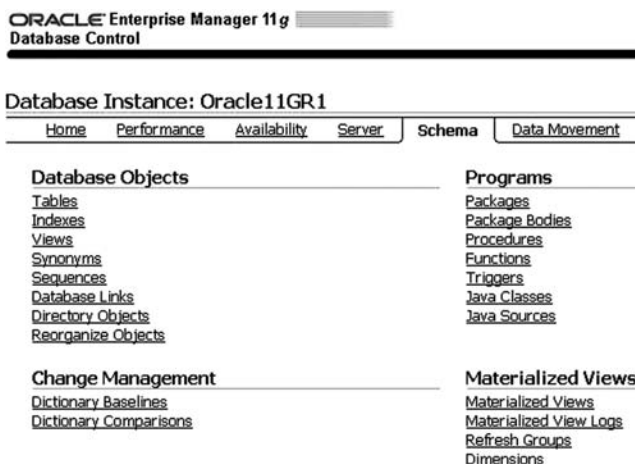


Figure 4.3 Oracle 11g HR sample schema: schema objects classified into the categories of database objects, programs, change management, and materialized views. Notice the objects in each category.

The first most important schema object type is the table type. As of Oracle 11g, there are four types of tables:

- *Ordinary or Heap Organized Table (HOT).* This is the most common type of table that table data is stored as an unordered collection (heap) with two separate segments maintained for the table and its indexes on disk.
- *Index-Organized Table (IOT).* With an IOT, rows of the table are stored in a B*Tree index structure in a primary key sorted manner. Also, an IOT contains both key columns and non-key columns. We'll explain what a B*Tree index structure is later.
- *Clustered Table.* A cluster can be formed with multiple tables that are closely related to each other by sharing common columns. The closely related data is stored in the same data block or chained with additional blocks if the total size of the data exceeds the size of a data block. A cluster is a helpful concept for frequently executed joins that access common columns of the joined tables. However, most tables are standalone without participating in any clusters.
- *Partitioned Table.* Partitioning allows data to be divided into multiple partitions so that each partition can be managed individually.

A database table has various attributes represented in columns, while the values assigned to each set of attributes constitute a row or a record. Two other concepts associated with tables and columns are the concepts of cardinality and domain. The concept of *cardinality* could mean the m to n mapping relationships among various tables or the unique values of a table column. The concept of *domain* is defined as the set of all allowable values of a column. Apparently, in the context of columns, the relationship of *cardinality* $\leq$ *domain* holds.

(a)

General					
Constraints					
Storage					
Options					
Statistics					
Constraints Storage					
Name:	EMPLOYEES				
Schema:	HR				
Tablespace:	EXAMPLE				
Table: ☒ Standard ☐ Organized Using Index (IOT)					
Columns					
Name	Datatype	Size	Scale	Nulls?	
EMPLOYEE_ID	NUMBER	6	0		
FIRST_NAME	VARCHAR2	20			✓
LAST_NAME	VARCHAR2	25			
EMAIL	VARCHAR2	25			
PHONE_NUMBER	VARCHAR2	20			✓
HIRE_DATE	DATE				
JOB_ID	VARCHAR2	10			
SALARY	NUMBER	8	2		✓
COMMISSION_PCT	NUMBER	2	2		✓
MANAGER_ID	NUMBER	6	0		✓
DEPARTMENT_ID	NUMBER	4	0		✓

(b)

General						
Constraints						
Storage						
Options						
Statistics						
Constraints Storage						
Constraint on the Table						
Name	Type	Disable?	Referenced Schema	Referenced Table	Cascade On Delete	Check Condition
EMP_DEPT_FK	FOREIGN		HR	DEPARTMENTS		
EMP_EMAIL_UK	UNIQUE					
EMP_EMP_ID_PK	PRIMARY					
EMP_JOB_FK	FOREIGN		HR	JOBS		
EMP_MANAGER_FK	FOREIGN		HR	EMPLOYEES		
EMP_SALARY_MIN	CHECK					salary > 0
Constraint Definition : EMP_DEPT_FK						
Table Columns		Referenced Columns				
DEPARTMENT_ID		DEPARTMENT_ID				

Figure 4.4 Oracle 11g HR sample schema: implementation details of a representative table named Employee: (a) General, (b) Constraints, (c) Storage, (d) Options, (e) Statistics, and (f) Constraints Storage.

To fully understand every detail about a table, a series of screenshots have been taken with a representative Employees table from the HR Schema. Figure 4.4 parts (a) to (e) illustrate how a table is implemented in Oracle with the common attributes as follows:

- **General.** As shown in Figure 4.4(a), the EMPLOYEES table belongs to Schema HR and stored in Tablespace EXAMPLE. Its type is standard (heap-organized) instead of Index-Organized. Also, the Columns section defines the table in terms of “Name,” “Data type,” “Size,” “Scale,” and “Nulls?”. In this particular

(c)

The screenshot shows the 'Storage' tab of the Oracle SQL*Plus configuration window. It includes sections for Extents, Space Usage, Number of Transactions, Free Lists, and Buffer Pool. The 'Explicit' radio button is selected under 'Extents'. The 'Initial Size' is 64 K Bytes, 'Next Size' is 0 K Bytes, 'Increase Size by' is 0 %, and 'Minimum Number' is 1. The 'Maximum Number' is set to 'Value' with a value of 2147483645. Under 'Space Usage', 'Percentage free space reserved for updates' is 10. Under 'Number of Transactions', 'Initial' is 1 and 'Maximum' is 255. Under 'Free Lists', there are empty fields for 'Free Lists' and 'Groups'. The 'Buffer Pool' is set to 'DEFAULT'.

(d)

The screenshot shows the 'Options' tab of the Oracle SQL*Plus configuration window. It includes sections for Parallel Option, Logging Option, Caching Option, and Monitoring Option. Under 'Parallel Option', the 'No Parallel' radio button is selected. Under 'Logging Option', the 'NO LOGGING' radio button is selected. Under 'Caching Option', the 'Place least frequently accessed data to the end of the buffer cache. (NOCACHE)' radio button is selected. Under 'Monitoring Option', the 'MONITORING' radio button is selected.

Figure 4.4 (Continued)

example, the column data types include NUMBER, VARCHAR2, and DATE. The Size column specifies the length for VARCHAR2 type, while the columns SIZE and Scale specify the precision (total # of numeric digits) and scale (# of digits after the decimal) for a number type. The column Nulls? simply indicates whether this column can be a null.

(e)

General	Constraints	Storage	Options	Statistics	Constraints Storage
Last Analyzed:	17-APR-2010 14:30				
Empty Blocks:	0				
Average Space:	0				
Number of Rows:	107				
Sample Size:	2000				
Average Row Length:	68				
Continued Row Count:	0				
Average Space Freelist Blks:	0				
No. of Freelist Blks:	0				

(f)

General	Constraints	Storage	Options	Statistics	Constraints Storage
Constraint Name: EMP_EMAIL_UK Tablespace: EXAMPLE					
Constraint Data Segment Storage for EMP_EMAIL_UK					
Data Segment ☐ Generate full redo Data Storage...					
Extents Initial Size: 64 K Bytes Next Size: 0 K Bytes Increase Size by: 0 % Minimum Number: 1 Maximum Number: ☐ Unlimited ☒ Value: 2147483645					
Space Usage Percentage free space reserved for updates: 10					
Number of Transactions Initial: 2 Maximum: 255					
Free Lists Free Lists: Groups:					
OK Cancel Help					

Figure 4.4 (Continued)

- Constraints.** The purpose of a constraint is for enforcing referential data integrity on the data of a table. As shown in Figure 4.4(b), constraints are composed of foreign keys (FK), unique keys (UK), primary keys (PK), and checks. In the case of FK's, the referenced schema and table are indicated as well. A PK is a unique entity that uniquely identifies a row of a table. In this example, the `EMPLOYEE_ID` is chosen as the PK for the `EMPLOYEES` table. To see how

a PK could act as a constraint, one could try to insert a new row with an existing PK value, and the result would be a constraint violation error thrown. A UK also places a constraint on a column or a combination of several columns to guarantee that the entity it represents will also be unique. A check constraint specifies a condition that must be satisfied, for example, the salary of an employee must be > 0 (The annual salary of Steve Jobs of Apple is \$1, possibly because Apple's HR database would not take it if he wanted zero annual salary.) We leave the discussion on FK to the next section when we elaborate more on referential integrity, which is an important aspect of every database.

- *Storage.* This part, as shown in Figure 4.4(c), specifies that storage management for the table is explicit, namely, allocating a fixed initial size of 64 KB for each extent and allowing it to grow up to about 2.1 billions. We'll discuss more about how Oracle manages storage later.
- *Options.* This part, as shown in Figure 4.4(d), specifies whether a table can be accessed in parallel, along with logging, caching, and monitoring options.
- *Statistics.* This part, as shown in Figure 4.4(e), summarizes the details about the statistics associated with a table. The statistic information displayed here is used by the Oracle Optimizer to determine the optimum execution plan for a query. Note the first item "Last Analyzed." The statistic information is generated by *analyzing* a database or a schema or a table or a few highly active tables. It can be done either manually or with a scheduled job to run regularly. We'll discuss more about the Oracle Optimizer and analyzing the whole or part of a database later.
- *Constraints Storage.* The content of this tab is similar to that of the Storage tab, and there is not much information here.

Note that we have indirectly introduced the concept of an index when we mentioned the table type of an index-organized table. Let's discuss next the second most fundamental schema object type: indexes.

Indexing helps speed up search queries with search keys based on certain criteria. An index essentially covers an entire or part of a search key with each search key value as an entry in an index. However, only properly designed indexes can help speed up queries. Improperly indexing may do nothing to speed up a query or even hurt the performance of INSERT/UPDATE type of SQL statements. Here are some common practices recommended for determining what to index:

- *Indexing Primary Keys.* Good table designs tend to use numerically typed numbers for PKs. However, one has to be careful that when the PK of a table is indexed, a full table scan might be invoked to verify whether the value of the PK exists in the table every time an INSERT SQL statement is executed. One instance is with a poorly written query like "SELECT MAX (PK) FROM table;" to determine the next PK ID. The best strategy is to use Oracle's sequence generator feature to get around such problems. Note that Oracle automatically creates indexes on PKs but not on FKs. Indexes on FKs are created manually if necessary.

- *Indexing Foreign Keys.* In general, it's desirable to create indexes on FKs. FKs are contained within child or referencing tables, while PKs are contained in parent or referenced tables. When a row is updated or deleted in the parent table, Oracle would search the child table for FKs. If FKs are indexed, searching for FKs will be much faster and locks on child tables could be released much sooner, thus preventing locking contentions from occurring.
- *Composite Indexes.* A composite index is created by combining more columns in a table with the primary key column included as well. Because of the inclusion of the primary key, a composite key is also a primary key. Other terms for a composite key include a *concatenated key* or an *aggregate key*. Composite indexes are created to match all or the leading part of the WHERE clause conditions of a SELECT statement to speed up the query. Regarding the order of the columns appearing in a composite index, Oracle recommends that the most commonly accessed or most selective columns take precedence over those columns that are less commonly accessed or less selective.
- *Indexing Non-Key Columns.* Non-key columns can also be indexed if necessary. Indexes on using a combination of columns that do not include referential integrity constraints (PKs and FKs) are called *alternate* or *secondary indexes*. However, use your discretion with the types of the non-key columns. If possible, limit to those non-key columns that are numeric types or short, fixed-length strings rather than large, variable-length strings for the sake of saving storage space. Dates and timestamps in secondary indexes could cause problems as well because of their implicit conversions.

In addition to looking at what columns are indexed, we can look at indexes based on their characteristics as well. Those index characteristics are closely related to the performance and scalability of an Oracle-based application. They are summarized as follows:

- *Unique versus Non-Unique Indexes.* An index can be unique or non-unique. A unique index identifies only one row of a table without allowing duplicates, whereas a non-unique index does not have this constraint. Keep in mind that an index doesn't have to be unique all the time. Whether an index should be unique or not should be determined by the context of the application in terms of what the indexed columns represent. By making an index a unique index duplicates of rows are not allowed, which may go against the requirements of an application. For example, when an index is created on the last name of a customer, duplicate last names should be allowed. Such a non-unique index can help speed up the queries of searching customers by last name significantly if the customer base is large.
- *Sorted versus Unsorted Indexes.* The distinction between a sorted and an unsorted index is that with a sorted index, the indexed entries are sorted, while with an unsorted index, the indexed entries are unsorted. For example, PK indexes are sorted indexes by nature. Searching using sorted indexes result in *index rang scans*, whereas searching on unsorted indexes result in *full index table scans*. We'll discuss more about such index accessing methods later.

- *Dense versus Sparse Indexes.* Dense indexes have one or almost one data entry for every search key represented by an index, whereas sparse indexes have a block of data entries for each index. Dense indexes are good for equality queries or point queries specified with the equality sign of “=,” whereas sparse indexes are good for range queries specified with “>,” “>=,” “<,” or “<=.”
- *One-Dimensional versus Multi-Dimensional Indexes.* A one-dimensional index has a linear order imposed on its index entries, whereas a multi-dimensional index does not have a linear order imposed on its index entries. A PK index with one column as the PK is a one-dimensional index. An index on multiple FKs or a composite index is a multi-dimensional index.

You might be interested in knowing exactly what actual index types an Oracle database has. The best way to make sure about this is to query a database itself rather than gathering such info from various sources including formal Oracle documentations. The following query executed against one of my Oracle 11g databases resulted in the output following the query:

```
SQL> select distinct index_type from dba_indexes;
```

```
INDEX_TYPE
-----
IOT - TOP
LOB
FUNCTION-BASED NORMAL
FUNCTION-BASED DOMAIN
BITMAP
NORMAL
CLUSTER
DOMAIN
```

```
8 rows selected.
```

Note that wherever you see **NORMAL**, it implies a b-tree index. We’ll discuss b-tree indexes along with function-based indexes, and bitmap indexes in depth later. The index types of **LOB** and **DOMAIN** will be briefly discussed in the next section. The **IOT** index type will be deferred to a later section when discussed in the same context as for covering indexes. In the remainder of this section, let’s discuss the cluster index.

With the cluster index type, you can arrange to have more tables stored on the same area instead of separate areas on disk. The intention for doing so is to save space while minimizing the number of IOs to read off and write to disk.

How is it possible to store different tables into the same area on disk? Let’s use the **COUNTRIES** and **REGIONS** tables of the **HR** sample schema to explain this. As shown in Figure 3.7 in the previous chapter, those two tables share the same column of **region_id**. If we create a cluster with those two tables, the column values for each

row of the cluster would be stored in the order of `country_id`, `country_name`, `region_id`, `region_name`, with `region_id` playing the role of an index. This index is called the cluster index of the cluster. As you see, when we say *cluster*, it means both the cluster index and the clustered tables joined together by the cluster index. Obviously, a cluster is mainly designed for joins that join multiple tables with the cluster key appearing in the join conditions.

So far, the concepts about columns, indexes, and index types apply to structured relational data only. Oracle also allows data types for storing unstructured data. In reality, very often a table has mixed columns of data types for both structured and unstructured data. We discuss indexes for unstructured data next.

4.5 DOMAIN AND LOB INDEX TYPES FOR UNSTRUCTURED DATA

In this section, we briefly discuss index types and related data types for dealing with unstructured data in Oracle. After understanding the purposes of having those data and index types in Oracle, you can look up more about them if you feel they might be very relevant to you based on the application domain or the problems your application solves.

First, let's explain what the *domain index type* is for. Without going into the mechanics of how to create a domain index, it's sufficient to state that a domain index is for helping speed up application domain specific queries—mostly searching on unstructured data. Let's say your application needs to scan a candidate's resume, which is a text file, to see if the candidate has the required job skills that can be searched by giving keywords like "Oracle Database," "UNIX," "Windows," "JAVA," and so on. Then you can consider creating a domain index for your application, which will enable searing an unstructured text file with input keywords. In this sense, a domain index does not fall into the same perimeter as all regular index types that all work on structured relational data.

The LOB index type is another index type that works on unstructured data. The acronym LOB stands for Large Objects such as large texts, graphic images, still video clips, full motion video, sound waveforms, and so on. Compared with relational table records, which typically are a few hundred bytes per record, those large objects can be thousands or tens of thousands of times larger.

Obviously, LOBs have brought up many issues that have never been met before with small structured relational data. The first question is how LOBs would be stored. This depends on the types of LOBs, which can be one of the following:

- *BLOB (Binary Large Object)*. Used to store binary data such as image files.
- *CLOB (Character Large Object)*. Used to store textual data including XML files that are compliant to the native database character set encoding.
- *NCLOB (National Character Large Object)*. Similar to CLOB except that the textual data is coded in Unicode character set compliant to the national character set.
- *BFILEs*. Stored externally as operating system files that can be accessed from the database.

Note that in favor of LOBs Oracle 9i deprecated the LONG and LONG RAW types. In addition, the limit to those LOB data types was 4 GB in 9i, but had been lifted to 1 TB in 10g, and 128 TB in 11g. The size of a BFILE is limited only by the underlying operating system.

As far as LOB storage is concerned, one has to make a distinction between a LOB locator and a LOB value. LOB locators are pointers to LOB values, and LOB values are the actual object contents of LOBs, be it a BFILE, or a {N, B, NC} LOB. All LOB locators are always stored in the row, just as non-BLOB columns usually are. The BFILE values are stored outside the database just like regular files except that the corresponding locators stored in the row make them accessible to the database. LOB values for BLOBs, CLOBs, and NCLOBs are stored in the database either in-line in the table or out-line in a separate segment or tablespace. LOB value in-line or out-line storage is determined based on the following criteria:

In-Line LOB Value Storage Conditions:

- When the LOB value is NULL, regardless of the LOB storage parameters for the column.
- When the size of the LOB is small, approximately 4 KB or less, whether you specify `ENABLE STORAGE IN ROW` or not when the table was created.

Out-Line LOB Value Storage Conditions:

- When the size of the LOB already stored in the given row grows to approximately 4 KB or larger, regardless of the LOB storage parameters for the column.
- When you explicitly specified `DISABLE STORAGE IN ROW` when the table was created.
- When the size of the LOB already stored out-line shrinks to approximately 4 KB or smaller, Oracle does not move it to in-line, namely, the LOB still stays out-line.

We'll not go into the detailed mechanics of how to create a table with LOB data types. We are mostly interested in the performance and scalability ramifications with LOBs stored in-line or out-line. Whether you should use in-line or out-line storage for your LOBs depends on the average size of your LOBs. In-line storage leads to better database performance and scalability if LOB values stored in your database are small in size. Note that in-line LOB values will be moved to out-line when the 4 KB threshold is crossed, but never the other way. Therefore, if you have some of your LOB values varying dynamically, you may want to take the preceding statement into account when deciding on in-line or out-line for your LOBs.

Oracle recommends that best performance for LOBs can be achieved by specifying storage for LOBs in a separate tablespace rather than the same tablespace used for the table. Oracle also recommends specifying a separate tablespace for each LOB column in order to reduce device contention. These statements are true if your database storage is based on individual disks. It may not matter much if you are using a RAID configuration with which data is spread across multiple disks.

LOBs can be indexed just like all regular data type columns, and indexes that include LOB columns are called LOB indexes. Since LOBs are designed for storing unstructured data, you might need to create non-regular indexes such as domain indexes that are specifically attuned to your application, or text indexes to speed up text-based queries over the CLOB columns if the LOBs are text files. However, one cannot create function-based indexes (FBIs) on LOB columns directly. Practically, you may rarely need to create FBIs on LOBs.

You can query the `user_lobs` table to find out all the details about the LOBs in your database. For your reference, all the columns of the `user_lobs` table queried on an Oracle 11g setup are listed below. Those columns that are particularly interesting have been highlighted in boldface.

```
SQL> desc user_lobs
Name                               Null?      Type
-----
TABLE_NAME                       VARCHAR2 (30)
COLUMN_NAME                     VARCHAR2 (400)
SEGMENT_NAME                   VARCHAR2 (30)
TABSPACE_NAME                 VARCHAR2 (30)
INDEX_NAME                    VARCHAR2 (30)
CHUNK                             NUMBER
PCTVERSION                       NUMBER
RETENTION                       NUMBER
FREEPOOLS                       NUMBER
CACHE                           VARCHAR2 (10)
LOGGING                         VARCHAR2 (7)
ENCRYPT                          VARCHAR2 (4)
COMPRESSION                     VARCHAR2 (6)
DEDUPLICATION                   VARCHAR2 (15)
IN_ROW                         VARCHAR2 (3)
FORMAT                         VARCHAR2 (15)
PARTITIONED                     VARCHAR2 (3)
SECUREFILE                     VARCHAR2 (3)

SQL>
```

Notice that there is a column named `SECUREFILE` from the above list. In fact, Oracle 11g has introduced a new feature named SecureFiles marketed as the next generation unstructured data management. According to Oracle's own testimony, the advent of this new feature is a response to the great performance disparity between OS-managed files and Oracle managed LOBs, as can be deciphered from its statement that "SecureFiles offers the best-of-both-worlds architecture from both the database and file system worlds for storing unstructured data." The good news is that SecureFiles is 100% backward compatible with LOBs. Further exploration into this new feature is beyond the scope of this text, but it is interesting to see where

Oracle is moving toward in supporting processing unstructured data in its database product.

There exist other classification schemes on indexes, but we leave further discussion to a later section when we examine more indexing schemes. In the next section, we'll review the concepts of views, materialized views, and synonyms, which are commonly encountered database concepts.

4.6 VIEWS, MATERIALIZED VIEWS, AND SYNONYMS

As discussed in the preceding section, Oracle database operates on tables. Tables are logical entities with their contents stored on disk. Applications issue queries against those tables. Queries with common patterns against a table or more tables can be built into a database as *views* so that users or applications can issue queries against the views rather than the tables directly. The tables that a view is built from are called *base tables*. Views do not have their data stored in the database directly, but rather act more like virtual tables. Also one can build new views based on existing views or a mixed views and tables.

What are the advantages of a view? Obviously, a view can be used to impose security by restricting access to a predetermined set of rows or columns. Besides, a view can be used to simplify SQL queries for the user just as if the similar SQL statements were moved from users' code to the inside of a database. Views save users from spending time figuring out which base tables to query against, and how to join the data together, and so on.

There is also the concept of a materialized view. In contrast to an ordinary view, a materialized view has its query results stored in a separate schema object. Apparently, materialized views can help improve performance for rarely changing, static data such as in the situations like data warehouses.

The other relevant concept is a synonym, which is essentially an alias for a table, a view, a materialized view, a sequence, a procedure, a function or a package. A synonym does not improve performance, but it's a very useful concept. For example, one can use a synonym to mask the real name and owner of a schema object or one can provide global access to a schema object by creating the synonym as a public synonym owned by the user group PUBLIC to which every user has access to.

In the next section, we'll discuss stored procedures and triggers that are often used to implement business logic at the database level.

4.7 STORED PROCEDURES, FUNCTIONS, AND TRIGGERS

A stored procedure is a piece of code that is written in PL/SQL. It is compiled and stored inside an Oracle database, and thus the name *stored* procedure. A stored procedure can be called inside another PL/SQL stored procedure, stored function, or trigger.

A stored procedure has the following structure:

```
CREATE PROCEDURE procedure_spec IS procedure_body
```

For example, the Oracle sample Schema HR contains a stored procedure, which was created as follows:

```
CREATE PROCEDURE add_job_history
(p_emp_id          job_history.employee_id%type
, p_start_date     job_history.start_date%type
, p_end_date       job_history.end_date%type
, p_job_id         job_history.job_id%type
, p_department_id  job_history.department_id%type
)
IS
BEGIN
INSERT INTO job_history (employee_id, start_date, end_date,
                        job_id, department_id)
VALUES (p_emp_id, p_start_date, p_end_date,
        p_job_id, p_department_id);
END add_job_history;
```

In the above example, the procedure name is `add_job_history` with the arguments in the parenthesis (...) following the procedure name. The procedure body is the part from `BEGIN` to `END add_job_history;`. A stored procedure can be executed in a variety of ways, for example:

- From SQL*Plus with `EXECUTE add_job_history` (actual parameter list)
- It can also be called within another stored procedure similarly
- It can be called by a trigger

A stored function is created similarly to a stored procedure except that: (1) it returns a value, and (2) it can be called directly without using the `EXECUTE` command explicitly.

A database trigger is a PL/SQL stored procedure associated with a table. A trigger implements the tasks to perform when a certain event occurs, or we may say a trigger is fired by the occurrence of an event. Thus, one can use triggers to instruct an Oracle server what to do as a reaction to the occurrences of the triggering events.

Although a trigger also is a stored procedure, it cannot be called within another stored procedure. Besides, a trigger does not have a parameter list. The complete syntax for a trigger is shown as follows, with the parts included in a pair of square parenthesis [...] optional:

```
CREATE [OR REPLACE] TRIGGER <trigger_name> fire_time trigger_event
ON <table_name>
[REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]]
[FOR EACH ROW [WHEN (<trigger_restriction>)]]
```

```
--trigger body
[DECLARE
    declarations]
BEGIN
    statements
[EXCEPTION
    WHEN exception_name
    THEN ...]
END trigger_name;
```

Now let's look at each part of the trigger syntax one at a time:

- **CREATE [OR REPLACE] ...** The existence of the part "OR REPLACE" is because triggers cannot be modified. So the only way to modify a trigger is to replace it.
- **Fire_time.** This part specifies when a trigger will be fired. It could be **BEFORE** or **AFTER**, which means before or after a trigger event or statement is executed. A trigger using **BEFORE** or **AFTER** keyword is called a **BEFORE** trigger or an **AFTER** trigger, respectively. The difference between the two is whether the affected row is both checked against the constraints and locked. Apparently, a **BEFORE** trigger is lightweight as it doesn't check the constraints and lock the affected row, whereas an **AFTER** trigger is heavyweight from the performance and scalability perspectives.
- **Trigger_event.** A trigger event could be any one of the three actions: **INSERT**, **DELETE**, or **UPDATE**. This essentially explains why a trigger is needed.
- **ON table_name.** This part specifies which table this trigger is for.
- **[REFERENCING ...]** This part specifies the old row name and new row name for an **UPDATE** statement so that both new and old values of the affected columns can be accessed.
- **[FOR EACH ROW [WHEN Trigger_restriction]]** This part specifies that this is a row trigger with the presence of **FOR EACH ROW** or a statement trigger with the absence of **FOR EACH ROW**, in addition to specifying additional conditions for a trigger to fire with **[WHEN (<trigger_condition>)]**.
- **[DECLARE ...] END <trigger_name>;** This is the trigger body or a regular PL/SQL block except that some rules apply; for example, one cannot trigger another trigger, thus eliminating the possibility of creating an infinite triggering loop.

With the Oracle HR sample schema, there are two triggers, which were created as follows:

```
/* trigger update_job_history */
CREATE TRIGGER update_job_history
    AFTER UPDATE OF DEPARTMENT_ID, JOB_ID
    ON HR.EMPLOYEES
```

```

REFERENCING NEW AS new OLD AS old
FOR EACH ROW

BEGIN
    add_job_history(:old.employee_id, :old.hire_date,
        sysdate, :old.job_id, :old.department_id);
END update_job_history;

/* trigger secure_employees */
CREATE TRIGGER secure_employees
    BEFORE INSERT OR DELETE OR UPDATE
    ON HR.EMPLOYEES

BEGIN
    secure_dml;
END secure_employees;

```

Pay attention to how **BEFORE** and **AFTER** are used in the above two triggers. However, our real intention here is not to teach how to code triggers, but rather to point out that one should not abuse the use of triggers, which may otherwise lead to additional strain on an Oracle server and thus disastrous performance and scalability problems. As is seen, triggers can go off as often as every time a row of a database table is changed. A better defense is to use triggers cautiously, for example, avoiding the use of triggers if the same tasks can be performed with the constraints instead. Also keep in mind that firing up too many triggers in a cascaded fashion for a transaction can exacerbate the performance and scalability problems.

In the next section, we'll discuss an important subject of a database regarding how referential integrity can be preserved with PKs and FKs.

4.8 REFERENTIAL INTEGRITY WITH FOREIGN KEYS

Referential integrity may have performance and scalability implications, as checking referential integrity incurs additional work. That's why I'd like to add a brief overview about referential integrity here.

Database tables in a relational database management system (RDBMS) are relational. This means that certain relationships among the participating tables must be maintained all the time whenever operations like modifying (updating or deleting) data in the referenced tables are performed. However, having relationships doesn't always mean that there must be a need for referential integrity—there is no need to enforce referential integrity if the tables are static in nature. Therefore, referential integrity is more pertinent if data is changing constantly and before a change is made, it must be validated first. For example, with the Order Entry sample schema shown in Figure 4.5, what happens if a customer must be deleted? As you see, the

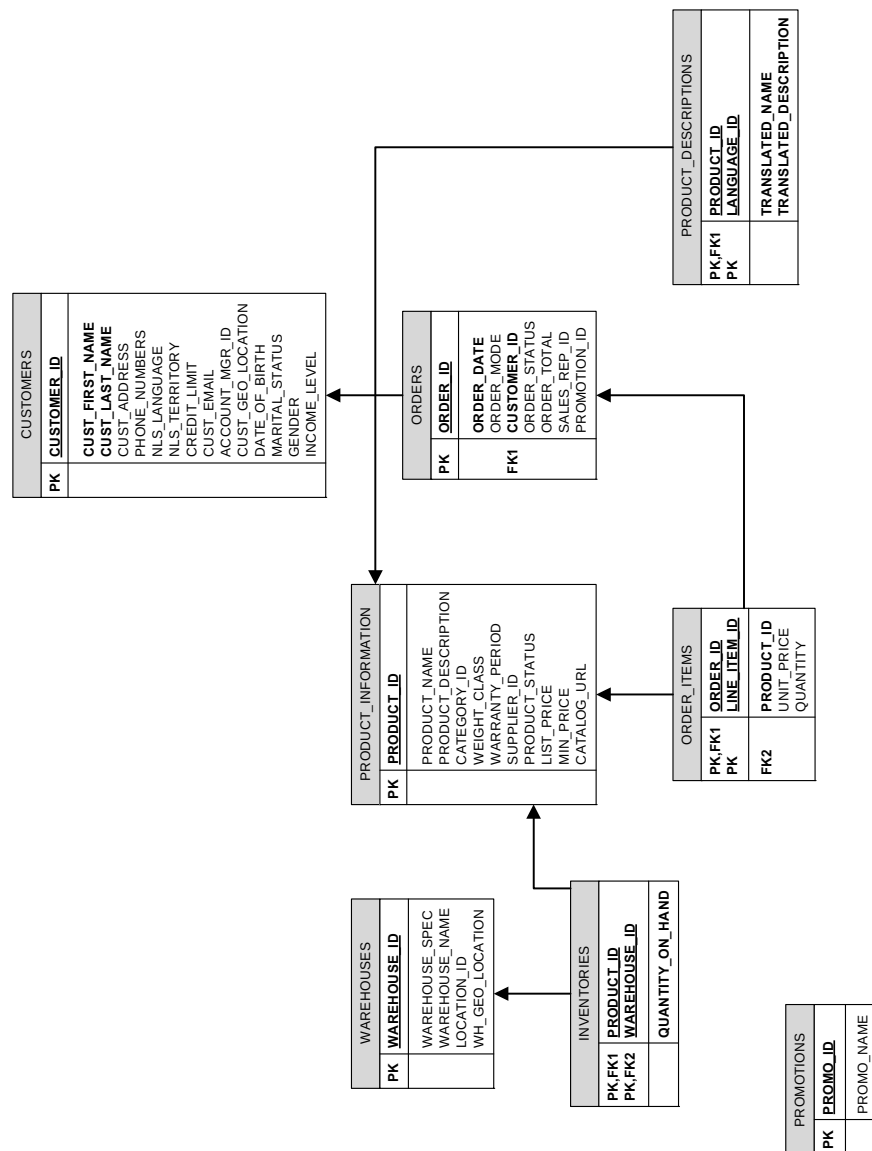


Figure 4.5 Oracle 11g Order Entry schema entity-relation diagram (ERD) generated with Visio via ODBC.

CUSTOMERS table has dependent or referencing table ORDERS. If a customer is deleted, the orders of that customer would end up being orphans.

Having explained when and why referential integrity is needed, the next question is how referential integrity can be implemented and enforced? Seeing how parent and child tables are associated with foreign keys, we already have the answer: FKs help enforce referential integrity.

4.9 SUMMARY

In this chapter, a quick tour has been given to help illustrate some of the major elements of an Oracle Server in the context of performance and scalability. A virtue of this tour is that the concepts are introduced by referencing a working Oracle setup so that the reader can be assured of the concrete values of those concepts.

In summary, we have introduced the following elements of an Oracle server according to how data is organized and stored in Oracle:

- Users and schemas
- Tables, indexes, domains, and LOBs for modeling structured and unstructured data
- Storage structure laid out in the hierarchy of tablespaces, segments, extents, and data blocks
- Views, materialized views, and synonyms
- Store procedures, functions, and triggers
- Referential integrity enforced with foreign keys.

A good understanding of all those basic concepts is essential for coping with Oracle performance and scalability challenges. If you are interested in learning more about each of the major elements introduced in this chapter, refer to the resources listed below.

Hopefully this chapter has given you a good, brief introduction to what an Oracle database server is about. Part Two, “Oracle Architecture from Performance and Scalability Perspectives,” which follows next, will give you a more systematic overview of the Oracle database server architecture from performance and scalability perspectives. We’ll cover up to 11g, which is the latest version of the Oracle relational database management system (RDBMS).

RECOMMENDED READING

The most authoritative texts about Oracle are those documents released with each version of Oracle. They are not only accurate but also up to date. You can consult Appendix A for a list of Oracle documents released with Oracle 11g.

Specific to this chapter, the following administrator's guide has two parts that are worthwhile to review (Part II, "Oracle Database Structure and Storage," and Part III, "Schema Objects"):

Oracle Corp., *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), March 2008 (http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.)

EXERCISES

- 4.1 Conceptually what's the difference between a user and a schema? Explain why they are closely related to each other.
- 4.2 Explain why the data block size may affect the performance and scalability of an Oracle-based enterprise application.
- 4.3 Assuming that a transaction needs to write 128 kB to disk and the data block size is 8 kB, calculate how many writes are needed for such a transaction.
- 4.4 Which type of table would be more favorable for performance and scalability, a heap-organized table or index-organized table?
- 4.5 What are the performance and scalability advantages of a clustered table over the non-clustered tables?
- 4.6 How could a secondary index affect performance and scalability adversely?
- 4.7 Is a view a physical or logical entity? What about a materialized view? What are the benefits of using a view and a materialized view?
- 4.8 Is a synonym a performance and scalability feature?
- 4.9 What are the pros and cons of using stored procedures versus coding the business logic in the application in the context of performance and scalability?
- 4.10 Explain why triggers may hurt performance and scalability if not used properly.
- 4.11 Is referential integrity always necessary? How can one minimize the impact of referential integrity on performance and scalability?

Part Two

Oracle Architecture from Performance and Scalability Perspectives

Only in quiet waters things mirror themselves undistorted. Only in a quiet mind is adequate perception of the world.

—HANS MARGOLIUS, quoted in *A Toolbox for Humanity*

Without delving into the details of how to administrate an Oracle server, in this part, we focus on exploring the performance and scalability features designed and built into Oracle from release to release.

Although Oracle tends to build more and more self-tuning features with each release, a good understanding of all essential Oracle performance and scalability features still is necessary, as it's very unlikely that one can just take a hands-off approach and let Oracle tune it by itself in the hope that it would perform and scale by itself. Based on my experiences of over a decade dealing with Oracle performance and scalability issues in both customer and internal tuning and testing environments, I have observed that Oracle performance and scalability issue scenarios and the underlying factors are countless. Both theoretically and practically, it's impossible for those auto-tune features to take every permutation of all those scenarios and offending factors into account.

The objective of this part is to present a self-consistent, coherent, and accurate view of all major Oracle performance and scalability features to the readers of college

students and software product stakeholders (developers, performance engineers, and managers). To achieve this objective, this part is organized with the following chapters:

- Chapter 5, “Understanding Oracle Architecture,” gives a high-level overview of Oracle architecture from the performance and scalability perspectives.
- Chapter 6, “Oracle 10g Memory Management,” introduces Oracle memory management schemes using Oracle 10g. It serves as a baseline for evaluating Oracle architecture from the memory perspective for latest and near future releases.
- Chapter 7, “Oracle 11g Memory Management,” introduces new memory management features in 11g on top of those available in 10g.
- Chapter 8, “Oracle Storage Structure,” explores the areas from I/O perspectives that are critical in supporting and determining the overall Oracle performance and scalability. Oracle I/O is one of the most important parts of the overall Oracle performance and scalability tuning parameter set.
- Chapter 9, “Oracle Wait Interface (OWI),” reveals all the ins and outs of the OWI features by explaining how OWI works and how one can make the most of it in real world Oracle performance and scalability tuning efforts. Oracle has opened itself up for looking into various performance and scalability issues by providing such a powerful framework of the OWI. This is one of the features that Oracle holds a strong lead over its competitors.
- Chapter 10, “Oracle Data Consistency and Concurrency,” discusses the data consistency and various isolation levels both in general and in Oracle’s context. Some important concepts such as Oracle locks, latches, enqueues, and so on, are explained. Oracle’s strengths in maximizing data concurrency through its efficient locking implementations are emphasized. A JDBC example is provided to demonstrate how to handle transactional aspects of developing an Oracle-based application at the application layer.
- Chapter 11, “Anatomy of an Oracle Automatic Workload Repository (AWR) Report,” walks you through all major parts of an AWR report taken from a real product. AWR is not only an indispensable performance and scalability diagnostic tool but also a very useful tool for studying Oracle performance and scalability characteristics in an academic setting.
- Chapter 12, “Oracle Advanced Features and Options,” presents a historical view of all the major features built into Oracle from 8i, 9i, to 10g and 11g. This is an interesting subject both academically and practically. Evolution of a product from generation to generation is inevitable, and it’s beneficial for all practitioners to embrace the newer features of a product in order to be able to work more effectively and efficiently.
- Chapter 13, “Top 10 Oracle Performance and Scalability Features,” summarizes the most important Oracle performance and scalability features from my viewpoint. Practitioners can check whether they have taken advantage of the

full set of Oracle performance and scalability features in developing their products. College students can get an update on the newest technologies that Oracle has to offer in the arena of database performance and scalability.

- Chapter 14, “Oracle-Based Application Performance and Scalability by Design,” is a self-contained chapter teaching how to build performance and scalability into a product based on Oracle from the ground up. The full life cycle of developing an Oracle database is illustrated with a sample enterprise application named SOBA (Secure Online Banking Application).
- Chapter 15, “Project: SOBA—A Secure Online-Banking Application,” illustrates how this sample application was developed using some of the most popular development frameworks such as Spring Source, Hibernate, and Restful Web services, and so on. This chapter alone can be used as a hands-on project for both college students and software developers.

Let’s start by looking at the Oracle overall architecture from performance and scalability perspectives in the next chapter.

5

Understanding Oracle Architecture

Architecture begins where engineering ends.
—Walter Gropius

The architecture of a software product basically is a design or blueprint that clearly depicts what parts it has, what functionality each part offers, and how those parts collaborate at a system level. The implementation of a software product is about the technologies and techniques used to build it against a given architectural blueprint. It's important to make the distinction between a design and an implementation when evaluating the performance and scalability of a software product, as the amount of efforts for fixing a performance and scalability issue may drastically differ, depending on whether it's a design issue or an implementation issue. When a change needs to be made to the architecture of a software product, it may affect other parts of the product as a whole from the functional point of view. When a change is made to the implementation of a software product, however, the effect would be localized and the functionality of the whole system is intact.

Much like the architecture of any other software product, the architecture of an Oracle Server is divided into two parts: specialized processes and specialized memory areas. The processes are responsible for performing various types of computing tasks, whereas the memory areas provide a necessary caching mechanism so that objects and data are closer to those processes than being accessed from

the far-end disk storage layer. Caching objects and data is purely driven by performance and scalability requirements.

Effective performance and scalability tunings with an Oracle database server depend on a good understanding of those specialized processes and memory areas, which are configurable through various externalized parameters. Oracle provides users with numerous views for looking into how those processes have been running, how each of the memory areas has been used, whether the system underperforms as the result of some initialization parameters set inappropriately, and so on. All those aspects pertinent to the latest releases of Oracle are covered in this chapter.

Specifically, this chapter covers:

- The version history of Oracle
- Oracle processes
- Oracle memory areas
- Dedicated versus shared Oracle server architecture
- Performance sensitive initialization parameters
- Oracle static data dictionary views
- Oracle dynamic performance (V\$) views

Next, let's start with a brief overview of the version history of Oracle to help put our coverage of Oracle architecture into perspective.

5.1 THE VERSION HISTORY OF ORACLE

Here is a count of the Oracle releases chronologically to help you gain insights into how Oracle has evolved into what it is today. More importantly, you are assured that you are not missing any major performance and scalability features built into each version of Oracle.

The major Oracle releases are as follows:

- 1977 – L. Ellison and his friends co-founded Software Development Laboratories (SDL).
- 1979 – Oracle V2 released, which implemented the basic SQL functionality of queries and joins with no support for transactions. Note V1 was never released. The company name was changed from SDL to RSL (Relational Software, Inc.).
- 1982 – The company changed its name to Oracle, which was the name of a project code-named Oracle funded by CIA (Center of Intelligence Agency) of the United States.
- 1983 – Oracle V3 rewritten in C and had COMMIT and ROLLBACK supported for transactions.
- 1984 – Oracle V4 released with supported read-consistency.

- 1985 – Oracle V5 released, which supported the client-server model. The rule-based optimizer (RBO) was introduced to speed up query execution to enhance overall Oracle performance.
- 1986 – Oracle 5.1 released, which supported distributed queries.
- 1988 – Oracle 6 released, which supported PL/SQL with Oracle Forms v3, row-level locking, and hot backups.
- 1992 – Oracle 7 released with support for referential integrity, stored procedures, and triggers. Note Oracle had no referential integrity support for about 15 years until this version.
- 1997 – Oracle 8 released with support for object-oriented development and multimedia applications. From performance and scalability perspectives, the effectiveness of the ratio-based performance tuning methodology was challenged with the introduction of the Oracle wait interface, which was rooted in mature queuing theory. Queuing theory is a standard theoretical framework for analyzing system performance issues.
- 1999 – Oracle 8i released with an Oracle JVM included to provide better support for developing Internet-based applications. With this feature, stored procedures can be coded in Java. And one can even run EJBs (Enterprise Java Beans) in Oracle. The letter *i* in 8*i* stands for *Internet*, emphasizing the computing paradigm targeted.
- 2001 – Oracle 9i released with over 400 new features including XML support and an option for Oracle RAC (Real Application Clusters) in place of the Oracle Parallel Server (OPS) option. The RAC doesn't only help HA (high availability) but also performance and scalability.
- 2003 – Oracle 10g released with *g* standing for *grid* computing. In this release, the RBO was formally phased out with the stabilized version of the cost-based optimization (CBO) to speed up query executions.
- 2007 – Oracle 11g released. Memory can be managed automatically from the top—the total memory specified for Oracle out of the total physical memory on a system—down to various sub-memory areas. Chapters 6 and 7 are dedicated to how Oracle manages memory.
- 2009 – Oracle 11g R2 released (most up-to-date release as of this writing). The first patch-set, Oracle 11.2.0.2 was released in September 2010. New features include improved data compression ratios (up to 20x), ability to upgrade database applications while users remain online, improved ease-of-use features that make grid computing more accessible, and automation of key systems management activities.

We won't be able to cover the performance and scalability features in every Oracle release even if there are any versions of Oracle older than 8i that are still running today. Instead, we focus on the versions of 10g and above throughout this text.

Next, we introduce various Oracle processes that are the most basic elements of an Oracle server for executing various database related computing tasks.

5.2 ORACLE PROCESSES

Without losing generality, we focus on the Oracle architecture on Windows. Note the major difference between Oracle on Windows and UNIX/Linux is that Oracle on Windows is based on a multi-threaded architecture, namely, one single process containing multiple threads for various components, whereas Oracle on UNIX/Linux is purely based on processes, namely, each component is implemented in its own process. We know that threads share memory, which minimizes context switches, while processes don't. Oracle on UNIX/Linux depends on the shared memory technique to close the performance gap between Windows and UNIX/Linux. Other than that major difference, the Windows and UNIX/Linux implementations of Oracle share the same architecture.

Figure 5.1 illustrates the Oracle architecture that consists of the *instance* in the upper part and *database* in the lower part. An Oracle instance is essentially composed of processes and memory areas, as we have made it clear previously. In terms of processes, there are three types of Oracle processes: user processes,

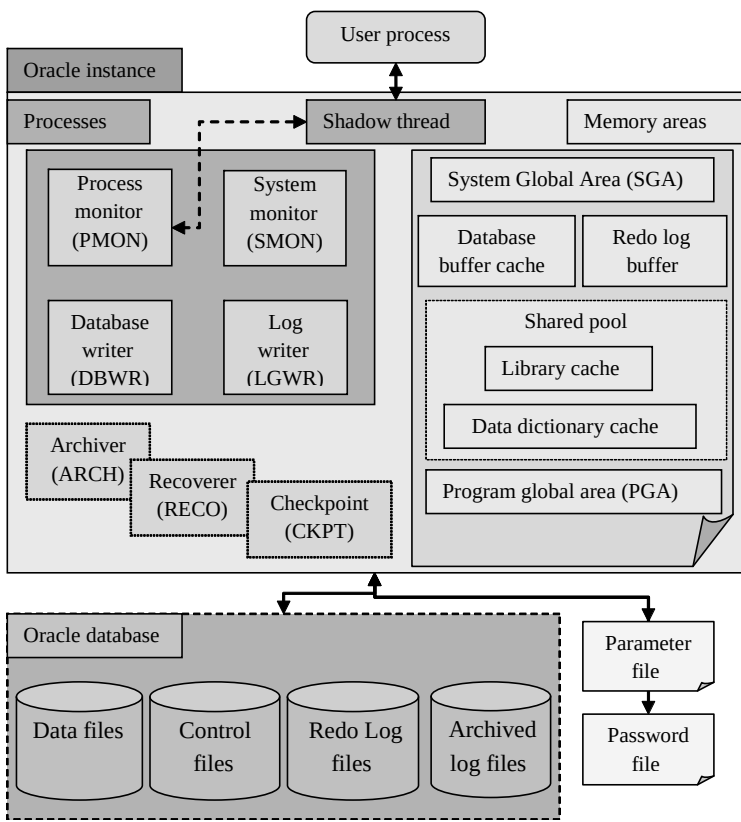


Figure 5.1 Oracle Architecture: the instance composed of processes and memory areas and database composed of various files.

shadow (or server) processes, and background processes. Each type of processes is explained as follows:

- *User Processes.* On the client side, a user process connects to the database and issues various requests on behalf of a user. The processes initiated with SQL*Plus are typical user processes. The GUI-based console and applications that access an Oracle server are user processes as well. Inside an Oracle server, each user connection is maintained as a session. One user may initiate multiple connections to the database, and therefore, there can be multiple sessions associated with a user process.
- *Shadow Processes.* A shadow process serves as the facade between a user and Oracle. It fulfills user's requests by working directly with the Oracle database. It can be either dedicated to a single user on a one server per user basis or it can be shared among multiple users in a multi-threaded server (MTS) configuration.
- *Background Processes.* These are the Oracle processes that perform server-side tasks to manage the database itself. The processes shown in Figure 5.1 are background processes, which constitute the majority of processes that we have to deal with from performance and scalability perspectives. Therefore, we focus on background processes for now.

As shown in Figure 5.1 and explained above, with Oracle on Windows, a user connects through a shadow thread to the Oracle instance. The shadow thread can interact directly with the process monitor, which in turn collaborates with the system monitor, database writer, and log writer, all of which are implemented as threads within the same process. The other three components, the Archiver, Recoverer, and Checkpoint, are implemented as three separate processes. Here we only discuss the major processes that are common for the last few versions of Oracle releases. From release to release, Oracle adds more processes to support new features.

Each component of the main server process has its own responsibilities as described below:

- *Process Monitor (PMON).* PMON is responsible for monitoring user connections. For example, if a user session is not ended gracefully either because of a network connection problem or a CTRL/C action from the user, PMON would notice that the user connection is broken. Then PMON would roll back the disconnected user session's transaction and release any of the session's resources to avoid inadvertently blocking other users from accessing the database normally. Other responsibilities of the PMON include: monitoring other server processes and restarting them if necessary, registering the instance with the listener dynamically, and restarting failed server processes and dispatcher processes.
- *System Monitor (SMON).* The SMON is responsible for many internal operations such as monitoring and defragmenting tablespaces, and so on. Its other responsibilities include crash recovery upon restart, coalescing free space,

recovering transactions active against unavailable files, shrinking rollback segments, and so on. It kicks in during the times of low activity or on-demand when certain operations are required immediately.

- *Database Writer (DBWn)*. A database writer is responsible for flushing or writing users' modified dirty data from database buffer cache to disks. Oracle in general does not write users' data directly and immediately onto the physical disks for performance reasons. Instead, it caches data in memory (database buffer cache) without writing to disks until being triggered by certain conditions such as (1) if a DBWR (Database Writer) sits idle for a few seconds, (2) if a new data block needs to be read into memory but no free space is available, or (3) when a checkpoint needs to be performed, and so on. Note that there could be multiple database writers working in parallel, which explains the letter *n* in the name of DBWn.
- *Log Writer (LGWR)*. A log writer is responsible for flushing the redo entries (both committed and uncommitted changes) in the redo log buffer to disks. Under the following circumstances, Oracle flushes the redo entries in the redo log buffer to disks before the data blocks are flushed to disks:
 - Every *n* seconds as configured
 - Whenever a transaction is being committed
 - When the redo log buffer exceeds a threshold percentage-wise or in absolute measurements as configured
 - Before the Database Writer writes when a checkpoint occurs
- *Archiver (ARCn)*. An Archiver is responsible for automatically backing up the transaction log files filled with redo entries in the database's archived transaction log. It is used only when the database is running in archive mode.
- *Recoverer (RECO)*. A Recoverer is responsible for recovering transactions that are left in a prepared state due to a crash or loss of connection during a two-phase commit. It is also responsible for recovering the database if a database failure occurs, for example, caused by a disk failure. It depends on the database backups and the archived log created by the Archiver to recover the database and all of the committed transactions.
- *Checkpoint (CKPT)*. This component is responsible for periodically initiating a process that a DBWR writes all modified data blocks in the database buffer cache back to the database's data files on disks. If a database server crashes, it can then be recovered from the checkpoints stored on disks. A checkpoint job involves such tasks as: (1) flushing the redo log buffers to the redo log files, (2) flushing the database log buffers to the data files, (3) writing a checkpoint record to the redo log file, and (4) updating the data file headers and control files, and so on.

Note that the process monitor, system monitor, database writer, and log writer are required, whereas the archiver, recoverer, and checkpoint are optional. The running background processes can be viewed and verified by issuing the following SQL query:

```
SQL> Select * from v$bgprocess where paddr <> '00';
```

Table 5.1 Output of the Query that Returned 12 Background Processes on an Oracle 10g Setup

Name	Description
PMON	Process Cleanup
PSP0	Process Spawner 0
MMAN	Memory Manager
DBW0	DB Writer Process 0
DBW1	DB Writer Process 1
LGWR	Redo etc.
CKPT	Checkpoint
SMON	System Monitor Process
RECO	Distributed Recovery
CJQ0	Job Queue Coordinator
MMON	Manageability Monitor Process
MMNL	Manageability Monitor Process 2

Table 5.1 lists the processes returned with the above query on an Oracle 10g setup. It is seen that there are two database writers (DBW0 and DBW1). There also are some processes that are not shown in Figure 5.1, for example, the Process Spawner (PSP0), Memory Manager (MMAN), Job Queue Coordinator (CJQ0) and Manageability Monitors (MMON and MMNL). These processes are discussed as follows:

- *Process Spawner (PSP0)*. This component is responsible for spawning Oracle processes whenever needed.
- *Memory Manager (MMAN)*. This component is responsible for managing memory as its name suggests. It uses the collected metrics to determine the desirable distribution of memory within Oracle. It constantly monitors the database and adjusts the memory allocations based on the workloads.
- *Job Queue Coordinator (CJQ0)*. This component is responsible for managing scheduled batch processes. It spawns job queue slaves (jnnn) to actually run the jobs.
- *Manageability Monitor (MMON)*. This component is responsible for performing manageability related tasks such as taking snapshots, raising alerts, and capturing statistics for SQL objects, and so on.
- *Manageability Monitor Lite (MMNL)*. This component is responsible for performing light-weight manageability-related tasks such as capturing session history and metrics, and so on.

For reference purposes, Table 5.2 lists the processes from an Oracle 11g R2 setup with the same query as shown above. All processes new in 11g are shown in boldface.

Another similar SQL is:

```
SQL>select spid, program, background FROM v$process;
```

Table 5.2 Output of the Query that Returned 19 Background Processes on an Oracle 11g R2 Setup

Name	Description
PMON	process cleanup
VKTM	Virtual Keeper of TiMe process
GEN0	generic0
DIAG	diagnosibility process
DBRM	DataBase Resource Manager
VKRM	Virtual sKeduler for Resource Manager
PSP0	process spawner 0
DIA0	diagnosibility process 0
MMAN	Memory Manager
DBW0	db writer process 0
LGWR	Redo etc.
CKPT	checkpoint
SMON	System Monitor Process
SMCO	Space Manager Process
RECO	distributed recovery
CJQ0	Job Queue Coordinator
QMNC	AQ Coordinator
MMON	Manageability Monitor Process
MMNL	Manageability Monitor Process 2

When this query was executed on the same Oracle 10g setup as for the previous query, Oracle returned all the same processes as listed in Table 5.1, with additional 56 processes labeled with (SHAD). These are shadow processes mentioned earlier in this section.

Note the prefix v\$ (pronounced “Vee-Dollar”) for the views of v\$bgprocess and v\$process in the above two queries. Such views are from the system statistics views (also known as the dynamic performance views) or simply v\$ views because of the v\$ prefix. These v\$ views contain information about the Oracle system, from the version of Oracle to resource utilizations, and so on. For example, Table 5.3 lists the version information, after the following query was executed against the Oracle system that yielded the processes listed above:

```
SQL>select * FROM v$version;
```

Table 5.3 Output of the Version Query on an Oracle 10g Setup

Product	Version
Oracle Database 10g Enterprise Edition Release	10.2.0.2.0 – Production
PL/SQL Release	10.2.0.2.0 – Production
Core	10.2.0.2.0 – Production
TNS for 32-bit Windows:	Version 10.2.0.2.0 – Production
NLSRTL	Version 10.2.0.2.0 – Production

Later, we'll provide a more systematical review about the Oracle system wide views including both static views and dynamic ψ views that are useful for troubleshooting Oracle performance and scalability issues. In the next section, we explore the Oracle memory areas that critically determine the performance and scalability of an Oracle-based application system.

5.3 ORACLE MEMORY AREAS

The Oracle memory areas such as SGA, Shared Pool and PGA depicted in Figure 5.1 help speed up data access, which is a large bulk of operations that a DBMS must perform. In one of my published papers (Liu, 2006), it was demonstrated that it took less than half a millisecond to fetch 32 blocks of data from the data buffer cache in memory, while fetching the same amount of data from disks would normally take 5 to 20 milliseconds. This 10 to 40 times performance disparity in accessing data between from memory and from disk well explains why caching data in memory is vigorously pursued not only in Oracle but also in other products. To some extent, it's not exaggerating to say that how well a server system performs and scales depends largely on how well various caches are used.

In this section, we focus on understanding how Oracle has various memory areas set internally, providing users with opportunities for tuning the performance and scalability of an Oracle-based application from the memory tuning perspective. Before delving into various Oracle memory areas, let's see how a cache implementation works in general. It's a well-known fact that it's always preferable to fetch data from memory than from remote disks. However, in reality, there is no guarantee that data is always available from memory, and there is no guarantee either that data can stay in memory forever for reuse. This brings up a few concepts such as a cache hit, a cache miss, and a cache load, which are vital for understanding caching performance.

Figure 5.2 explains the concepts of a cache hit, a cache miss and a cache load, with all the possible scenarios as follows:

1. *A Cache Hit.* The data requested by the CPU is in cache, which saves a direct disk access. A measure of how successful cache hits are is called *cache hit ratio* in percentages. Usually, the goal is to have upper 90s or close to 99% cache hit ratios. However, a high cache hit ratio does not guarantee the performance of an Oracle database, as will be explained later.
2. *A Cache Miss.* The data requested by the CPU is not in cache. This is the opposite of a cache hit. In this case, data is fetched from disk while having it loaded into the cache for future reuse.
3. *Data Aging Out to Disk.* This is a required operation when the cache is full and some space in the data buffer cache must be made for the newly loaded data. The cache implementation typically adopts a policy of aging out the least recently used (LRU) data blocks while keeping the most recently used

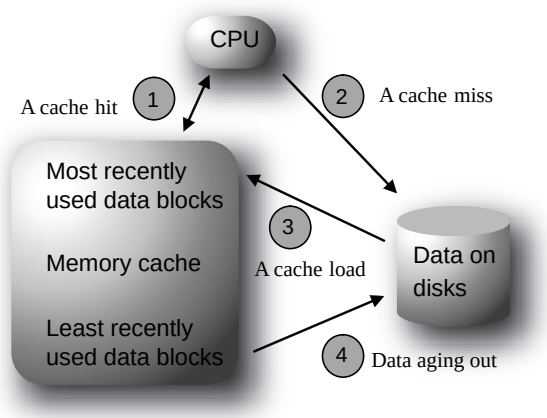


Figure 5.2 The concepts of a cache hit, a cache miss, and a cache load in Oracle's context.

(MRU) data blocks on the top. The MRU policy is used for accessing a data block in the cache, whereas the LRU policy is used for finding a data block to age out when needed. The newly loaded data block is placed at the MRU end of the cache.

Next, let's explore what objects are cached in the respective memory areas of an Oracle server. The first most important memory area is the System Global Area (SGA). An SGA is divided into two regions, one for the buffer cache, and the other for the shared pool. The buffer cache is for caching data from user tables, whereas the shared pool is for caching parsed and executed SQL statements, PL/SQL programs, and data dictionary information, which is the metadata about the database such as definitions of schema objects, user privileges and roles, integrity constraint information, and so on. Table 5.4 summarizes the objects stored in each region of an SGA. Needless to say, the data buffer cache is the most important and the largest area to tune, as it stores user data, which is the bulk of data that a DBMS deals with.

Now let's explore what a Program Global Area (or PGA) is for. For each connected user, Oracle creates a relatively small private memory area that holds a user's session information. This area is called PGA. Since a PGA is created on a per connected user basis, it's much smaller in size than an SGA.

Table 5.4 Objects Stored in an SGA

Region	Objects Stored
Buffer Cache	User data
Shared Pool	Library cache and dictionary cache
Library Cache	Processed SQL statements and PL/SQL programs
Dictionary Cache	Metadata on schema and security objects, etc.

In addition to SGA and PGA, there also is a concept of a `sort area`. A sort area is a small amount of server memory that a user session can use as a temporary work space to carry out sorting-related operations. The size of a sort area is adjustable with the corresponding externalized parameter.

We leave the details of how to size and tune an SGA, a PGA, and a sort area to a later chapter. But before leaving this section, let's mention that besides an SGA and a PGA, there is yet another global area called a UGA (user global area). The purpose of a UGA is to have a memory area for storing session-related information. Note that a UGA is not a separate area from an SGA or a PGA. It could reside in an SGA or a PGA depending on whether `DEDICATED` or `SHARED` server mode is used: it resides in a PGA with `DEDICATED` server mode or an SGA with `SHARED` server mode. `DEDICATED` versus `SHARED` server mode is an important but often confusing Oracle option. We'll shed some light on it in the next section.

5.4 DEDICATED VERSUS SHARED ORACLE SERVER ARCHITECTURE

As we explained earlier, Oracle database management tasks are taken care of by specialized background processes. The user requests are handled by two different types of shadow server processes, depending on which server architecture is chosen: `DEDICATED` or `SHARED`.

Before elaborating on dedicated versus shared Oracle architecture, let's first state the following:

- *Default Setting.* By default, an Oracle Server is set to run in `DEDICATED` mode. This `DEDICATED` mode is easier to set up and tune than the `SHARED` mode. If you want to run your Oracle Server in `SHARED` mode, you'll have to justify why you want to do so, as warned in some other texts like Kyte (2010). Also you'll have to consult the relevant Oracle documents pertinent to your versions of Oracle to learn how to post-configure an Oracle server to run in `SHARED` mode, if you decide to go along this path.
- *Which Mode is In Effect?* To determine whether your Oracle server has been configured to run in `DEDICATED` or `SHARED` mode, execute the following queries:

```
SQL> SELECT * FROM v$shared_server;
SQL> SELECT * FROM v$dispatcher;
```

Your Oracle server is not configured to run in `SHARED` mode if the results are “no rows selected.”

What's the major difference between the `DEDICATED` and `SHARED` architecture? As shown in Figure 5.3(a), in a dedicated server configuration, a dedicated server process is created for each of the connected users. That server processes the requests

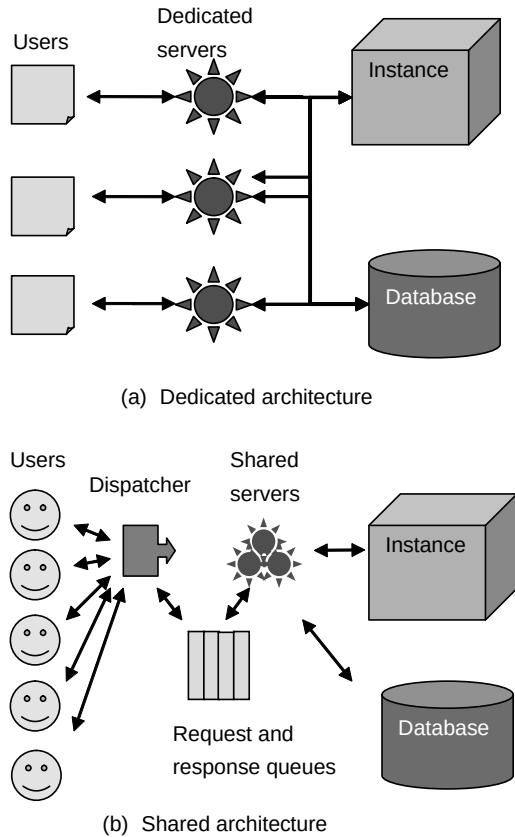


Figure 5.3 *Dedicated versus shared Oracle architecture.*

from and returns the results back to that user only. If there are n connected users, there will be n dedicated servers. The number of the dedicated servers varies dynamically as users connect and disconnect.

In a shared server configuration, however, the situation is a little bit more complicated than in a dedicated server configuration as shown in Figure 5.3(b). First, a user's requests are put into the request queues. The requests are then picked up by a shared server to execute. The responses for a user are put into the response queues, and then picked up and directed back to the user by the dispatcher. The number of shared servers varies as well based on the user load intensity. The main difference between the dedicated configuration and the shared configuration is that there will be n dedicated servers if there are n connected users with a dedicated server configuration, while with a shared server configuration, the number of shared servers (m) is much smaller than the number of connected users (n), or $m \ll n$.

Theoretically, the dedicated server configuration has been designed for long-running batch jobs whereas the shared server configuration has been designed for

OLTP type of applications. An OLTP application might be accessed interactively by hundreds or thousands or even tens of thousands of human users, which poses challenges for using the DEDICATED architecture. However, based on many years of real world use cases, the consensus is that in most cases including the case with many interactive users, a dedicated server configuration performs better and operates more reliably than a shared server configuration. Some complainants stated that the shared server configuration often uses a lot more memory and results in higher CPU usage compared with the dedicated server configuration with the same application and the same workload. Those observations seem to be against the rationales behind the dedicated versus shared Oracle server architecture. Nevertheless, the author is not in a position to agree or disagree with those empirical observations until getting a chance to conduct some rigorous tests in the future with extra care and precision.

Whether running in dedicated or shared mode, an Oracle server has many initialization parameters that predominantly determine its performance and scalability characteristics. Those parameters are designed for optimizing the performance and scalability of an Oracle server as much as possible with given hardware resources and application workload characteristics. Some of them are dynamically tunable, while some can only be changed with the Oracle server shutdown and restarted for the changes to take effect. Let's review some of those parameters in the next section.

5.5 PERFORMANCE SENSITIVE INITIALIZATION PARAMETERS

Oracle initialization parameters can be used to set limits or fixed values globally for the entire database or for users, processes and resources. Some parameters affect the performance and scalability of an Oracle-based application more than others. Those performance sensitive parameters can be fine-tuned for optimized performance with the same hardware and application workloads. The purpose of this section is to review such parameters so that one can become more knowledgeable in tuning such parameters to maximize the overall database and application performance.

Note that there is a default value for each parameter. The default value of an initialization parameter may vary depending on the operating system and available hardware resources.

There are three types of initialization parameters:

- *Derived Parameters.* The values of derived parameters are calculated based on the values of the underlying more basic parameters. For example, the parameter SESSIONS is derived from the parameter PROCESSES. And therefore, the default value of SESSIONS depends on that of PROCESSES. Note that you can override the default values of the derived parameters.
- *OS-Dependent Parameters.* The values of OS-dependent parameters vary from OS to OS. Such examples include the parameters DB_BLOCK_BUFFERS and

DB_BLOCK_SIZE, which specify how many data buffers to set aside and the size of those buffers, respectively.

- *Variable or Dynamic Parameters.* Those parameters are most performance sensitive. Their values can be varied dynamically while Oracle is running. Note that *more* does not necessarily mean *better* for performance. One example is that increasing the values of most parameters will increase the size of the SGA as well, and up to a certain point, a too large SGA will affect performance adversely.

To give a glimpse of those initialization parameters, Table 5.5 lists the top 20 initialization parameters performance-wise. The default values were set on a Windows XP Professional system with 2 Intel Core™ 2 Duo E8500 CPUs at 3.16 GHz each and 3.25 GB RAM. Those default values listed in Table 5.5 serve as a baseline reference that can be compared with settings on more advanced hardware.

Table 5.5 Top 20 Initialization Parameters Performance-wise with the Corresponding Default Values Set with an Oracle 11g Server Installed on a 2 CPU, 3.25 GB RAM Windows XP System

Category/Initialization Parameter	Default
General	
cursor_sharing	EXACT
db_file_multiblock_read_count	128
open_cursors	300
processes	150
session_cached_cursors	50
Memory	
db_cache_size	0
log_buffer	5653504
memory_max_target	820M
memory_target	820M
pga_aggregate_target	0
sga_max_size	512M
sga_target	0
shared_pool_reserved_size	11324620
shared_pool_size	0
sort_area_size	65536
Optimizer	
optimizer_dynamic_sampling	2
optimizer_mode	ALL_ROWS
statistics_level	TYPICAL
timed_statistics	TRUE
trace_enabled	TRUE
optimizer_mode	ALL_ROWS
statistics_level	TYPICAL
timed_statistics	TRUE
trace_enabled	TRUE

The initialization parameters are stored in a parameter file, which has both a binary version and a textual version. The binary file is called the server parameter file or SPFILE, which is located at `%ORACLE_HOME%/database` with the name of `SPFILE{SID}.ORA`. The textual parameter file (or PFILE) is named `init.ora`, which is located at `%ORACLE_HOME%/srvr/admin`.

Several basic rules on setting Oracle initialization parameters include:

- Case-sensitivity depends on the OS, which means `yes` on UNIX/Linux and `no` on Windows.
- A pound sign (£) starts a comment line. Only those lines that have no prefix of `#` are effective.
- A backslash (\) indicates continuation of the parameter specification.

You can change the value of a parameter in one of the following three ways:

- By editing the textual initialization parameter file directly using a text editor. Oracle may pick up some of the modified values without requiring restarting, but in most cases, the modified value takes effect only after Oracle is restarted.
- By issuing an `ALTER SYSTEM SET <parameter>=<value> COMMENT='your comments' SCOPE=<scope>;` statement to dynamically modify a parameter in the server parameter file while Oracle is running. Note that the parameter `<scope>` has three distinct values: `SPFILE`, `MEMORY`, and `BOTH`. The value of `SPFILE` indicates updating the SPFILE to take effect only after the next database restart, whereas the value of `MEMORY` indicates updating it for the current instance only without updating the SPFILE. The value of `BOTH` indicates changing it now both in memory and in SPFILE stored on disk.
- By using a console such as the OEMJC or the EM DBConsole.

A question is how we know if Oracle is using the PFILE or the SPFILE for its initialization parameters, since two files may not be synchronized with each other. Oracle prefers the SPFILE to the PFILE. Which one is used can be verified using the OEMJC or EM DBConsole. One can also query the `V$SPPARAMETER` view with the following command (note the first four commands are for formatting the output):

```
SQL>SET pagesize 0
SQL>SET linesize 500
SQL>SET colsep '|'
SQL> COLUMN name FORMAT 40
SQL>SELECT name || ',' || value FROM V$SPPARAMETER WHERE
value <> 'null';
```

If it returns a non-empty result set, then SPFILE is used; otherwise, PFILE is used.

One can easily create one type of initialization parameter file from the other with the following commands (but remember to back up the original one first):

```
SQL> CREATE PFILE FROM SPFILE;
SQL> CREATE SPFILE FROM PFILE;
```

The initialization parameter settings can be viewed in a few different ways. First, one can use the SQL*Plus command `SHOW PARAMETERS` to see all parameters or `SHOW PARAMETER <parameter_name>` to see only one parameter. Alternatively, one can query the V\$ views of `V$PARAMETER` and `V$PARAMETER2` for all the currently in-effect parameter values, and `V$SPPARAMETER` for the current contents of the SPFILE. The difference between `V$PARAMETER` and `V$PARAMETER2` is that the latter displays a list parameter value in multiple rows.

For example, to view parameters and their values, use the following commands:

```
SQL> SHOW PARAMETERS
SQL> SHOW PARAMETERS DB
```

The first command shows all parameters, whereas the second command shows only those parameters having DB in their names. This is a very useful filter.

Finally, note that some parameters are dynamical parameters, which means they can be modified for the duration of the scope specified while Oracle is running. The dynamic parameters are changed by using the command

```
SQL> ALTER SYSTEM SET parameter_name = value [DEFERRED];
SQL> ALTER SESSION SET parameter_name = value;
```

The first command above applies globally for the entire system whereas the second command applies to the session that invokes the statement only. The `DEFERRED` keyword modifies the value of a parameter for future sessions only. However, the recommended method is to make changes to dynamic parameters using a console, which will not only help you identify which parameters are dynamic ones but also help ensure the integrity.

Next, let's get familiar with the concept of Oracle static data dictionary views.

5.6 ORACLE STATIC DATA DICTIONARY VIEWS

First, let's clarify what the Oracle data dictionary is. The Oracle data dictionary contains information about the structures of the database as well as information about the database schema objects such as tables, columns, users, and data files, and so on. Such information is called *metadata*, which remains static throughout the life of a database, and thus the name of static data dictionary views for those views of querying such information. The other part of the data dictionary contains tables

for monitoring ongoing database activities. Since such activities describe the dynamic state of the database, which varies with time, the corresponding views are called dynamic performance views. The dynamic views are covered in the next section.

To list the data dictionary views available to you as a user in Oracle, query the view `Dictionary` with the command `“SELECT * FROM Dictionary WHERE ROWNUM < n;”` where specifying `rownum < n` is for limiting the # of rows returned. For example, on one of my Oracle 10g setups, `Dictionary` contains 1870 rows. Using the `“DESC Dictionary”` command, it shows that `Dictionary` has only two columns: `TABLE_NAME` and `COMMENTS`.

All static data dictionary views are classified with the following three prefixes:

- *ALL_ Views.* This class of views displays all the information such as schemas accessible to the currently logged-in user. For example, the `ALL_TABLES` view describes the relational tables accessible to the current user.
- *DBA_ Views.* This class of views displays all relevant information in the entire database intended for DBAs. The `DBA_TABLES` view describes all relational tables in the database.
- *USER_ Views.* This class of views displays all the information from the schema of the current user with no special privileges required. For example, the `USER_TABLES` view describes the relational tables owned by the current user.

In addition to `TABLES`, you can apply the above three prefixes to many other Oracle schema objects such as those introduced in Chapter 4, listed here in no particular order: `USERS`, `OBJECTS`, `TABLESPACES`, `SEGMENTS`, `EXTENTS`, `INDEXES`, `LOBS`, `JOBS`, `SEQUENCES`, `SYNONYMS`, `TRIGGERS`, `VIEWS`, and so on. You can consult the document listed at the end of this chapter for a complete list of static data dictionary views in addition to those illustrated above.

Next, let’s take a look at the Oracle `V$` dynamic performance views, which are more relevant to troubleshooting Oracle performance and scalability issues than those static data dictionary views. As we pointed out earlier, those dynamic views contain information about database activities over time.

5.7 ORACLE DYNAMIC PERFORMANCE (V\$) VIEWS

Oracle contains a set of built-in views under the built-in database administrator user `SYS`. These views are called dynamic performance views because their contents relate primarily to performance and change dynamically with time while a database is open and in use. They are also called `V$` views because they all have the common prefix of `V$`. The `V$` views are the performance information source as a basis for all Oracle database performance tuning tools.

The actual dynamic performance views are identified by the prefix V_\$ under the schema SYS. Public synonyms for those views have the prefix V\$. Users should access V\$ objects only instead of V_\$ objects. Note that only users with a SYSDBA role, for example, the built-in users SYS and SYSTEM, can access V\$ views. Some texts suggest that you may need to run the `catalog.sql` script to create those views. That is necessary only if you created your database manually from the command line. If you created your database with the Oracle installer, all those V\$ views had already been created automatically for you and there is no need to run any additional scripts.

The dynamic performance views provide metrics on Oracle memory structures, disk structures and many other performance oriented statistics. They are used by Oracle Enterprise Manager Java Console, Oracle Trace (outdated, primarily prior to 10g) and EM DBConsole (starting from 10g) as the primary interface for accessing information on system performance. One can use the following query to get a count of the V\$ views from an existing Oracle server:

```
SQL> SELECT count (*) FROM V$FIXED_TABLE;
```

To learn more about V\$ views, you can run the following query to get a complete list of all V\$ views in your Oracle system:

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

There are also views prefixed with GV\$. Those are global V\$ views for all Oracle instances in a clustered environment. Many of them are redundant with V\$ views and you can eliminate them by adding `where view_name like 'V$%'` to the above query. Table 5.6 provides a subset (90) of all Oracle V\$ views (484) from performance and scalability perspectives. They were obtained with an Oracle 11g server. For your convenience, those views have been classified by the categories of General, Event, File IO, Lock, Memory, PGA, Process, Session, SGA, Sort, SQL, and System. What each V\$ view is about is quite self-explanatory by its name following the V\$ sign.

The number of V\$ views has grown from 187 with Oracle 8.1.7 to 398 and 484 with Oracle 10g and 11g, respectively. That kind of explosive expansion rate in the number of V\$ views has made it more and more difficult to completely rely on the V\$ views for your Oracle specific performance and scalability optimization work. Since they are designed mainly for those auto-tune features to consume, it is recommended that you rely more on the Oracle EM DBConsole to find the performance and scalability symptoms with your Oracle server, rather than spending endless efforts doing the drill-down type of analysis using those V\$ views on your own. It's only occasionally beneficial to look into deeper into your Oracle performance and scalability issues by finding and examining those relevant V\$ views. For these reasons, we'll just wrap up this chapter here, and leave some more space for other more effective Oracle performance and scalability tuning methodologies in the remainder of this text.

Table 5.6 Some of the Oracle V\$ Views from Performance and Scalability Perspectives (11g)**1.1 General**

v\$database
 v\$db_cache_advice
 v\$db_object_cache
 v\$metric
 v\$resource
 v\$resource_limit
 v\$services
 v\$spparameter
 v\$statistics_level
 v\$type_size

1.2 System

v\$sys_optimizer_env
 v\$sys_time_model
 v\$sysmetric
 v\$sysstat
 v\$system_cursor_cache
 v\$system_event
 v\$system_wait_class

1.3 Process

v\$process
 v\$process_memory

1.4 File IO

v\$filestat
 v\$io_calibration_status
 v\$iostat_file
 v\$segment_statistics
 v\$segstat

1.5 Session

v\$ses_optimizer_env
 v\$sess_io
 v\$sess_time_model
 v\$session
 v\$session_connect_info
 v\$session_cursor_cache
 v\$session_event
 v\$session_longops
 v\$session_object_cache
 v\$session_wait
 v\$session_wait_class
 v\$session_wait_history

2.1 Memory

v\$open_cursor
 v\$java_library_cache_memory
 v\$library_cache_memory
 v\$llibrarycache

v\$memory_current_resize_ops
 v\$memory_dynamic_components
 v\$memory_resize_ops
 v\$memory_target_advice

2.2 PGA

v\$pga_target_advice
 v\$pga_target_advice_histogram
 v\$pgastat

2.3 SGA

v\$sga
 v\$sga_current_resize_ops
 v\$sga_dynamic_components
 v\$sga_dynamic_free_memory
 v\$sga_resize_ops
 v\$sga_target_advice
 v\$sgainfo
 v\$sgastat

2.4 Shared Pool

v\$shared_pool_advice
 v\$shared_pool_reserved

3.1 SQL

v\$sql
 v\$sql_bind_capture
 v\$sql_bind_data
 v\$sql_bind_metadata
 v\$sql_cursor
 v\$sql_hint
 v\$sql_optimizer_env
 v\$sql_plan
 v\$sql_plan_statistics
 v\$sql_plan_statistics_all
 v\$sqlarea
 v\$sqlstats
 v\$sqltext
 v\$sqltext_with_newlines

3.2 Event

v\$event_histogram
 v\$event_name
 v\$eventmetric
 v\$wait_chains
 v\$waitstat

3.3 Lock

v\$lock
 v\$lock_activity
 v\$locks_with_collisions

5.8 SUMMARY

In this chapter, we explored Oracle architecture from performance and scalability perspectives by looking at what processes and memory areas an Oracle server has internally. We also covered how one can probe the structure of an Oracle server with the help of those built-in static data dictionary views. We pointed out that one can track the evolving database activities with the help of those built-in V\$ dynamic performance views. Very often, database activities exhibit themselves as performance and scalability symptoms, for which those V\$ views can be used as a powerful diagnostics tool.

We also reviewed some of the Oracle initialization parameters in the context of performance and scalability. We specifically pointed out that those dynamic parameters are more critical to the performance and scalability of an Oracle-based enterprise application than the static ones. In general, there is an optimal range of values for each of those dynamic parameters. And if each dynamic parameter operates in such an optimal range, the overall system will perform optimally. That's the subject of tuning and sizing an Oracle-based enterprise application that we will cover later.

Before concluding this chapter, it's necessary to remind you of the controversial issue of a dedicated versus a shared Oracle server configuration. Theoretically, a dedicated configuration is for batch job type of applications, whereas a shared configuration is for OLTP type of applications. However, keep in mind that there is a possibility that a dedicated configuration is better than a shared configuration even with OLTP type of applications with a large number of users. It might be hard to cast a clear cut about it, but nothing will be more convincing than your own tests with your own application.

RECOMMENDED READING

Although there are many Oracle texts on the market, they become outdated quickly. For the most up-to-date text about Oracle architecture, refer to the following:

T. Kyte, *Expert Database Architecture*, A Press, New York, 2010.

You can also refer to the following Oracle product documents:

Oracle Corp, *Oracle Database Concepts*, 11g Release 1 (11.1) B28318-05 (556 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28318.pdf.

This document has four parts, with Part II very relevant to this chapter:

- Part I, What is Oracle?
- Part II, Oracle Database Architecture
- Part III, Oracle Database Features
- Part IV, Oracle Database Application Development.

Oracle Corp, *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.

Chapters 4 and 5 of this document describe Oracle processes and memory structure. It's worthwhile to review those two chapters for a more in-depth coverage of those two subjects.

Oracle Corp, *Oracle Database Reference*, 11g Release 1 (11.1) B28320-03 (1132 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28320.pdf.

This document covers all Initialization Parameters (Part I), all Static Data Dictionary Views (Part II), all Dynamic Performance Views (Part III), and Descriptions of all Wait Events (Appendix C). This is the place you need to go if you want to know more about a specific item, be it an initialization parameter, a static data dictionary view, a dynamic performance view, or a wait event.

For a benchmarking of fetching data from cache and disk, see:

H. H. Liu, *Applying queuing theory to optimizing enterprise software applications*, in CMG 2006 Proceedings, Reno.

EXERCISES

- 5.1 What's the origin for the name of *Oracle*?
- 5.2 What's the major difference between the architecture and the implementation of a software product?
- 5.3 Why is it important to identify performance and scalability issues from architectural perspectives as early as possible during the life cycle of a software product?
- 5.4 What versions of Oracle have you worked on? Give some examples of using certain Oracle performance and scalability features with your product.
- 5.5 What are the differences among three types of Oracle processes? What does the initialization parameter *process = 150* mean exactly?
- 5.6 What is an Oracle shadow process? How do you find out how many shadow processes you have running in your Oracle server?
- 5.7 What's the major difference between the Windows and UNIX versions of Oracle? Let's say your enterprise application is based on Oracle. What's your opinion on whether your enterprise application will exhibit similar performance and scalability characteristics on the two drastically different platforms? Can you concentrate on optimizing the performance and scalability of your application with Oracle on one OS platform with the assumption that your application will perform and scale similarly on the other platform as well—as long as the underlying hardware supporting each OS is comparable to each other?
- 5.8 What's the major difference between an SGA and a PGA? How do you find out how much memory is being used by your SAG and PGA, respectively?
- 5.9 Is a sort area a part of an SGA or a PGA? What is it for?
- 5.10 What's your take on a dedicated versus a shared Oracle server configuration? Let's say you have an OLTP application that needs to support a very large user

base. Will you go with a dedicated or shared Oracle server configuration? Or how will you go about it?

- 5.11** What are dynamic initialization parameters? If you need to change the value of a dynamic initialization parameter, how do you determine which method you will use to change it?
- 5.12** What's the major difference between static data dictionary views and dynamic V\$ performance views? What's the designed purpose for each type of view?
- 5.13** There is a notion that you should spend time understanding all dynamic V\$ performance views so that you will be equipped better for troubleshooting Oracle performance and scalability issues. Do you feel if this is a good idea given the fact that the dynamic performance views indeed give a very complete set of information about Oracle database activities?
- 5.14** If you are familiar with other database products like Microsoft SQL Server and DB2 as well, compare the architectural differences among them using the similar aspects outlined in this chapter.

6

Oracle 10g Memory Management

The artist does not see things as they are, but as he is.

—*Alfred Tonnelle*

Memory management is an important aspect for every software server product from performance and scalability perspectives. That is even more obvious with Oracle, as caching data in memory is one of the most frequently used strategies for optimizing Oracle performance and scalability.

In this chapter, we choose the version of Oracle 10g for illustrating the concepts associated with Oracle memory management. This version is a good choice here for two reasons. First of all, Oracle 10g represents the most comprehensive memory management schemes out of all versions of Oracle backward so far. Secondly, the gap in memory management between 10g and the latest version of 11g is not very wide. In Oracle 10g, both SGA and PGA can be automatically managed independently, while having the total physical memory allocated to Oracle managed manually. In Oracle 11g, Oracle has gone one step further that the total physical memory allocated to Oracle can be managed automatically. Therefore, seeing how both SGA and PGA can be separately, automatically managed is retrospectively educational in understanding how Oracle manages its two most important memory areas: SGA and PGA.

The objective of this chapter is to help you get a clear, unambiguous understanding of how Oracle manages memory allocated to it on a system. That is necessary for

anyone who is concerned with whether Oracle has been using its memory resource optimally and how one can size those memory areas properly to help Oracle achieve maximum efficiency in using its memory resource.

This chapter consists of the following main sections:

- SGA Sub-Areas
- SGA Sizing: Automatic Shared Memory Management (ASMM)
- PGA Sizing: PGA_AGGREGATE_TARGET

Next, let's explore how those memory areas are configured and managed in Oracle 10g.

6.1 SGA SUB-AREAS

An SGA in Oracle 10g is further divided into two categories of pools: the dynamic pools that can be managed by Oracle dynamically, and the static pools that are fixed and cannot be changed dynamically. The dynamic pools include:

- *A Database Block Buffer Cache.* This sub-area is defined with the parameter `DB_CACHE_SIZE`. It holds copies of user data blocks in memory so that they are closer to CPUs than those on disks. Inside Oracle, this is also designated as DEFAULT buffer cache. This perhaps is the most crucial SGA sub-area from which one can feel the joy or pain easily depending on whether it's properly sized. If it's too small, you will feel that your SQL queries will run very slow. If it's too big, there will be no room left for the PGA and the chances are that the Oracle Server even doesn't start up.
- *A Shared Pool.* This sub-area is for query-related objects such as shared cursors, stored procedures, dictionary caches, and so on, to speed up SQL query processing.
- *A Java Pool.* This sub-area stores Java objects inside an Oracle Server. This pool was introduced in Oracle 8i to support coding some parts of an Oracle application in Java and running them inside Oracle.
- *A Large Pool.* This sub-area stores larger than usual objects cached in a shared pool. A large pool is used in the situations where a large chunk of memory is needed, and when it's done, it doesn't have to be kept in the pool for reuse as the chances for reuse are rare. It's mostly used in such situations as: in a shared server configuration where there is no need to keep a session that has ended, in parallel servers for inter-process message buffers used to coordinate the parallel query servers, and in backup for RMAN disk IO buffers. This pool was introduced in Oracle 8.

The static pools include the following:

- *A Streams Pool.* This sub-area supports the concept of Streams introduced in Oracle 10g. Oracle Streams captures and stages database changes at a source

database, and then propagates and applies the changes to one or more destination databases. A destination database could just be the originating database itself, or another Oracle or non-Oracle database. So one can think that Oracle Streams helps keep databases synchronized with each other.

- *A Keep Pool.* This sub-area is for small objects such as fully scanned tables and indices that are small in size. This prevents small tables and indices from being fully scanned every time when such an operation is needed.
- *A Recycle Pool.* This sub-area is the opposite of a Keep Pool. It was introduced in Oracle 8i for storing large, transient data blocks resulting from fully scanned large tables. Such large objects are subject to immediate disposal as they are unlikely to be needed again.
- *Redo Log Buffers.* This sub-area holds transaction specific information to support transaction rollback if a failure occurs. The contents of redo log buffers are flushed to disks from time to time.
- *A Fixed_SGA Sub-Area.* This sub-area is determined at the Oracle installation time. This is named *fixed* as it cannot be changed by a user. It stores information on other SGA components. It is usually small in size.

Figure 6.1 shows the screenshot of an Oracle 10g R2 memory configuration taken with the OEMJC connected to an Oracle 10g R2 server (this server was equipped with a total amount of 16 GB RAM). It is seen that all dynamical pools were visible on the UI while the static pools were invisible. You can query both dynamic and static pools with the following SQL:

```
SQL> select component, current_size from v$sga_dynamic_
components order by current_size desc;
```

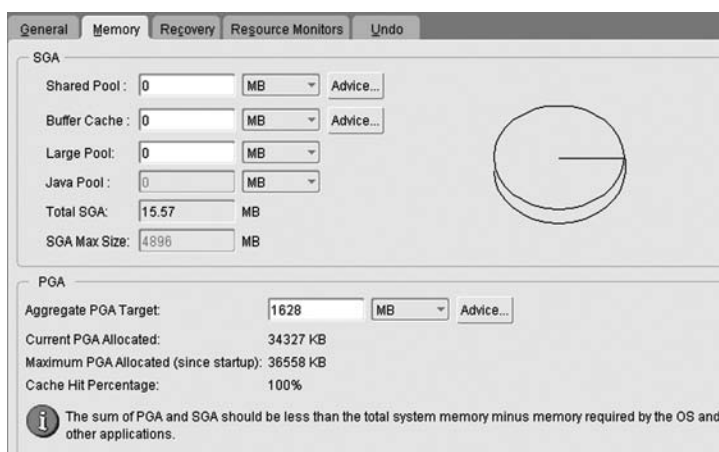


Figure 6.1 Oracle 10g memory management: SGA and PGA are managed separately with the option of having each managed automatically or manually.

Table 6.1 Buffer Pools of an Oracle 10g Server

COMPONENT	CURRENT_SIZE
DEFAULT buffer cache	2,617,245,696
shared pool	2,432,696,320
streams pool	33,554,432
large pool	16,777,216
java pool	16,777,216
DEFAULT 16K buffer cache	0
DEFAULT 32K buffer cache	0
DEFAULT 8K buffer cache	0
DEFAULT 4K buffer cache	0
DEFAULT 2K buffer cache	0
ASM Buffer Cache	0
KEEP buffer cache	0
RECYCLE buffer cache	0

The results are shown in Table 6.1. It is seen that the buffer cache and shared pool are 2.6 GB and 2.4 GB, respectively, whereas the streams pool, large pool, and java pool are 33.6 MB, 16.8 MB, and 16.8 MB, respectively. All other pools are zero in size.

If you are interested in knowing the details of a pool, you can execute the following commands as described in Kyte (2001):

```
SQL> compute sum of bytes on pool
SQL> break on pool skip 1
SQL> select pool, name, bytes from v$sgastat order by pool,
name;
```

The above query would also give the size information on the fixed_sga and log buffer sub-areas. The results are omitted here as they are too tedious.

After this dazzling list of various pools of an SGA, you might wonder what a daunting task it is to properly size those pools. The good news is that Oracle has made this task easy by providing an option of managing those buffer caches automatically with the option termed ASMM (Automatic Shared Memory Management). This is the recommended memory configuration, as is discussed next.

6.2 SGA SIZING: AUTOMATIC SHARED MEMORY MANAGEMENT (ASMM)

To use ASMM, all one has to do is to set the initialization parameter `SGA_TARGET` to a non-zero value and the `STATISTICS_LEVEL` initialization parameter to `TYPICAL` or `ALL`. The parameter `SGA_TARGET` specifies the amount of memory available for Oracle to manage automatically, whereas the `STATISTICS_LEVEL` initialization parameter specifies the granularity with which statistics are collected. Section 9.5 explains more about the other settings of the `STATISTICS_LEVEL` initialization parameter.

Edit Database : Configuration - system@ORA10GR2

All Parameters

You are not logged on with SYSDBA privilege.
Only dynamic parameters are editable.

☒ Running ☐ SPFile

Running parameters are the parameters currently in effect for this database instance.

Parameter Name	Value	Default	Dynamic	Category
sessions	170	✓		Processes and Sessions
sga_max_size	5133828096	✓		SGA Memory
sga_target	5133828096		✓	SGA Memory
shadow_core_dump	partial	✓		Diagnostics and Statistics
shared_memory_address	0	✓		SGA Memory
shared_pool_reserved_size	121634816	✓		Pools
shared_pool_size	0	✓	✓	Pools
shared_server_sessions		✓	✓	Shared Server
shared_servers	1	✓	✓	Shared Server
skip_unusable_indexes	TRUE	✓	✓	Miscellaneous
smtp_out_server		✓	✓	Miscellaneous
sort_area_retained_size	0	✓	✓	Sort, Hash Joins, Bitmap Indexes
sort_area_size	65536	✓	✓	Sort, Hash Joins, Bitmap Indexes
spfile	D:\ORACLE...			Miscellaneous
sql92_security	FALSE	✓		Security and Auditing
sql_trace	FALSE	✓	✓	Diagnostics and Statistics
sql_version	NATIVE	✓		Miscellaneous
sqltune_category	DEFAULT	✓	✓	Miscellaneous
standby_archive_dest	%ORACLE...	✓	✓	Standby Database
standby_file_management	MANUAL	✓	✓	Standby Database
star_transformation_enabled	FALSE	✓	✓	Optimizer
statistics_level	TYPICAL	✓	✓	Miscellaneous
streams_pool_size	0	✓	✓	Miscellaneous
tape_async_io	TRUE	✓		Backup and Restore
thread	0	✓	✓	Cluster Database

Figure 6.2 Oracle 10g memory management: to enable ASMM for SGA, set the initialization parameter `SGA_TARGET` to a nonzero value and the `STATISTICS_LEVEL` initialization parameter to `TYPICAL` or `ALL`. In contrast, to enable/disable automatic management for PGA, set the `WORKAREA_SIZE_POLICY` initialization parameter to `AUTO/MANUAL` and `PGA_AGGREGATE_TARGET` to a non-zero value.

Figure 6.2 confirms the settings of the above two initialization parameters set to have ASMM enabled. For your Oracle-based application, what value `SGA_TARGET` should be set to should be based on the performance and scalability tests with proper workloads. Then, one should observe the usage of the SGA over a period of time so that it can be adjusted to a more accurate value before settling down.

When enabling ASMM with the SGA, one can leave all the sub-areas of the SGA unspecified or set to 0 (See Figure 6.1). However, keep in mind that:

- Only four SGA sub-areas participate in ASMM: database block buffer cache, Java pool, large pool, and shared pool. All other buffer caches (`KEEP`, `RECYCLE`, `STREAMS`, and `LOG BUFFERS`, etc.) are not affected by ASMM. These caches either take their default values or need to be set manually based on the application specific requirements.

- Even with ASMM used for automatically managing the SGA, one can still manually set the size of a managed buffer cache, which will be considered by Oracle as the minimum amount of memory that a user knowingly specified for that managed buffer cache. This is especially necessary when your application demands a minimum amount of memory for one of those buffer caches; otherwise, your application will not perform or scale properly. This to some extent gives us some control over each sub-area of an SGA while having the SGA managed automatically by Oracle.
- Memory areas for the shared pool, large pool, Java pool, and buffer cache are allocated in units of granules. The granule size is 4 MB if the SGA size is less than 1 GB, and changes to 16 MB if the SGA size is greater than 1 GB. The minimum size for each of those four pools is 4 MB.

There is another closely related initialization parameter named `SGA_MAX_SIZE`. This is the parameter that specifies the limit or the maximum size an SGA is restricted to. One can consider the values of `SGA_TARGET` and `SGA_MAX_SIZE` the lower and upper bounds to an SGA if they are not set to the same value. However, keep in mind that only setting the parameter `SGA_TARGET` to a non-zero value, together with a proper value for the parameter `STATISTICS_LEVEL`, enables ASMM, and setting `SGA_TARGET` to zero would disable ASMM.

The following V\$ views provide information about the dynamic SGA resizing operations:

- `V$SGA_CURRENT_RESIZE_OPS`. This V\$ view contains information about the current SGA resizing operations in progress.
- `V$SGA_RESIZE_OPS`. This V\$ view contains information about the last 400 completed SGA resizing operations.
- `V$SGA_DYNAMIC_COMPONENTS`. This V\$ view contains information about the dynamic components in SGA, as was obtained and shown in Table 6.1.
- `V$SGA_DYNAMIC_FREE_MEMORY`. This V\$ view contains information about the amount of free memory in SGA that is subject to resizing.

Next, let's take a look at how the other memory area, the PGA, is sized in Oracle.

6.3 PGA SIZING: `PGA_AGGREGATE_TARGET`

The PGA is a private memory area, mainly consisting of several SQL work areas whose sizes are determined by the initialization parameters of `SORT_AREA_SIZE`, `HASH_AREA_SIZE`, `BITMAP_MERGE_AREA_SIZE` and `CREATE_BITMAP_AREA_SIZE`. A PGA is managed automatically by default (you flip between manual and auto memory management for PGA by setting the parameter `WORKAREA_SIZE_POLICY` to `MANUAL` or `AUTO`). Correspondingly, there is an initialization parameter named `PGA_AGGREGATE_TARGET` that sets the upper

Table 6.2 PGA Statistics of an Oracle 10g Server

NAME	VALUE
aggregate PGA target parameter	1,707,081,728
aggregate PGA auto target	1,522,225,152
global memory bound	170,700,800
total PGA in use	15,718,400
total PGA allocated	34,438,144
maximum PGA allocated	37,992,448
total freeable PGA memory	8,323,072
process count	21
max processes count	23
PGA memory freed back to OS	822,607,872
total PGA used for auto work areas	0
maximum PGA used for auto work areas	1,536,000
total PGA used for manual work areas	0
maximum PGA used for manual work areas	0
over allocation count	0
bytes processed	150,883,328
extra bytes read/written	0
cache hit percentage	100
recompute count (total)	2,328

limit to a PGA (note the word AGGREGATE means summing over all users). Oracle maximizes the performance of all memory-intensive SQL operations by maximizing the number of work areas that are using an optimal amount of PGA memory while staying below the limit set by the PGA_AGGREGATE_TARGET parameter.

The details of a PGA can be queried with the following SQL (the first statement is to limit the length of the column name so that the output for a row would fit in one line):

```
SQL> column name format a40
SQL> select name, value from V$PGASTAT;
```

Table 6.2 shows the output obtained with the above SQL executed on the same Oracle 10g setup as mentioned in the previous section (you can verify some items in this table with the PGA data shown in Figure 6.1. Note that they may not match with each other due to their dynamic nature). Let's explain some of the statistics listed in this table as follows:

- *Aggregate PGA Target Parameter.* This is the value of the PGA_AGGREGATE_TARGET parameter. If this parameter is set to zero, automatic management of the PGA memory is disabled.
- *Aggregate PGA Auto Target.* This is the amount of PGA memory that Oracle can manage automatically for work areas. It should not be too small relative to the value of PGA_AGGREGATE_TARGET. Otherwise, SQLs will not run optimally.

- *Total PGA in Use/Allocated.* This is the current amount of PGA memory in use or allocated by the instance. It's seen that 15.7 MB out of 34.4 MB or 46% of PGA is in use.
- *Cache Hit Percentage.* Finally it's seen that the cache hit percentage is as high as 100%, which indicates the high usage of the PGA.

It's hard to know precisely what value should be set to PGA_AGGREGATE_TARGET when starting up with an Oracle-based application. Oracle recommends a three-stage procedure for PGA_AGGREGATE_TARGET to settle down to an optimal setting:

1. Set PGA_AGGREGATE_TARGET to 20% of the SGA size. However, this might be too low for a large data support system (DSS) or data warehouse application.
2. Apply a typical workload to your application and see if your PGA is under-sized or over-sized with the help of the PGA statistics you can collect.
3. Tune PGA_AGGREGATE_TARGET using Oracle PGA advisor available on the EM DBConsole installed with your Oracle server.

Finally, note from Figure 6.1 that at the bottom it's suggested that "The sum of PGA and SGA should be less than the total system memory minus memory required by the OS and other applications." Ideally, one should size the amount of memory that Oracle needs for a specific application, and then plan for the total amount of physical memory required by Oracle, OS and application all together. Bluntly setting a specific percentage of the total available memory on a system to Oracle without going through a rigorous sizing exercise is not a sound practice.

This concludes our discussion of memory management in Oracle 10g. In the next chapter, we will explore how memory is managed in Oracle 11g.

6.4 SUMMARY

This chapter discussed how an SGA and a PGA are managed in Oracle 10g. The choice is whether letting Oracle manage each area automatically or letting an Oracle DBA manage each area manually. It is important to understand that even with automatic memory management for a memory area, SGA or PGA, Oracle does not automatically manage all the sub-areas.

In addition to introducing all the major concepts associated with Oracle memory management, some instructions have been given on how to set proper values for some of the relevant initialization parameters to enable or disable automatic memory management. We have also discussed a strategy on how one can progressively size an SGA and a PGA in a production environment. If you start new with an Oracle application, you might want to consider the SGA and PGA sizing recommendations as illustrated in Figure 6.3, which was recommended by Oracle. However, keep in mind

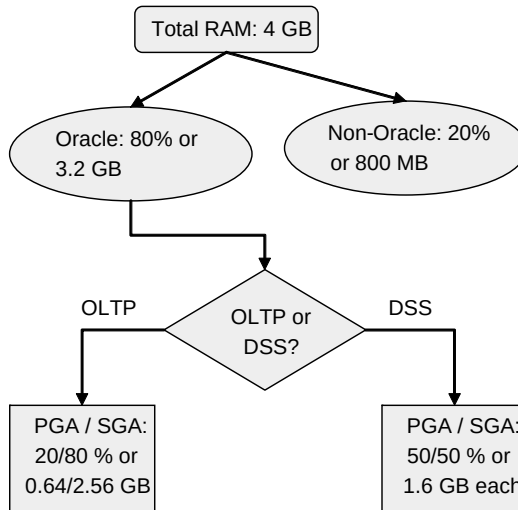


Figure 6.3 Oracle 10g memory sizing guidelines using an example Windows system with a 4GB total RAM. For a Linux system, 90% instead of 80% total memory is recommended for the Oracle Server.

that these are not hard-cut rules, as it may depends on a lot of factors such as your hardware limit, OS, workload characteristics, and so on. If you start with or upgrade to Oracle 11g, Oracle can automatically manage SGA and PGA as one entity. This is the subject of the next chapter. We will see how the latest version of Oracle 11g differs from 10g in memory management.

RECOMMENDED READING

Refer to the following Oracle documents for more information on how Oracle 10g manages memory:

Oracle Corp, *Oracle Database Concepts*, 10g Release 2 (10.2) B14220-02 (542 pages), October 2005, available free online at: http://download.oracle.com/docs/cd/B19306_01/server.102/b14220.pdf.

This document has four parts (with Part II very relevant to this chapter):

Part I What is Oracle?

Part II Oracle Database Architecture

Part III Oracle Database Features

Part IV Oracle Database Application Development.

Oracle Corp, *Oracle Performance Tuning Guide*, 10g Release 2 (10.2) B14211-03 (474 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B19306_01/server.102/b14211.pdf

Chapter 7 of this document describes Oracle memory configuration and use.

For the SQL statement described in Section 6.1 for querying pools, see:

T. Kyte, *Expert One-on-One Oracle*, A Press, New York, 2001.

EXERCISES

- 6.1** List the pros and cons of automatic versus manual memory management in general. If you are working on a server product, either in development or in production, explore how memory is managed in the server.
- 6.2** What is Oracle ASMM? Does ASMM automatically manage all sub-areas of an SGA? Is ASMM enabled by default?
- 6.3** Which parameters determine whether an SGA is managed automatically or manually?
- 6.4** What's the difference between the two parameters of `SGA_TARGET` and `SGA_MAX_SIZE`? How should one set those two parameters?
- 6.5** Which V\$ views contain information about the memory distribution of an SGA and a PGA? Query your Oracle database with those V\$ views and explore how memory is distributed among all sub-areas.
- 6.6** What's the desirable ratio between an SGA and a PGA? Are both an SGA and a PGA managed automatically by default in Oracle 10g?
- 6.7** What parameters determine whether a PGA is managed automatically or manually? How can one verify whether an Oracle database has its PGA managed automatically or manually?
- 6.8** What's the recommended strategy for sizing each major memory area in Oracle 10g?

7

Oracle 11g Memory Management

Every artist dips his brush in his own soul, and paints his own nature into his pictures.

—Henry Ward Beecher

In Oracle 10g, an SGA and a PGA can be managed automatically as two separate entities. In contrast to 10g, Oracle 11g allows an SGA and a PGA to be managed automatically as one single entity. This has further simplified memory management in the sense that one only needs to specify how much total physical memory to allocate to Oracle, and Oracle manages that total amount of memory between an SGA and a PGA automatically and dynamically. Other than that, everything else in 11g under the hood remains largely the same as Oracle 10g.

The objective of this chapter is to help you understand how Oracle 11g has combined the SGA and PGA into one area conceptually as one single entity to manage automatically. However, this does not mean that the concepts of an SGA and a PGA do not apply any more. These concepts still are applicable, but the ratio of a PGA to an SGA is variable while having the total amount of memory kept fixed.

This chapter consists of the following main sections:

- Automatic Memory Management (AMM)
- Memory Sizing Options Configurable at Database Creation Time
- Checking Memory Management and Usage Distribution at Run Time

Let's start with understanding how Oracle automatically manages the entire memory assigned to it.

7.1 AUTOMATIC MEMORY MANAGEMENT (AMM)

Oracle 11g introduced Automatic Memory Management (AMM) at one level above the SGA and PGA. With the concepts of an SGA and a PGA still inherited from the previous versions of Oracle, Oracle 11g manages an SGA and a PGA as a whole within the same perimeter. AMM is achieved in Oracle 11g with two more memory-related initialization parameters introduced: `MEMORY_TARGET` and `MEMORY_MAX_TARGET`. The parameter `MEMORY_TARGET` defines the desirable target memory to be tuned to and to be settled down eventually, while the parameter `MEMORY_MAX_TARGET` defines the limit that should not be trespassed.

Note that the ASMM for SGA and automatic management for PGA available in Oracle 10g remain to be available in Oracle 11g. Therefore, one can disable AMM in Oracle 11g and fall back to the same memory management schemes available in Oracle 10g, if desirable.

Oracle strongly recommends the use of AMM to manage the total amount of physical memory allocated to an Oracle Server. If one does not want to use AMM and prefers a manual approach, then consider using the Memory Advisor available in Oracle 11g.

Now let's see how AMM can be enabled at the time when an Oracle 11g database is created.

7.2 MEMORY SIZING OPTIONS CONFIGURABLE AT DATABASE CREATION TIME

If you refer back to Figure 2.14(a), you would see that AMM can be set when you create your Oracle 11g database. In that case, the `Typical` option was chosen, and a total memory size of 2354 MB (or 40% of the total RAM on that system) was allocated to SGA and PGA combined. Next, the check box of `Use Automatic Memory Management` was checked, which specified that AMM should be enabled.

As is seen in Figure 2.14(a), you can alternatively choose `Custom`, which would allow you to fall back to Oracle 10g's ASMM feature as was described in the previous chapter. Note also that if this option was chosen, 1536 MB would be allocated to SGA and 818 MB would be allocated to PGA. This would correspond to a ratio of 65%/35% to SGA/PGA out of a total 100% of 2354 GB.

If you have not sized your application and come up with a reliable sizing guide to how much memory your Oracle database would need and how the total Oracle memory should be partitioned between SGA and PGA, then taking Oracle's default settings as illustrated in Figure 2.14(a) is a good start point. You can evaluate how

these default settings would work for you and make adjustments as necessary over time. This can be done via Oracle Enterprise Management Database Control as discussed next.

7.3 CHECKING MEMORY MANAGEMENT AND USAGE DISTRIBUTION AT RUN TIME

If you have Oracle Database Control installed and enabled with your Oracle database, a lot of Oracle management tasks can be simplified significantly, including memory management. For example, to check and reconfigure memory management after you created your Oracle database, navigate to **Advisor Central** on your Database Control and you should see a screen similar to Figure 7.1. As you see, at the time when this screenshot was taken, AMM was enabled. You can disable AMM just by clicking the **Disable** button there. It also shows the allocation history of SGA and PGA over a period of time.

At the bottom of this screen, you could also check the current allocation of both SGA and PGA. Figure 7.2 shows the current memory allocation for SGA on a

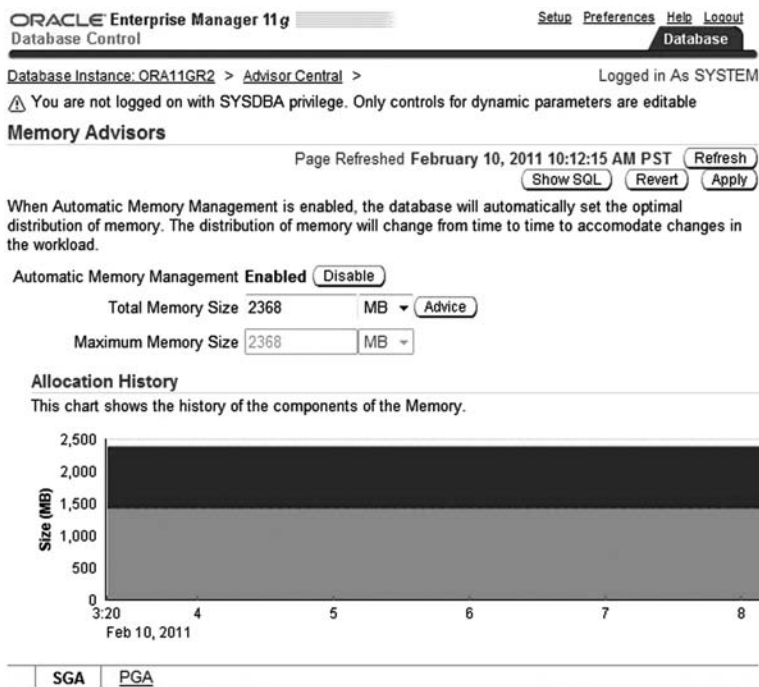


Figure 7.1 Oracle 11g memory management: SGA and PGA are managed together as a whole with the option of having each managed automatically or manually.

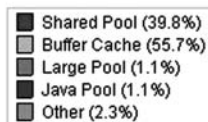
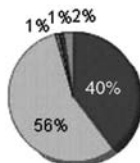
Current Allocation

Automatic
Shared
Memory
Management

Enabled

Total SGA
Size (MB) **1408**

SGA Component	Current Allocation (MB)
Shared Pool	560
Buffer Cache	784
Large Pool	16
Java Pool	16
Other	32



SGA

PGA

Figure 7.2 Oracle 11g memory allocation for SGA.

per-pool basis, indicating shared pool 39.8%, buffer cache 55.7%, large pool 1.1%, Java pool 1.1%, and Other 2.3%. Figure 7.3 illustrates the current PGA memory allocation. By clicking PGA Memory Usage Details, you can get a glimpse of how PGA is used according to varying work area sizes.

I hope this gives you a clear idea on how memory is managed in Oracle 11g. It's strongly recommended to take advantage of the Database Control for your management tasks, not only because it's more convenient than typing and querying those V\$ views but also because it's much less error-prone than manually editing a textual configuration file.

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started.

Aggregate PGA Target (B)

Current PGA Allocated (KB) **101062**

Maximum PGA Allocated (KB) **104654**
(since startup)

Cache Hit Percentage (%) **100**

PGA Memory Usage Details

☒ **TIP** The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications.

SGA

PGA

Figure 7.3 Oracle 11g memory allocation for PGA.

7.4 SUMMARY

This chapter discussed how the entire memory allocated to an Oracle 11g database can be managed automatically. We illustrated how one can enable AMM at the database creation time and how one can verify if AMM is enabled with an Oracle 11g database at run time. We also described how one can enable/disable AMM after an Oracle 11g database has been created.

The next chapter discusses Oracle storage structure, which is as critical as memory in terms of performance and scalability.

RECOMMENDED READING

The following Oracle 11g documents are helpful for understanding how memory is managed in Oracle 11g:

Oracle Corp, *Oracle Database Concepts*, 11g Release 1 (11.1) B28318-05 (556 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28318.pdf.

Oracle Corp, *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2(11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 7.1 How does AMM in Oracle 11g differ from ASMM in Oracle 10g? Can one disable AMM and enable ASMM in Oracle 11g?
- 7.2 How can one enable or disable AMM in Oracle 11g? Which parameters determine whether AMM is used in Oracle 11g?
- 7.3 There is a recommendation that 65% of the total amount of physical memory on a system be allocated to an Oracle database system up-front, regardless of how much RAM a system has. If you follow that recommendation, how would you set the parameter `MEMORY_TARGET` for Oracle server systems with a total amount of physical RAM varying from 4 GB, to 8 GB, to 16 GB, and to 32 GB, respectively? Is this fixed-percentage recommendation a proper recommendation from a practical point of view?
- 7.4 What's the difference between the two parameters of `MEMORY_TARGET` and `MEMORY_MAX_SIZE`? How should one set those two parameters?
- 7.5 Which V\$ views contain information about the memory distribution in an Oracle database? Query your Oracle database with those V\$ views and explore how memory is distributed among all sub-areas.

8

Oracle Storage Structure

Art is the only way to run away without leaving home.
—Twyla Tharp

Oracle storage structure is about how Oracle stores data on disks. We have seen that to maximize performance and scalability, Oracle has various elaborate designs on cache buffers and pools and so on. However, data must be eventually stored on physical disks after all. It has been my experience that for very large enterprise applications, performance and scalability bottlenecks are eventually found on disk I/Os when CPUs and memory are properly sized. Perhaps that's because storage technology couldn't keep up with the pace at which CPUs and physical memory have been advancing rapidly. Therefore, understanding Oracle storage structure is an essential part of tuning the overall performance and scalability of an Oracle-based enterprise application.

The objective of this chapter is to help you understand the concepts associated with Oracle storage structure. You will also learn some concrete skills such as managing tablespaces, data files, and redo logs as an integral part of your Oracle-related performance and scalability optimization efforts.

This chapter consists of the following main sections:

- Overview
- Managing Tablespaces
- Managing Data Files
- Managing Redo Logs

Let's begin next with an overview of how Oracle meets its data storage requirements based on the nature of data to be managed in the next section.

8.1 OVERVIEW

A good start point for understanding Oracle storage architecture is to look at how Oracle organizes and manages various files. As shown in Figure 8.1, the types of Oracle files may include application data files, server parameter files, control files, and redo log files. You can verify these types of files further from Figure 8.2, which was taken on an OEMJC against an Oracle 11g R2 install. Each type of Oracle file is explained as follows:

- **Control Files.** Control files are small binary files that contain such information about an Oracle database as: the database name, names and locations of associated data files and online redo log files, the timestamp of the database creation, the current log sequence number, and checkpoint information. Control files are needed when an Oracle database is being started.
- **Tablespaces.** Tablespaces are logical storage units for organizing all system and user data blocks, as was explained previously. Refer to Chapter 4 about the logical division of Oracle data storage into tablespaces, segments, extents, and data blocks. Data blocks are the smallest data storage units in Oracle (also called logical blocks, Oracle blocks, or pages). As is seen in Figure 8.2, there were six tablespaces with this Oracle 11g R2 install: EXMAPLE, SYSAUX, SYSTEM, TEMP, UNDOTBS1, and USERS (when an application is installed, application specific tablespaces will appear here as well. See Table 8.1 for some typical

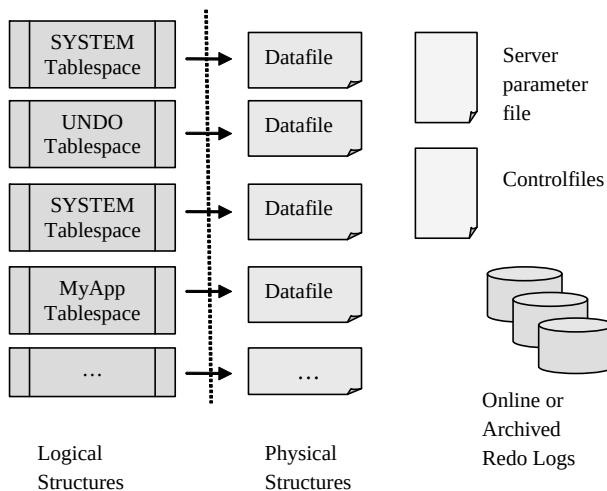


Figure 8.1 Oracle 11g logical and physical storage structures consisting of application data files, server parameter file, control files, and online or archived redo log files.

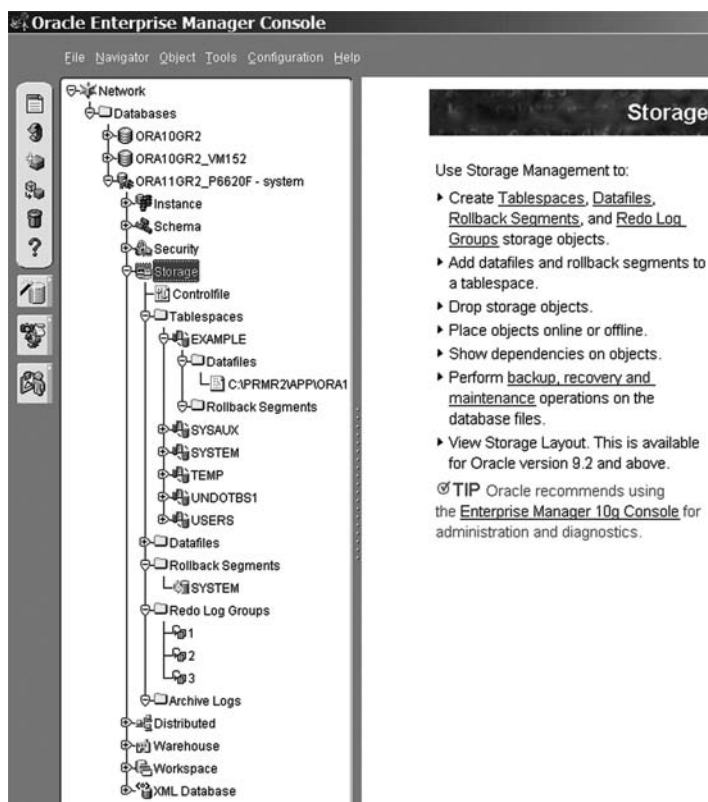


Figure 8.2 Oracle 11g storage structure reflected on the OEMJC console.

Table 8.1 Tablespace Usage from a Real Product

Tablespace	Total Size (MB)	Used (MB)	Used (%)
DATA01	1150	617.6875	54
DATA02	300	95.0625	32
DATA03	950	222.75	23
DATA04	50	.0625	0
INDEX01	50	3.3125	7
INDEX02	50	10.5625	21
INDEX03	600	264.9375	44
INDEX04	50	.0625	0
NSDP01	50	.0625	0
PEX01	6550	3759.6875	57
PSDPX01	116767.984	78664.6094	67
PVX01	72767.9844	49468.1094	68
PVX02	72767.9844	49109.7344	67
SYS_AUX	505.625	321.125	64
SYSTEM	300	193.585938	65
TEMP	1536	536	35
TOOLS	50	.0625	0
UNDOTBS	30112	6677.25	22
USERS	50	.0625	0

tablespace sizes out of a real product). The EXAMPLE and USERS tablespaces are set up to store data related to examples and users, whereas the SYSAUX and SYSTEM tablespaces are set up for storing system related data. The TEMP tablespace provides an area for performing intermittent operations during a transaction. The UNDO tablespace contains UNDO segments, which contain changes to transactional data prior to committing so that all effects of a SQL statement can be undone (or rolled back) if an error occurs during a transaction. It's also seen that associated with each tablespace are data files and rollback segments, which will be explained next.

- *Data Files.* Data files are physical files for actually storing system and user data. One data file belongs to one tablespace only, whereas one tablespace can contain multiple data files.
- *Rollback Segments.* The UNDO segments in Oracle 10g and 11g conceptually are the same as the Rollback Segments prior to Oracle 9i. However, unlike the rollback segments, the undo segments are managed automatically through undo tablespaces by Oracle in 11g. In this auto-management mode, no DBA interventions are required. If you set your Oracle database to run in *manual undo management mode* instead, then undo space is managed through rollback segments and no undo tablespace is used. In summary, the UNDO tablespace is a new feature in 11g for auto-managing rollbacks or undo's, whereas the Rollback Segments stays to be compatible with the manual rollback management feature, which existed prior to 11g.
- *Redo Log Groups.* Redo log groups contain both uncommitted and committed changes made to data during a transaction. The redo logs are divided into two parts: the online redo log and the archived redo log. Archived redo logs are created only at every checkpoint when the database operates in ARCHIVELOG mode. All redo logs are used for recovery purposes after a hardware, software, or media failure. Online redo logs are used for restoring a database back to a more recent state, while archived redo logs are used for restoring a database back to a more distant state. Oracle recommends creating a flash recovery area for storing and managing archived redo logs, but this is only necessary in production. In development or performance and scalability test environments, the archived redo log or a flash recovery area is not necessary in general unless you are testing archiving and recovering performance.

In the remainder of this chapter, we discuss how Oracle manages tablespaces, data files, and redo logs. Let's start with how Oracle manages tablespaces next.

8.2 MANAGING TABLESPACES

As explained previously, to simplify maintenance tasks, Oracle divides a database into logical units called `tablespaces` at the highest level. Tablespaces help create a means to physically locate data on storage. Some tablespaces are created by default, such as the SYSTEM and USERS tablespaces, while an application tablespace is

ORACLE Enterprise Manager 11g Database Control

Database Instance: ora11gr1 > Tablespaces > Edit Tablespace: SYSTEM

Actions: Add Datafile [Go] [Show SQL] [Revert] [Apply]

General Storage Thresholds

Name: SYSTEM

Bigfile tablespace: No

Extent Management

☒ Locally Managed
☐ Dictionary Managed

Type

☒ Permanent
☐ Set as default permanent tablespace
☐ Encryption [Encryption Options]
☐ Temporary
☐ Set as default temporary tablespace
☐ Undo

Status

☒ Read Write
☐ Read Only
☐ Offline
 Offline Mode: Normal

Datafiles

Select	Name	Directory	Size (MB)	Used (MB)
☒	SYSTEM01.DBF	C:\APP\HELIV\ORADATA\ORA11GR1	1,690.00	

Figure 8.3 Oracle 11g tablespace configuration.

created when the application is set up and configured against its Oracle database. An application tablespace contains all application specific objects.

Figure 8.3 shows how a tablespace is configured in general. First, it shows that it's not a Bigfile tablespace, which is a new type of tablespace introduced in Oracle 10g. With the new Bigfile tablespace feature, one can create a big tablespace with a single big file, for example, storing up to 32 TB with an 8K-block size. This feature can help minimize the tasks of managing data files while maximizing the performance related to managing those data files. It becomes especially desirable when used with the Automatic Storage Management (ASM) feature.

The next attribute of a tablespace is Extent Management as shown in Figure 8.3. It could be either locally managed or dictionary managed. The locally managed option is more advantageous performance-wise and recommended in general.

Following the extent management in Figure 8.3 is the tablespace type. There are three types of tablespaces as indicated in Figure 8.3:

- *Permanent Tablespaces.* A permanent tablespace is for storing data permanently from system and application perspectives.

- *Temporary Tablespaces.* A temporary tablespace is for storing temporary data, for example, data created during a SQL sort operation. Typically, an application has its own temporary tablespace created when it is installed and set up initially against its Oracle database.
- *Undo Tablespaces.* An Undo tablespace is for storing undo data for a variety of purposes, for example, to roll back transactions, to provide read consistency, and to enable features such as Oracle Flashback Query introduced in Oracle 9i. With Flashback Query, data can be viewed as it existed in the past.

Following the tablespace type in Figure 8.3 is the status of a tablespace, which includes: Read Write, Read Only, and Offline. The Read Only option disables the write permission from the Read Write option, whereas the Offline option disables user access to the tablespace completely. The offline mode includes Normal, Temporary, Immediate and For Recover.

At the bottom in Figure 8.3, the data files associated with the tablespace are shown, along with such attributes as the file name, file directory, size, and usage.

Figure 8.4 shows storage options such as whether the extent allocation and segment space management are automatic, whether the tablespace is compressed, and whether redo logging is enabled. Note the block size is shown at the bottom.

Database Instance: ora11gr1 > Tablespaces > Logged in As SYSTEM

Edit Tablespace: SYSTEM

Actions:

Extent Allocation

Allocation Type **Automatic**

Segment Space Management

Type **Manual**

Compression Options

Enabling data segment compression can reduce disk usage.

Compression ☒ Disabled
☐ Enabled on direct-path INSERT operations only
☐ Enabled on all operations

Enable logging

☒ Yes
 Generate redo logs for creation of tables, indexes and partitions, and for subsequent inserts.
 Recoverable

☐ No
 Redo log entries are smaller, the above operations are not logged and not recoverable.

Block information

Block Size (B) **8192**

Figure 8.4 Oracle 11g tablespace storage options.

Database Instance: ora11gr1 > Tablespaces > Logged in As SYSTEM

Edit Tablespace: SYSTEM

Actions:

General Storage **Thresholds**

Available Space (MB) 32,767.98	Space Used (MB) 683.19
Space Used (%) 2.08	Available Free Space (MB) 32,084.80

Tablespace Full Metric Thresholds

Monitor the fullness of the tablespace using either of the metrics below.

Space Used (%)

A warning or critical alert will be generated if the percentage of space used exceeds the corresponding threshold.

☒ Use Database Default Thresholds

Warning (%) **85**

Critical (%) **97**

☐ Specify Thresholds

Warning (%)

Critical (%)

☐ Disable Thresholds

Free Space (MB)

A warning or critical alert will be generated if the remaining free space falls below the corresponding threshold. This metric is especially useful for large tablespaces.

☒ Use Database Default Thresholds

Warning (MB) **Not Defined**

Critical (MB) **Not Defined**

☐ Specify Thresholds

Warning (MB)

Critical (MB)

☐ Disable Thresholds

Figure 8.5 Oracle 11g tablespace threshold settings.

Figure 8.5 illustrates the capacity thresholds associated with a tablespace. With automatic extent allocation, a tablespace will be extended automatically if it reaches its size limit. However, automatic extent allocation is set from the data file properties, not from the tablespace properties, as shown in Figure 8.6 with the check box for *Automatically extend data file when full* checked.

Regarding the segment space management options shown in Figure 8.4, it refers to how the free space in a tablespace is managed. With automatic segment space management, objects in the tablespace automatically manage their free space, while with the manual management, objects in the tablespace will manage their free space using free lists, which is less efficient. Therefore, automatic management of a tablespace's segments is preferred over manual management for performance reasons.

8.3 MANAGING DATA FILES

How data files are managed can have an impact on the performance and scalability of an Oracle server and ultimately on the performance and scalability of your Oracle-based application.

Oracle data files are physically stored on a storage device. In production, Oracle data files typically are stored on high-performance Storage Area Networks (SANs)

Database Instance: ora11gr1 > Tablespaces >
 Edit Tablespace: SYSTEM: Edit Datafile

File Name **SYSTEM01.DBF**
 File Directory **C:\APP\HELIU\ORADATA\ORA11GR1**
 Tablespace **SYSTEM**
 Status ☒ Online ☐ Offline
 File Size

Storage

☒ Automatically extend datafile when full (AUTOEXTEND)
 Increment
 Maximum File Size ☐ Unlimited
☒ Value

Figure 8.6 Oracle 11g tablespace auto-extension configuration.

with multiple disks lumped together to form a specific RAID configuration. However, in an R&D test environment, one may not have this kind of enterprise-class storage. Nevertheless, we can circumvent such a situation as described below.

In an R&D test environment, if a SAN-based storage is not available, a much less costly approach is to configure an internal RAID using multiple local disks on a server system. Typically, three separate disks configured as a RAID 0 would be sufficient. Then all Oracle data files including redo log files can be placed on such a RAID. One can get hundreds of GB or TB storage capacity easily on a commodity server with built-in or configurable RAID configurations, which provides much better IO performance than simply putting all data on a single local disk.

To emphasize more, if it's even difficult to have a RAID 0 configuration configured with three separate disks, the minimum requirement is to use at least two independent disks to spread data files across. In general, it's less desirable to install an entire Oracle server onto a single local disk unless one is not concerned with performance and scalability.

In addition to the question of *where* to place data files, one should also consider the size limit of a data file and the number of data files an Oracle tablespace can have. Those limitations are operating system dependent, and if a certain limit is being exceeded, a warning or an error would be thrown and one can make corrections accordingly.

The other factor one needs to be concerned with is the usage of a tablespace, which is determined by the total disk usage of all data files of a tablespace. One can configure a data file to grow automatically when the limit is reached, or let it grow, constantly check it, and adjust it accordingly when it gets close to 100% full. Figure 8.7 shows tablespace usage taken at a point of time for all tablespaces on an Oracle server.

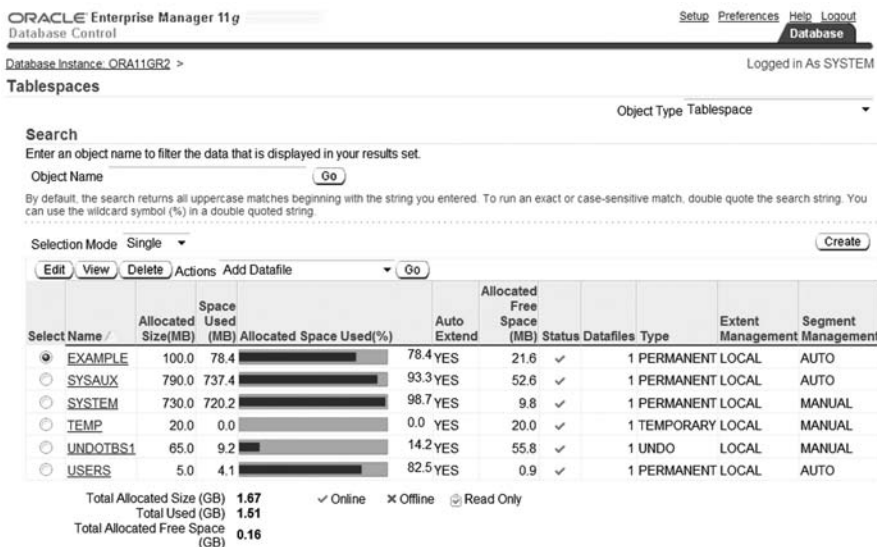


Figure 8.7 Oracle 11g tablespace usage statistics.

As is seen, managing data files is a simple task. A very important point to keep in mind is choosing proper storage configurations such as RAIDs. Also, it's easy to determine whether your Oracle database is bottlenecked on I/O by simply looking up the average read and write times from an AWR report with a proper workload applied. We will present some real world case studies later to help you learn how to analyze collected counters and identify I/O issues.

Next, let's explore how Oracle manages redo logs.

8.4 MANAGING REDO LOGS

For high transaction rate applications, properly configured redo logs are as critical as data files in terms of performance and scalability. It's necessary to understand all the related concepts and make sure that redo logs are not hindering the performance and scalability of an Oracle-based application.

Figure 8.8 illustrates the Oracle redo log structure. Oracle redo logs consist of multiple redo log groups, with each redo log group containing one or more redo log files. As shown in the figure, each redo log group has such attributes as status, group number, # of members, archived, size, sequence and first change #. Figure 8.8 also shows how one can activate an inactive redo log group, create a new redo log group, and switch a log file from the current one to a new one.

Like many other settings of an Oracle server, the desirable size of a redo log file and the number of redo log groups depend on applications. In general, there are no reliable, quantitative sizing guidelines to help predetermine the optimal settings for such parameters. However, Oracle has provided a very useful feature called Oracle

ORACLE Enterprise Manager 11g Database Control

Setup Preferences Help Logout

Database Instance: ORA11GR2 > Logged in As SYSTEM

Redo Log Groups

Object Type Redo Log Group

Search

Enter an object name to filter the data that is displayed in your results set.

Object Name

By default, the search returns all uppercase matches beginning with the string you entered. To run an exact or case-sensitive match, double quote the search string. You can use the wildcard symbol (%) in a double quoted string.

Selection Mode

Actions

Select	Group	Status	# of Members	Archived	Size (KB)	Sequence	First Change#
☒	1	Current	1	No	51200	328	6943016
☐	2	Inactive	1	No	51200	326	6890858
☐	3	Inactive	1	No	51200	327	6917305

Figure 8.8 Oracle 11g redo log management.

Wait Interface (OWI) that can help determine the optimal settings for many Oracle server parameters, to some extent. OWI is the subject of the next chapter.

8.5 SUMMARY

This chapter reviewed Oracle storage structure in terms of tablespaces, data files, and redo logs. The size of an Oracle database grows with time. One should plan for accommodating larger and larger spaces needed with time. Constant monitoring of the tablespace usage is called for. One can set auto-expansion on a tablespace. However, this can only be set at the data file level. Also, it's necessary to make sure that sufficient storage space is reserved for auto-extension of a tablespace.

It was emphasized that the most important decision about configuring the data storage for Oracle is to use RAID configurations rather than a single local disk for all data storage needs. Most large-scale enterprise applications are eventually bottlenecked on I/O, after CPUs and memory have been properly sized. Therefore, adequate storage configurations such as RAID's are required not only in production but also in performance and scalability test environments. Best practices and more accurate IO sizing guidelines should be propagated to customers to help prevent potential IO bottlenecks with your products.

RECOMMENDED READING

Part II Oracle Database Structure and Storage of the following Oracle document provides more in-depth coverage on the topics discussed in this chapter:

Oracle Corp, *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.

For the impacts of storage on the performance and scalability of enterprise applications, refer to the following text:

H. H. Liu, *Software Performance and Scalability: A Quantitative Approach*, John Wiley & Sons, Hoboken, May 2009.

EXERCISES

- 8.1 What would be the most likely bottleneck with large-scale enterprise applications? How could one prevent such a bottleneck proactively?
- 8.2 Does application data share the same storage segment with indexes? What could be the potential impacts of sharing or not sharing the same segment between data and indexes on the performance and scalability of an Oracle-based application?
- 8.3 How would you make sure that your Oracle server would not encounter 100% disk full problem with time? How do you set auto-extension with a tablespace? Is it set at the tablespace or data file level? If you are working on a production Oracle database, is auto-extension enabled or disabled with your application tablespace?
- 8.4 What is the difference between a rollback segment and an UNDO segment?
- 8.5 How are the redo logs rotated? How would you determine if your redo logs are causing performance problems to your Oracle server?

9

Oracle Wait Interface (OWI)

Art disturbs, science reassures.
—**Georges Braque**, *Le Jour et la nuit*

An Oracle server is a complex software system, given its pivotal role in supporting various types of large-scale enterprise applications. A basic question is how one would troubleshoot various performance and scalability issues encountered both in test and production environments. Oracle started with guiding users to pay close attention to various cache ratios as the primary performance tuning methodology. This was well justified when memory was a scarce resource decades ago. With time, however, when memory is no longer a limitation, Oracle shifted its performance tuning methodology from ratio based to wait event based. The wait-event-based performance tuning methodology is more scientific, as it is deeply rooted in well-established queuing theory matured in 1970s as the primary scientific discipline for analyzing the performance of a system that fulfills its tasks by consuming various types of resources in a customer-server fashion.

To support the wait-event-based performance tuning methodology, Oracle started with version 8 to build an elaborate framework, which eventually evolved into what is called Oracle Wait Interface (OWI) today. I'll help you get an adequate exposure to the OWI in this chapter, but we will not delve into the details of every type of wait event. It's more important to help you understand how OWI works and how to apply OWI to

solving real world performance and scalability issues than turning this chapter into a *complete reference* for the OWI to compete with the Oracle product documents or some specialized OWI texts publicly available. The other reason for not covering all wait events in this text is that there are simply too many types of wait events and what wait events one encounters are completely dependent on many factors such as a specific application, Oracle server configurations, and how the application and Oracle communicate with each other, and so on.

This chapter consists of the following main sections:

- Ratio-Based versus OWI-Based Oracle Performance Tuning Methodologies
- Wait Event—the Core Concept of OWI
- Classification of Wait Events from OWI
- The Other Part (CPU Time) of the Equation of *Elapsed Time = CPU Time + Wait Time*
- AWR as a Compass to Tuning Oracle Performance and Scalability

To put it into perspective, let's first have a brief discussion in the next section on the two drastically different Oracle performance tuning methodologies: the ratio-based approach and the OWI-based approach.

9.1 RATIO-BASED VERSUS OWI-BASED ORACLE PERFORMANCE TUNING METHODOLOGIES

In the previous chapters, we emphasized how important it is to keep as many data blocks and objects in memory as possible. We also covered various Oracle memory management schemes aimed at managing various buffer caches more efficiently. In fact, the notion of having the highest possible cache hit ratios had helped form the ratio-based Oracle performance and scalability tuning methodology prior to Oracle 9i. If there were an Oracle performance issue, the first suspicion would be that the buffer cache ratios might be low and more memory was needed.

It is undeniable that the ratio-based Oracle performance tuning approach had its merits and might work some times. However, completely depending on the ratio-based approach to solve Oracle performance problems had turned out to be insufficient, as high buffer hit caches are the necessary conditions for the high performance of a software system, but not necessarily the sufficient conditions for guaranteeing the high performance of a software system. The case study presented in section 6.2 of *Isolating Performance and Scalability Factors* (Liu, 2009) is a perfect anecdote to why high buffer cache hit ratios are not necessarily a sufficient condition for high performance. In that case study, all buffer hit ratios were close to 100%, yet the poor scalability occurred. The poor scalability issue was resolved effectively by adding covering indexes, guided by the OWI-based Oracle performance tuning methodology. For your reference, Figure 9.1 shows the buffer hit ratios before and after the covering indexes were added. As you see, the buffer hit ratios really didn't change much, yet the

9.2 WAIT EVENT—THE CORE CONCEPT OF OWI

To start off our discussion on OWI, let me first introduce a few key concepts in the framework of queuing theory that OWI is deeply rooted in. Queuing theory has been the theoretical foundation for understanding the performance of a system that completes its various tasks by consuming various types of resources. The basic concepts of queuing theory include `wait events` and `service demands`. The concept of a wait event can be expanded to what is called a `wait chain`, which appears as a series of *queues* at various resources formed in certain predetermined order. Wait events and wait chains associated with system resources are the core elements in understanding the performance and scalability factors associated with a software product. The concept of *service demand* quantifies the total time spent on a resource servicing multiple visits of a user over a transaction under an uncontended condition. For a more in-depth understanding of queuing theory and its application to solving software performance and scalability issues, I prefer to refer you to my other text (Liu, 2009) rather than repeating bulk of it here.

It's important to make a distinction between an *event* and a *wait event*. For example, the fact that an error has occurred is an event, but not a wait event. An error event lacks two fundamental elements to be qualified as a wait event. First, it is not a necessary step of a transaction. Secondly, the end-to-end elapsed time associated with its occurrence does not contribute to the total transaction time of a computing task. In contrast, reading or writing blocks of data from or to a data storage device is a wait event when measured with the above two metrics. We will see a lot more such wait events as you read along.

Understanding various isolated wait events is important. However, it makes more sense to look at all the relevant wait events in a wait event chain so that the longest wait event can be isolated as the performance and scalability bottleneck. Once a wait event is identified as the performance and scalability bottleneck, the next step is to analyze the quantitative performance data logged either in a production environment or a lab test environment and look for the root cause that causes the bottleneck wait event to occur. In order to be able to complete this step successfully, it's very necessary to take a scientific attitude and look at all potential factors based on facts rather than wild guessing. Once the leading factors resulting in the bottleneck wait event are sorted out, one can apply the corresponding fixes accordingly. This process of identifying the bottleneck wait event, sorting out all potential factors leading to the bottleneck wait event, and applying corresponding fixes should be repeated until there are no more obvious wait events that affect the performance and scalability of a software product.

Fortunately, Oracle has provided us with a solid, feature-rich OWI framework, which works exactly in the same perimeter of the causality-based model as explained above. OWI is centered on the concepts of resource and wait events, as is discussed above for solving the performance and scalability challenges exhibited from any applications using Oracle as the backend. The challenge is that there are both physical and logical resources, with logical resources and the associated wait events far outnumbering the physical resources and wait events.

In general, we hope that a performance bottleneck is related to a physical resource so that it's more obvious and there is less guessing work to do, as we have only a few distinct types of physical resource such as CPU, memory, disk I/O and network. But that may not be always the case. When the hardware for an Oracle server is sized properly, most likely, the performance bottleneck is with one or a few logical resources that represent the corresponding Oracle objects on Oracle's side. In such cases, knowledge about the Oracle internals is necessary to resolve the performance of the system bottlenecked on one or a few logical resources.

In the next section, we'll explore how wait events are presented to their consumers through OWI. Note that it will not be just a list of all Oracle wait events. Instead, I'll try to divide them up more logically so that they could be understood one at a time.

9.3 CLASSIFICATION OF WAIT EVENTS FROM OWI

The number of wait events available from various releases of Oracle has been growing rapidly, starting from 140 in Oracle 8.0 to about 400 in 9i, 873 in 10g, 959 in 11g R1, and to 1116 in 11g R2. Those wait events are assigned to different classes, which can be queried against the V\$EVENT_NAME dynamic performance view with the following command:

```
SQL>SELECT name, wait_class FROM V$EVENT_NAME ORDER BY
wait_class;
```

The above command returned 959 wait events together with their belonging wait classes on an Oracle 11g R1 server. To see how those events are classified, use the following command:

```
SQL>SELECT wait_class, count (wait_class) FROM V$EVENT_NAME
ORDER BY wait_class;
```

The above query returned 13 classes as shown in Table 9.1 with the corresponding count for each wait class included as well. In the table, the source resulting in wait events for each wait class is given as well along with a typical wait event example. Refer to Appendix C for a complete list of the wait events for each wait class. The wait events in the Idle and Other classes are omitted except the first wait event for each class, in order to save space while still giving a glimpse of what wait events are included in each wait class.

As shown above, wait events can be classified with corresponding wait classes. Wait events can also be classified based on the dependency relations or layers from the session and system points of view, represented by the following three V\$ views:

- **V\$SESSION_WAIT.** This view displays detailed information about an event or resource that a session is waiting for in real time. The wait event could be the last wait event that has completed or a wait event that the session still is waiting

Table 9.1 Wait Class and the Number of Wait Events in Each Wait Class

Wait Class (Count)	Event Source or Resource (Example)
Administrative (51)	DBA tasks (index rebuilding)
Application (15)	User code (lock waits)
Cluster (47)	RAC (global cache resources)
Commit (2)	Redo log writer (log file sync)
Concurrency (26)	Internal DB resources (latches)
Configuration (21)	Mis-configured parameters (shared pool size)
Idle (80)	Session waiting for work (SQL*Net related)
Network (35)	Network messaging (SQL*NET related)
Other (630)	Unusual wait events ('wait for EMON to spawn')
Queueing (4)	Streams related (...)
Scheduler (3)	Resource Manager ('resmgr: cpu quantum')
System I/O (23)	Background process I/O ('Network file transfer')
User I/O (22)	User I/O ('db file sequential read')

for. It provides a good troubleshooting start point to find out, for example, why a user is stuck. The `Event` attribute indicates the resource or event for which the session is waiting. The `WAIT_CLASS` attribute tells further about the origin of the wait event. The other three attributes, `P1TEXT`, `P2TEXT`, and `P3TEXT`, reveals the wait event parameters that can help place a wait event into proper context further.

- **`V$SESSION_EVENT`**. This view displays aggregated wait event statistics by session with such attributes as `EVENT`, `TOTAL_WAITS`, `TOTAL_TIMEOUTS`, `TIME_WAITED`, `AVERAGE_WAIT`, `MAX_WAIT`, `TIME_WAITED_MICRO`, and so on.
- **`V$SYSTEM_EVENT`**. This view is similar to `V$SESSION_EVENT` except that the aggregated wait event statistics are collected at the system level from all sessions. Note that the statistical data about a wait event is accounted for since the instance startup time.

As there are too many wait events to cover in this text, one should resort to the references listed at the end of this chapter to understand the details about a specific wait event. However, I'd like to list two wait events that are commonly encountered with I/O operations on large volumes of data, `db file scattered read` and `db file sequential read`, as follows:

- ***db file scattered read***. This wait event typically is caused by full table scans and full fast index scans, which read data blocks from physical disks into the buffer cache in memory. In this case, each read operation reads multiple data blocks, which is determined by the initialization parameter `DB_FILE_MULTIPLE_READ_COUNT`. It's called `scattered read`, as data blocks are not necessarily placed contiguously in the buffer cache. To avoid this wait event, one

should make sure that large tables are properly indexed so that Oracle server is not bringing unnecessary data blocks into the buffer cache.

- *db file sequential read*. This wait event typically is caused by reading from an index, table access by rowid, undo, or rollback segments, and so on. It's called sequential read, as data blocks are read one at a time and placed into contiguous areas of a buffer cache.

As described above, the `V$EVENT_NAME` view defines all wait events as a base for the three most fundamental OWI V\$ views, `V$SESSION_WAIT`, `V$SESSION_EVENT` and `V$SYSTEM_EVENT`. These three V\$ views provide us with finer granularities in looking into wait events on an individual event basis, in real time, and accumulatively since the startup time of an Oracle instance. However, this level of information might be too overwhelming for a human consumer to digest. Thus, instead of looking at the wait event statistics accumulated since the startup time of an Oracle instance, a more feasible method is to look at the wait event statistics that cover roughly the period of time over which the performance problems persisted. Besides, instead of looking at the wait events from a session or an entire system point of view without relating them to proper wait classes, it might make more sense to look at the wait events from the wait class point of view to help pinpoint quickly the common root causes for all sessions affected. Oracle provides opportunities for us to look into wait events from those different perspectives with the following V\$ views:

- *V\$SESSION_WAIT_CLASS and V\$SYSTEM_WAIT_CLASS*. These views provide information on wait events from the `WAIT_CLASS` perspective for a session or the entire system. For example, if the origin of a hot wait event is Administrative wait class, the reason why the Oracle server is slow might be due to a DBA-initiated, long, resource-intensive job.
- *V\$SESSION_WAIT_HISTORY*. This view provides information on the last 10 wait events from the `WAIT_CLASS` perspective for a session. The `Event` attribute signifies the resource or event for which the session is waiting, while a non-zero value for the `WAIT_TIME` attribute tells the total wait time for that event. A zero value for the `WAIT_TIME` attribute means the wait event is being waited for currently. The other three attributes, `P1TEXT`, `P2TEXT`, and `P3TEXT`, reveal the wait event parameters that can help place a wait event into proper context further.
- *V\$ACTIVE_SESSION_HISTORY*. Oracle takes snapshots of active database sessions once a second. A session is considered active if it is on the CPU or is waiting for an event that didn't belong to the `Idle` wait class. This view displays database activities on an active session basis. It encompasses a lot more information than the view `V$SESSION_WAIT_HISTORY` does. For example, it displays such additional information as user ID, SQLs executed, blocking sessions, client, executing module, executing OS program, and so on. Consult Oracle's Reference document for more information about this view.

- *V\$EVENT_HISTOGRAM*. This view provides a histogram of the number of waits (WAIT_COUNT) during a series of wait time intervals (or buckets) of < 1 ms, < 2 ms, < 4 ms, < 8 ms, < 16 ms, . . . , < 2²² ms and ≥ 2²² ms on an event basis.

One would think that the information made available solely with OWI is all we need for analyzing Oracle performance issues, but, in fact, that's not the case at all. Oracle stores other performance-related metrics in other V\$ views, for example, CPU usage information in the V\$SESSTAT and V\$SYSSTAT dynamic views. Let's briefly touch upon these two V\$ views in the next section, as sometimes they provide even more pertinent information about an Oracle performance and scalability issue than OWI.

9.4 THE OTHER PART (CPU TIME) OF THE EQUATION ELAPSED TIME = CPU TIME + WAIT TIME

Based on my experience, it's so easy to encounter a situation that all DB server CPUs are fully busy. You might be told that it's a good thing that all CPUs are working hard, which maximizes the server throughput. You might be even told that it's so easy to solve the problem: just adding more CPUs to your Oracle server or updating existing CPUs to faster ones. That's true if the DB server CPUs were executing useful code logic, but it might well be the case that those CPUs were "driven crazy" simply by some unthought-of inattention either in the application coding logic in the upstream of the database or on the database side because of missing proper indexes.

The reasons for paying close attention to the CPU usage of an Oracle database server are twofold. First, it's necessary to make sure that an Oracle server has sufficient CPU capacity for the work it is supposed to do. Secondly, we want to make sure that the CPU power of an Oracle server is put into good use. The case study to be presented in Chapter 24 is one of such compelling examples. In that case study, the Oracle server was spending excessive CPU time in performing logic reads from the buffer cache, which resulted in poor scalability as the performance degraded rapidly with data volumes. The question was why the Oracle server was driven into such a state. The problem was cured with a few covering indexes added to two critical tables (note that in that case adding more or faster CPUs won't be helpful because the needs for CPU resources caused by missing indexes were characteristically exponential and simply insatiable).

Also, recall one of the fundamental performance law equations that:

$$\textit{Total Elapsed Time} = \textit{Service Time} + \textit{Wait Time}$$

Note that the service time part on the right-hand-side of the above equation is equivalent to CPU time. Apparently, a large part of CPU time contributing to the total elapsed time is harmful to the performance of a system.

So where does an Oracle server store CPU times spent in various parts of Oracle? There are two V\$ views that can be used to query for such information: the

V\$SESSTAT view and the V\$SYSSTAT view, with one at the session level and the other at the system level. A more detailed discussion on each of these two views is given below:

- **V\$SESSTAT.** This view displays at the session level the value of a metric identified with the attribute of STATISTIC#, which is defined in the V\$STATNAME view.
- **V\$SYSSTAT.** This view displays at the system level the value of a metric identified with the attribute of STATISTIC#, which is defined in the V\$STATNAME view.

Appendix D presents a complete list of all metrics defined in the V\$STATNAME view, while Appendix E presents a complete list of all statistics from the view V\$SYSSTAT, both obtained with an Oracle 11g setup. If you already have some exposure to Oracle, you probably will be able to immediately identify some of the metrics that are very interesting in the context of troubleshooting Oracle performance and scalability issues, such as:

- #8 session logical reads
- #13 DB time
- #20 session uga memory
- #25 session pga memory
- #37–42: physical read/write total ...
- #47–50: db block gets ...
- #56–60 and #66–72: physical read(s)/write(s) ...
- #281–289: table scan(s)/fetch ...
- #315–319: index fast full scans ...
- #407–410: ... parallelized
- #430–434 parse time/count ...
- #447–449: sorts (...)
- ...

To illustrate how to query about a specific metric contained in the view V\$SYSSTAT, I executed the following command against one of my 11g installs, which returned a value of 7,456,860:

```
SQL>SELECT VALUE FROM V$SYSSTA WHERE STATISTIC# = '8' ;
```

That large value means that about 7.5 million logical reads have occurred up to the time of issuing the above query since my Oracle 11g instance was started up about one week ago. The purpose here is not to assess whether that many logical reads are normal, but rather to show how to query about a specific metric identified at

the session or system level. In a real world Oracle performance troubleshooting scenario, you need to query the statistics you are interested in multiple times, and then look at the deltas of those statistics in order to make sense out of them.

Obviously, the information presented so far about so many wait events as well as session and system level statistics is dizzying and overwhelming. In the next section, we explore how one can consume such vast amount of information in a more sensible way with the help of AWR reports.

9.5 AWR AS A COMPASS TO TUNING ORACLE PERFORMANCE AND SCALABILITY

Manually querying about those V\$ views might not be the first thing one would do in troubleshooting an Oracle performance and scalability problem. A wiser approach is to let Oracle do all the laborious work and we look at the summary report provided to us with some built-in features such as listing top *n* wait events and hot SQLs, and so on. In this regard, Oracle has done an outstanding, no-second job with a utility improved significantly over time from 9i to 10g and above—the Automatic Workload Repository (AWR) tool. The AWR tool does an in-memory aggregation on the wait events from two given snapshots taken at the start and end points of a period of time, and then outputs the summary into an HTML file for offline consumption.

Once again, Chapter 24 offers an impressive, real-product-based example of how one can make an OWI-based Oracle performance and scalability troubleshooting task an easy and enjoyable exercise—all through the use of an AWR report. I'll leave the topic of how to read an AWR report to the next chapter. In this section, I'll show you the logistics behind generating an AWR report, for example, how to enable AWR, and so on.

Enabling AWR is controlled by the initialization parameter `STATISTICS_LEVEL`. This parameter has three settings:

- **TYPICAL.** This is the default setting that enables AWR to collect all major statistics required for database self-tuning that provides best overall performance. This is recommended by Oracle for most environments.
- **ALL.** This setting adds additional statistics such as OS statistics and plan execution statistics on top of the statistics collected with the default setting of **TYPICAL**.
- **BASIC.** This setting disables many features from the default setting of **TYPICAL** such as:
 - AWR snapshots scheduled regularly
 - Automatic Database Diagnostic Monitor (ADDM)
 - All server-generated alerts
 - Automatic SGA Memory Management (ASMM)
 - Automatic optimizer statistics collection

- o Object level, service level, segment level, and timed statistics as well as monitoring of statistics
- o Buffer cache, MTTR (Mean Time To Recover), shared pool sizing, and PGA target advisories

As you see, you are turning most of Oracle OWI tuning lights off when you set `STATISTICS_LEVEL` parameter to `BASIC`. To know for sure what statistics level that parameter is set to and whether statistics and advisory are enabled at the session and system levels, query the `V$STATISTICS_LEVEL` view.

One particularly interesting aspect of an AWR report is that it sorts all SQLs based on a variety of metrics, for example, by elapsed time, by CPU time, by buffer gets or logical reads, and so on. I can't emphasize more that AWR reports are an indispensable part in troubleshooting the performance and scalability of an Oracle-based enterprise application both in development stage and in customer environments. When examining an AWR report, we typically look for enormous wait times caused by contentions. Contentions are caused by concurrent access to the underlying resources. Therefore, before giving a full coverage of an AWR report, let's get a good understanding of Oracle data consistency and concurrency in the next chapter.

9.6 SUMMARY

This chapter discussed the Oracle Wait Interface, which is the foundation of the new, wait-event-based Oracle performance tuning methodology. OWI was introduced in Oracle version 8 and matured in 10g and 11g. When used in conjunction with Oracle AWR reports, it's a powerful framework for resolving various Oracle performance and scalability issues—both reactively in external customer production environments and proactively in internal development stage.

However, if AWR reports are not available or do not suit your needs, then you may need to query those V\$ views such as `V$ACTIVE_SESSION_HISTORY` and `V$SYSSTAT` to troubleshoot your Oracle performance issue. If this is the approach you have to take, then it's recommended that you get familiar with all the wait events and system statistics as listed from Appendix C through Appendix E at the end of this text.

RECOMMENDED READING

The following Oracle product document covers more about OWI with detailed explanation about every type of wait events:

Oracle Corp, *Oracle Database Reference*, 11g Release 1 (11.1) B28320-03 (1132 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28320.pdf.

For a more complete text dedicated to the coverage of OWI, refer to the following text:

R. Shee, K. Deshpande, and K. Gopalakrishnan, *Oracle Wait Interface: A Practical Guide to Performance and Diagnostics & Tuning*, Oracle Press, Emeryville, 2004.

Refer to the following text for some quantitative case studies of diagnosing and resolving Oracle-based enterprise application performance and scalability problems in the context of OWI:

H. Liu, *Software Performance and Scalability: a Quantitative Approach*, John Wiley & Sons, Hoboken, 2009.

EXERCISES

- 9.1 What's the drive behind the ratio-based Oracle performance tuning methodology giving its way to wait-event-based Oracle performance tuning methodology?
- 9.2 Explain what a wait event is and what a wait chain is. How do you define a bottleneck?
- 9.3 What's the difference between a physical wait event and a logic wait event? Explain why there are so many types of wait events.
- 9.4 Why is it helpful to classify wait events into classes?
- 9.5 What's the relationship between OWI and AWR reports? Which one is the preferred Oracle performance troubleshooting approach, querying OWI related V\$ views directly or using AWR reports?
- 9.6 What is an Oracle logic read conceptually? Isn't it the purpose of caching that data should be read as much as possible from cache rather than from disks? And explain how logic reads may significantly affect the performance and scalability of an Oracle-based enterprise application.

10

Oracle Data Consistency and Concurrency

Pure mathematics is, in its way, the poetry of logical ideas.

—Albert Einstein

It's very rare that an Oracle-based application would be accessed by a single user only all the time. A large-scale Oracle database might be accessed by millions of users *concurrently*, which means that same data could be queried and/or modified by multiple users. In such a case, it's necessary to guarantee that any user will get a consistent view of the data at any point of time no matter how many users are querying and/or modifying the data. This is formally termed a *data consistency/concurrency* issue. This is a very delicate issue from performance and scalability perspectives, because concurrency could both enhance and hinder performance and scalability of an application. Concurrency helps performance and scalability because it enables more efficient use of the hardware computing resources and gets more work done for a given period of time. In the meanwhile, it may cause various types of contentions because sometimes requests from multiple users have to be serialized so that data consistency is preserved while having performance and scalability compromised. We'll see how data consistency and concurrency play out in Oracle in this chapter.

This chapter consists of the following sections:

- SELECT ... FOR UPDATE Statement
- ACID properties of Transactions
- Read Phenomena and Data Inconsistencies
- Oracle Isolation Levels
- Multi-Version Concurrency Control (MVCC) and Read Consistency
- Oracle Locks
- Oracle Lock Escalations and Conversions
- Oracle Latches
- Oracle Enqueues
- Taking Advantage of Oracle's Scalable Concurrency Model
- Case Study: a JDBC Example

Next, we begin with introducing the SELECT ... FOR UPDATE statement first rather than delve into data consistency and concurrency topics immediately. The reason is that although this less standard SQL statement is syntactically similar to a regular SELECT SQL statement, semantically, it's actually treated more like those DUI (DELETE/UPDATE/INSERT) SQL statements in terms of how locks are applied. Therefore, we need to clarify this SQL statement for those who are less familiar with it.

10.1 SELECT ... FOR UPDATE STATEMENT

When a regular SELECT query statement without FOR UPDATE clause is processed by Oracle, no locks are placed on the rows that satisfy the selection criteria. However, locks would get involved when a statement of SELECT ... FOR UPDATE SQL is processed by Oracle. Let's use an example to illustrate the case.

The syntax for the SELECT ... FOR UPDATE SQL statement in PL/SQL is:

```
CURSOR cursor_name
IS
  SELECT_statement
  FOR UPDATE [of column_list] [NOWAIT];
```

For example, with a banking account table, a SELECT ... FOR UPDATE statement may look like:

```
CURSOR account_cursor
IS
  SELECT account_ID, balance FROM ACCOUNT WHERE branch_ID = 'San Jose'
  FOR UPDATE OF balance;
```

In the above case, the SELECT ... FOR UPDATE statement requires that the rows in the cursor result set be locked to indicate the user's intention to update the values of the

`balance` column of these rows. No other user will be able to update any of these rows until you perform a `ROLLBACK` or a `COMMIT` to signify that you are done, although it's not mandatory for you to actually update any of them.

One can use the `FOR UPDATE` clause in a `SELECT` statement against multiple tables. In this case, selected rows in a table are locked only if the `FOR UPDATE` clause references a column in that table. The number of columns placed on the “OF” list of the `FOR UPDATE` clause does not change the fact that locks are placed on the entire rows that match the `SELECT` criteria expressed in the `WHERE` clause. However, be careful that if there are no columns after the `OF` keyword or there is no `WHERE` clause, then all identified rows across all tables listed in the `FROM` clause will be locked, which might limit concurrent access to the tables involved in the query inadvertently.

Finally, the keyword `NOWAIT` can be appended optionally to the `FOR UPDATE` clause to tell Oracle not to wait if no locks are available immediately. In this case, control will be returned to your PL/SQL code in which you can decide what to do next, for example, perform other work or simply wait for a period of time and retry. Without the `NOWAIT` option appended, the other process will block until the locks for the tables become available. Note that your wait will never time out unless the tables are remote. For remote tables, the Oracle initialization parameter, `DISTRIBUTED_LOCK_TIMEOUT`, is used to set the timeout limit.

Next, let's begin by looking at the ACID properties of Transactions both in general and in Oracle's context. It's necessary to understand what a database transaction means exactly, because a transaction defines the boundaries between multiple program units executed in a database engine.

10.2 ACID PROPERTIES OF TRANSACTIONS

A *transaction* is an overloaded term that has different meanings in different context. In databases, a *transaction* is defined as a single unit of work that satisfies the following ACID properties, which were defined by Jim Gray in the later 1970s (Gray and Reuter, 1992):

- Atomicity. This is an “all or nothing” rule requiring that either all operations of a transaction complete or none of it happens. If a transaction fails, a series of rollback operations are performed to undo the effects of some of the tasks in the transaction. Transaction failures may come from hardware, a system, a database, or an application. With this requirement in place, users don't have to worry about such unpredictable incidents.
- Consistency. This property requires that a transaction should never bring the database from a consistent state to an inconsistent state. This is a necessary requirement, as a database may process billions of transactions, and if an intermediate state is inconsistent, then all its work from that point on cannot be trusted.
- Isolation. This property requires that concurrent transactions must be isolated from each other so that they don't see each other's uncommitted data

until they complete. Isolation goes against performance and may even cause deadlocks.

- Durability. This property requires that once a transaction is committed, its effects are permanent. This property is maintained with Oracle's redo logs that can be used to reprocess the committed changes in case any failures occur later.

Oracle's superior transactional implementation is one of the most crucial elements making Oracle a highly performing and scalable database platform. All terms you might have heard like COMMIT, UNDO, ROLLBACK, REDO, SAVEPOINT, and so on, are all related to the concept of a transaction in Oracle. A detailed coverage of how transactions are implemented in Oracle is beyond the scope of this text, but I'd like to cover two transaction control statements here: COMMIT and ROLLBACK. This is sufficient if you are not a full time PL/SQL programmer.

There are three actions in the course of a transaction that need to be specified in a program so that Oracle would know how to react accordingly. These three actions are: `transaction BEGIN`, `transaction COMMIT`, and `transaction ROLLBACK`. The BEGIN modifier marks the beginning of a transaction, while the COMMIT and ROLLBACK modifiers mark the end of a transaction. Note that different products may mark a transaction differently. So if you are a programmer, you definitely need to check out all the conventions and concrete transaction statements on how a transaction is marked in your context or environment. Here, we cover how Oracle marks a transaction.

Here is how a transaction is marked in SQL*Plus:

- `Transaction BEGIN`. Oracle has a convention that a transaction implicitly begins with the first SQL statement that modifies data such as those DML SQL statements and the `SELECT . . . FOR UPDATE . . .` statements. This means that a user does not need to tell Oracle that a transaction begins.
- `Transaction COMMIT`. The term COMMIT means that the user intends to make all the changes made so far permanent. This is simple with SQL*Plus: just enter the "COMMIT;" command at a SQL> prompt. Note that SQL*Plus has a "SET AUTOCOMMIT" command. By default, it's set to OFF, which means that you have to manually commit by entering a "COMMIT" command. However, you can set it to ON by executing the command "SET AUTOCOMMIT ON" so that every SQL statement or a PL/SQL block will be committed automatically.
- `Transaction ROLLBACK`. While a transaction COMMIT can be considered a ROLLFORWARD, a ROLLBACK command would undo whatever has been done since your last COMMIT. A quick quiz here: if you issue a COMMIT command and a ROLLBACK command at the SQL> prompt consecutively, will the effects from both commands cancel out if they both succeed?

At the end of this chapter, I'll go through a JDBC example with you to help you understand the concept of a transaction as well as other topics covered in the remainder of this text. Next, let's explore how we can correlate certain read patterns or phenomena with resultant data inconsistencies.

10.3 READ PHENOMENA AND DATA INCONSISTENCIES

The consistency or inconsistency of a *view* of a data item depends on certain *read* phenomena that can be classified as follows:

- *Dirty Reads.* A read is a *dirty read* if the data read is actually modified but not persisted or committed yet. Let's use a banking example to illustrate what a dirty read is about. Suppose you have a checking account and a savings account. When you initiate a transfer from your checking to your savings account, there is an intermediate state that a certain amount of money has been subtracted from your checking account but has not been deposited into your savings account. Let's define this as Transaction 1 (T1). Let's say, right at this instant, your bank checks your total balance from both accounts, and let's define this as Transaction 2 (T2). The T2 transaction would determine that your total was less than what you should actually have. In this case, the less total balance read or queried by your bank is a *dirty read* from the T1's uncommitted data. This dirty read scenario is illustrated in Figure 10.1(a).
- *Non-Repeatable or Fuzzy Reads.* Let's use the same banking example above to illustrate what a non-repeatable read is about. Let's say you initiated a request to check your checking account balance. Immediately after the row containing your checking account balance is read, your bank deducted a bill pay from your checking account. If you immediately query your checking account balance again, you might find that you have less than you had a moment ago, or you might find that you still have the same balance, depending on how the two queries were executed. This is a problem only if the two queries executed consecutively are defined in the same transaction, because some isolation levels requires the old value be returned while some other isolation levels require the new value returned. Basically, a *non-repeatable read* occurs when transaction 1 queries a row, transaction 2 subsequently updates the row, and transaction 1 queries the same row again. In this case, transaction 1 queries the same row twice in a single transaction but gets inconsistent data. See Figure 10.1(b) for the scenario of a non-repeatable read.
- *Phantom Reads or Phantoms.* A phantom read is associated with a range-based query. Using the same banking example, let's say you issue a query for all the transactions that occurred between a start date and an end date. Initially, say, 10 transactions existed for that period of time. Once again, just before you execute the same query again without delay, your bank updated your account activities by inserting more transactions into that period of time. Now your query returned more rows or transactions for that period of time than a moment ago. Once again, this is a problem only if the two queries executed consecutively are defined in the same transaction, because some isolation levels requires the old value be returned while some other isolation levels require the new value returned. So this is how a *phantom read* may happen: Transaction 1 runs a range-based query, and transaction 2 adds more rows that satisfy the same range condition,

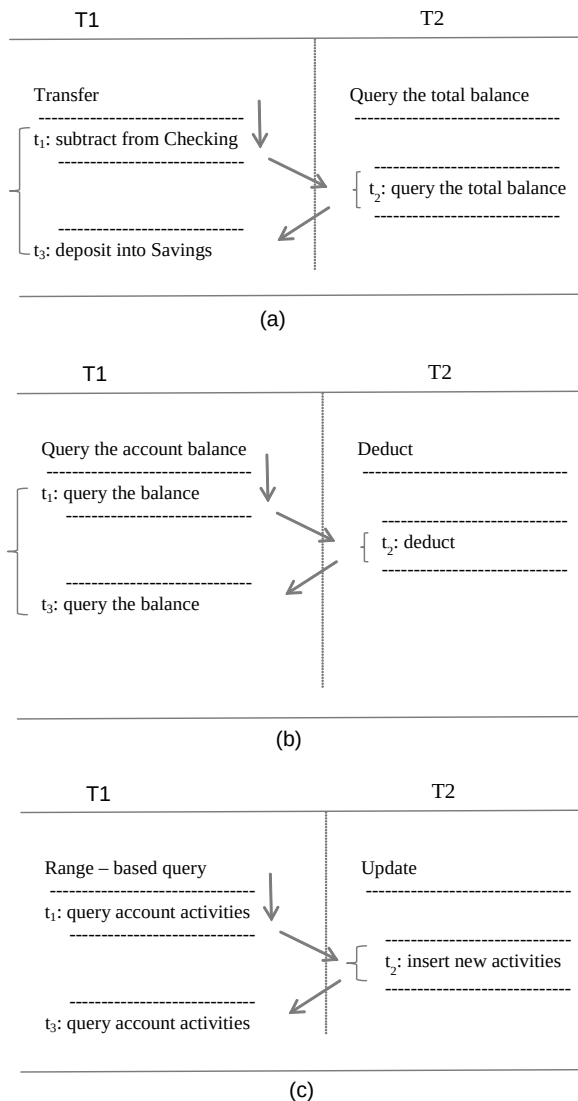


Figure 10.1 Various read phenomena: (a) a dirty read, (b) a non-repeatable read, and (c) a phantom read. Note that t_1 , t_2 , and t_3 represent three consecutive points of time while arrows represent an execution sequence between the two transactions.

and transaction 1 reruns the same range-based query and sees some more rows returned, which are called phantoms. See Figure 10.1(c) for the scenario of a phantom read.

As we see, read inconsistency is caused by allowing multiple users to access the same data concurrently from a transactional point of view. If we do not allow such kind of intermingling to occur among multiple users, then data inconsistency issue could be

Table 10.1 Effects of Isolation Levels on Data Inconsistencies Arising from Various Read Phenomena

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
Read Uncommitted	Possible	Possible	Possible
Read Committed	Not possible	Possible	Possible
Repeatable Read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

potentially prevented. This is actually what the ANSI/ISO standard SQL 92 has specified in terms of isolation levels, as is discussed next.

10.4 ORACLE ISOLATION LEVELS

Based on the read phenomena discussed in the preceding section, SQL 92 specifies four levels of isolation to prevent some or all of those read inconsistencies from happening. See Table 10.1 for the effects of those four isolation levels on the possibility of the read inconsistencies corresponding to dirty read, non-repeatable read, and phantom read.

Since SQL 92 is only a spec, it's up to a vendor to decide on what to implement. Oracle implements `READ COMMITTED` (the default isolation level) and `SERIALIZABLE` isolation levels, as well as a `READ ONLY` level that is not specified in SQL 92. With the `READ ONLY` isolation level, no modifications such as `INSERT`, `UPDATE`, and `DELETE`, and so on, are allowed except regular `SELECT` operations.

Note that dirty read gets only one SQL query involved, whereas non-repeatable read and phantom read get more than one query involved. This distinction to some extent leads to the concept of a *transaction* that is defined as a single unit of work. Therefore, we can have two differing types of read consistencies: (1) *statement-level read consistency* for dirty reads, and (2) *transaction-level read consistency* for non-repeatable and phantom reads, in which case, all of the queries from a user are lumped into one transaction. A statement-level read consistency issue becomes a transaction-level read consistency issue if a single statement is considered a transaction, which is especially true with a query that aggregates many rows of a table, for example, summing up a column of all rows of a table.

Also note that all types of read phenomena are about reading data at different points of time. If changes to data at subsequent points of time are recorded after a query begins, then the query can always check such records before returning the result, which leads to a technique called *multi-version concurrency control* (MVCC), which is discussed next.

10.5 MULTI-VERSION CONCURRENCY CONTROL (MVCC) AND READ CONSISTENCY

Oracle implemented MVCC as early as in version 3 in 1983. With MVCC, a data object is tagged with read and write timestamps. This way, an access history is

maintained for a data object. This timestamp information is maintained via `system change numbers` (SCNs), which are stored together with modified data in rollback segments. So the core concept here is the multiple “point in time” consistent views enabled by MVCC. This is Oracle’s method for guaranteeing read consistency at both the statement level and transaction level. It has the advantage that readers can read the same data and writers do not block readers, thus providing maximum read concurrency while preserving read consistency. Perhaps this is one of the most important scalability features, which differentiates Oracle from other database server products with which writers block readers.

10.6 ORACLE LOCKS

Oracle uses locking to enforce both `READ COMMITTED` and `SERIALIZABLE` isolation levels. Locking operates on database resources, which fall into two categories: (1) user objects, such as user tables and rows; and (2) system objects, such as shared memory and data dictionary rows. Oracle supports the following types of locks:

- *DML Locks (Data Locks)*. These locks protect data. They are further classified into row-level locks and table-level locks. DML row locks and table locks are also known as *TX locks* and *TM locks*, respectively.
- *DDL Locks (Dictionary Locks)*. These locks protect the structure of schema objects such as the definitions of tables and views.
- *Internal Locks and Latches*. These are automatic locks that protect internal database structures, for example, data files and data structures in memory, and so on.

One of the finest locking methods in Oracle is *row-level locking*. When a transaction needs to modify a row or multiple rows, these rows are locked automatically by Oracle. If a second transaction needs to read these locked rows, they can proceed to read under the constraint of MVCC as described previously. However, when it needs to modify these locked rows, it has to wait until the first transaction either commit or undo and release the lock. Now, there are some subtleties regarding whether the second transaction is specified with a `READ COMMITTED` isolation level or `SERIALIZABLE` isolation level, depending on the outcome of the first transaction, as is discussed below:

- *First Transaction Rolled Back*. If the first transaction rolled back, which means that no changes applied to the data, the second transaction, regardless of its isolation level (`READ COMMITTED` or `SERIALIZABLE`), can proceed to modify the previously locked rows as if the first transaction had not existed.
- *First Transaction Committed*. If the first transaction committed and released the lock, then this is what would happen:

- *Second Transaction is a Read Committed Transaction.* If the second transaction has the `READ COMMITTED` isolation level, it can proceed and modify the committed rows by the first transaction if it needs to (note that it doesn't have to if it does not have a `DUI SQL` statement in it).
- *Second Transaction is a Serializable Transaction.* If the second transaction has the `SERIALIZABLE` isolation level, it would fail and throw an error of `Cannot serialize access error`, because a serializable transaction does not allow data to be modified from when it began. This is how the `SERIALIZABLE` isolation level prevents non-repeatable and phantom reads while the `READ COMMITTED` isolation level doesn't. In this sense, we can consider serializable isolation equivalent to "not reading committed data by other transactions."

Therefore, it's clear that row-level locking may result in different outcomes for transactions with an isolation level of `READ COMMITTED` or `SERIALIZABLE`. It's very important to keep in mind all the differences between `READ COMMITTED` and `SERIALIZABLE` isolation levels when coding an application. Table 10.2 further summarizes the differences between `READ COMMITTED` and `SERIALIZABLE` isolation levels.

In contrast to row-level locking, Oracle table locks apply to an entire table. As is summarized in Table 10.3, a table lock can operate in any of the following modes:

- *The Row Share (RS) Mode.* This mode indicates that the transaction holding the table lock has locked rows in the table and *intends* to update them. It allows other transactions to acquire `RS/RX/S/SRX/X` locks and to query, insert, update, or lock rows concurrently in the same table (the modes of `RX`, `S`, `SRX`, and `X` are

Table 10.2 Differences between `READ COMMITTED` and `SERIALIZABLE` Isolation Levels

Behavior	Read Committed	Serializable
Dirty write	Not possible	Not possible
Dirty read	Not possible	Not possible
Non-repeatable read	Possible	Not possible
Phantoms	Possible	Not possible
Compliant with ANSI/ISO SQL 92	Yes	Yes
Read materialized view time	Statement	Transaction
Transaction set consistency	Statement level	Transaction level
Row-level locking	Yes	Yes
Readers block writers	No	No
Writers block writers	No	No
Different-row writers block writers	No	No
Same-row writers block writers	Yes	Yes
Waits for blocking transaction	Yes	Yes
Subject to cannot serialize access	No	Yes
Error after blocking transaction terminates	No	No
Error after blocking transaction commits	No	Yes

Table 10.3 Summary of Table Locks

SQL Statement	Table Lock Mode	RS	RX	S	SRX	X
SELECT ... FROM <i>table</i> ...	none	Y	Y	Y	Y	Y
INSERT INTO <i>table</i> ...	RX	Y	Y	N	N	N
UPDATE <i>table</i> ...	RX	Y*	Y*	N	N	N
DELETE FROM <i>table</i> ...	RX	Y*	Y*	N	N	N
SELECT ... FOR UPDATE OF ...	RX	Y*	Y*	N	N	N

*Yes, if no conflicting row locks are held by another transaction. Otherwise, waits occur.

explained next). However, it does not allow other transactions to obtain exclusive write access to the same table.

- *The Row Exclusive (RX) Mode.* This mode indicates that the lock-holding transaction has made one or more updates to the rows in the table or issued SELECT ... FOR UPDATE statements. It allows other transactions to acquire RS/RX locks and to query, insert, update, or lock rows concurrently in the same table. However, it does not allow other transactions to obtain exclusive write/read access to the same table.
- *The Share (S) Mode.* This mode indicates that other transactions can only query the table and acquire row share (RS) and share (S) table locks.
- *The Share Row Exclusive (SRX) Mode.* This mode indicates that other transactions can only query the table or acquire a share lock (S).
- *The Exclusive (X) Mode.* This is the most restrictive mode of a table lock that all other transactions can do is to query the table and no locks can be acquired. The lock-holder has exclusive write access to the table.

It's helpful to know how locks are automatically acquired for DML statements in Oracle. This is summarized in Table 10.4. It is seen that queries (SELECT without FOR UPDATE clause) do not acquire any locks and therefore do not block any other types of operations. This applies to embedded queries or subqueries in other DML statements as well. On the other hand, INSERT, UPDATE, DELETE, and SELECT ... FOR UPDATE ... statements all require exclusive row locks and row exclusive table locks.

DDL locks operate on schema objects rather than user objects. Since a DDL statement implicitly commits its transaction, DDL locks are managed automatically by Oracle and cannot be requested by a user. Besides, only individual schema objects

Table 10.4 How Locks are Automatically Acquired for DML Statements in Oracle

SQL Statement	Row Locks?	Mode of Table Lock
SELECT ... FROM <i>table</i> ...	none	none
INSERT INTO <i>table</i> ...	X	RX
UPDATE <i>table</i> ...	X	RX
DELETE FROM <i>table</i> ...	X	RX
SELECT ... FOR UPDATE OF ...	X	RX

that are to be modified or referenced are locked during DDL operations. The whole data dictionary is never locked.

DDL locks fall into the following three categories:

- *Exclusive DDL Locks.* Acquired on DROP TABLE or ALTER TABLE DDL operations. For example, an exclusive DDL lock makes sure that no table can be dropped while an ALTER TABLE DDL operation on it is in progress, or in other words, a table can be dropped only when it's not referenced by any other users.
- *Share DDL Locks.* Acquired on CREATE/AUDIT DDL operations. Share DDL locks allow multiple transactions to create objects like procedures against the same tables concurrently. However, they do not allow schema objects to be dropped or altered in contrast to Exclusive DDL locks.
- *Breakable Parse Locks.* Acquired during the parse phase of SQL statement execution and held as long as the shared SQL area for that statement remains in the shared pool. A breakable parse lock does not disallow any DDL operations and is breakable if it conflicts with any DDL operations.

Next, let's discuss lock escalations versus lock conversions, which may affect the performance and scalability of an Oracle database significantly.

10.7 LOCK ESCALATIONS VERSUS CONVERSIONS

Lock escalations refer to the implementation that a database raises a lock to a higher level of granularity when too many locks are held at a lower level. For example, such an escalation may happen from row-level locking to table-level locking if there are too many row-level locks. Apparently, lock escalation may have a huge impact on the scalability of a database-centric application, because when an entire table is locked, other users are excluded from accessing the same table. Lock escalation also increases the likelihood of deadlocks.

As is stated in Oracle's documentation, Oracle never *escalates* locks. Instead, it *converts* locks. For example, when a transaction begins with a SELECT ... FOR UPDATE statement, Oracle acquires the exclusive row locks and a row share table lock for the table. If the transaction later modifies one or more of the locked rows, the row share table lock is automatically converted to a row exclusive table lock, in which case the locked table is not fully shut off for other users to access. This conversion is possible thanks to a finer granularity of table locks built into Oracle as we discussed previously.

10.8 ORACLE LATCHES

Oracle latches are low level serialization mechanisms to protect shared data structures in the system global area (SGA). Although their occurrence and duration are out of a user's control, latches appear frequently in AWR reports, and their related metrics even appear on the top five event list very often. Therefore, a minimum understanding of what an Oracle latch is about is helpful.

To be more specific, latches are a type of locks that protect data structures in database block buffer cache or the library cache in the shared pool. Since they operate on data structures in memory, latches are acquired and released much faster than regular DML and DDL locks that we have discussed so far. How a latch is acquired is quite interesting: It's based on a "willing-to-wait" policy, which means that if a latch is not available, the requestor would sleep for a short period of time and retry later. However, there is no wait queue for latch requestors: if n requestors failed and went to sleep, then the $(n + 1)$ -th requestor will get it if this lucky requestor jumped in at the right moment just when a latch is released. Because of this uncoordinated latch acquisition policy, an option is offered similar to a `SELECT FOR UPDATE NOWAIT` that some latches can be requested in an immediate mode, and in this case, if the latch is not available, the requestor can go on to request an equivalent latch that is free rather than go to sleep and retry again endlessly.

10.9 ORACLE ENQUEUES

Enqueues are shared memory structures or locks that serialize access to database resources. A resource uniquely identifies an object that can be locked by a session or a transaction. The acquired lock on a resource is called an *enqueue*. Enqueues are defined in the `DBA_LOCK` and `DBA_LOCK_INTERNAL` data dictionary views. Here are a few examples of enqueues:

- TX—Transaction Enqueue
- TT—Temporary Table Enqueue
- SQ—Sequence Number Enqueue
- DX—Distributed Transaction Enqueue
- ...

In contrast to latches, enqueues are not operated on a "willing-to-wait" policy. Instead, enqueues wait and are dequeued on a FIFO (First-In First-Out) basis.

10.10 DEADLOCKS

A deadlock may occur when two or more users are waiting for the same data or resource locked by each other. Since Oracle does not escalate locks and does not use read locks for queries, deadlocks may potentially occur. Oracle automatically detects and breaks deadlocks through applying statement-rollback on the transaction that detected the deadlock.

Oracle recommends a few best practices at the application level to help avoid deadlocks, including:

- If a sequence of locks is required for a transaction, then consider acquiring the locks in the order of the most exclusive lock first and the least exclusive lock last.

- If multiple tables are involved in a transaction, then follow the same order to both lock and access the tables involved. For example, if both a master and a detail table are updated, then the master table should be locked first and then detail table.

In addition, most applications implement time-outs, which could also potentially help break deadlocks.

Next, we proceed to discussing how one can take advantage of Oracle's scalable concurrency model in developing Oracle-based applications in the next section.

10.11 TAKING ADVANTAGE OF ORACLE'S SCALABLE CONCURRENCY MODEL

Oracle provides `READ COMMITTED` and `SERIALIZABLE` isolation levels to ensure data consistency while maximizing concurrency. Whether one should choose `READ COMMITTED` or `SERIALIZABLE` is a trade-off between data consistency and concurrency. If you tilt more toward `READ COMMITTED` isolation level, you get more concurrency with compromised data consistency than the `SERIALIZABLE` isolation level, and vice versa. The following factors should be considered when making such a decision:

- The nature of an application. If it's a financial application, typically it may have a zero data inconsistency requirement, so certainly one should carefully consider the use of the default isolation level of `READ COMMITTED`.
- In general, the `READ COMMITTED` isolation level is more favorable if:
 - The application is more read than write intensive
 - The chances for issuing two same query consecutively are small so that non-repeatable and phantom reads are less likely to be an issue
- In general, the `SERIALIZABLE` isolation level is more favorable if:
 - There is a relative low chance that two concurrent transactions will modify the same rows
 - Long-running transactions are primarily read only
 - Transactions don't last long and update only a few rows
- On the application's side, the `READ COMMITTED` isolation level does not need to track "ORA-08177: Cannot serialize access for this transaction," whereas the `SERIALIZABLE` isolation level does need to track this error.

In addition to these standard practices as recommended by Oracle, I'd like to offer a few more based on my own real experiences. One is that one needs to properly size the hardware on which Oracle and the application will be deployed. This will provide a necessary environment for fully taking advantage of Oracle's highly performing and scalable concurrency models. The second recommendation is that

both the application and Oracle need to be optimized and tuned diligently, which is especially important for maximizing the benefits of the `READ COMMITTED` and `SERIALIZABLE` isolation levels. Also note that typically some concurrency and transactional models might have already been implemented on the application side so that the same transaction will not be serialized twice: once on the application's side and once on Oracle. Chapter 26 provides a quantitative case study to demonstrate how bulk-transaction APIs can help enhance the performance and scalability of a real world, Oracle-based application.

Before wrapping this chapter up, let's use a JDBC example presented in the next section to help corroborate what we have discussed in this chapter.

10.12 CASE STUDY: A JDBC EXAMPLE

JDBC is one of the most widely used protocols serving as a bridge between an application and a database like Oracle. In this case study, we demonstrate how transactions are managed at the JDBC level. Before showing the code snippet, let's briefly mention a few JDBC transactional specifications as follows:

- *Connection Object.* Before issuing SQLs to Oracle, a `java.sql.Connection` object must be created first. All transactional specs are set through a `Connection` object.
- *Auto-Commit Mode.* JDBC runs in auto-commit mode by default. In auto-commit mode, each SQL statement is considered a *transaction*. This auto-commit behavior can be turned off or on by calling a connection object's method of `setAutoCommit (false)` or `setAutoCommit (true)`, respectively.
- *Isolation Level.* While each database product implements its own isolation levels, JDBC manages its own isolation levels at the application level based on the isolation levels supported by an underlying database. The `Connection` interface supports all SQL 92 isolation levels as shown in Table 10.1. Any of those four isolation levels can be set through the `Connection` interface. However, what happens if the specified isolation level is not supported by the underlying database? In this case, the next more restrictive isolation level supported by the underlying database will be picked. Specific to Oracle, only `READ COMMITTED` and `SERIALIZABLE` isolation levels are supported. If you do not call the `setTransactionIsolationLevel` method through a `Connection` object, the default isolation level of `READ COMMITTED` is used. You can use the `Connection` method `getTransactionIsolation ()` to find out the isolation level set in your database. You can also check whether an isolation level is supported in your database as is shown in the following code snippet.

This sample JDBC/Oracle example uses the sample schema `HR` included in Oracle 11g R2. The `Employees` table is chosen for this example. See Figure 10.2 for all the attributes of this table. Note that this table has two very interesting attributes: `salary`

ORACLE Enterprise Manager 11g Setup
Database Control

Database Instance: ORA11GR2 > Tables >
View Table: HR.EMPLOYEES

Actions Create Like ▼

General

Name	EMPLOYEES
Schema	HR
Tablespace	EXAMPLE
Organization	Standard (Heap Organized)

Columns

	Name	Data Type	Size	Scale	Not NULL	Index
🔑	EMPLOYEE_ID	NUMBER	6		☒	
	FIRST_NAME	VARCHAR2	20		☐	
	LAST_NAME	VARCHAR2	25		☒	
✓	EMAIL	VARCHAR2	25		☒	
	PHONE_NUMBER	VARCHAR2	20		☐	
	HIRE_DATE	DATE			☒	
	JOB_ID	VARCHAR2	10		☒	
	SALARY	NUMBER	8	2	☐	
	COMMISSION_PCT	NUMBER	2	2	☐	
	MANAGER_ID	NUMBER	6		☐	
	DEPARTMENT_ID	NUMBER	4		☐	

🔑 Indicates a Primary Key column
✓ Indicates a Unique Key column
📁 Indicates a SecureFile LOB column

Figure 10.2 The Employees table of the sample schema HR associated with the JDBC example.

and `commission_pct`. In the example code, we'll try to give an employee a 5% raise in salary and a 10% commission assignment.

The code snippet of this JDBC example is shown in Listing 10.1 below. If you have a minimum understanding of Java, it should be easy to go through this simple sample and understand what it does. Here is a summary of the logic of this JDBC example:

- It first creates a Connection object and then checks the supported isolation levels in Oracle. If you want to try it out on your system, you need to replace the Oracle connect string hard-coded into it to match your environment.
- It then retrieves the salary and commission of the employee with `id = 100` using the `queryEmployees` method.
- It then calls the `updateEmployees` method, which executes the following logic:
 - Giving the employee a 5% salary raise and then calling the `commit` method. So this transaction will be materialized.

- o Assigning a 10% commission to this employee and then calling the rollback method. Because this 10% commission assignment is rolled back, it will not materialize.

You can check the output of this program following the code listing to verify the execution results of a run.

Listing 10.1 OraJDBCTx.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.DatabaseMetaData;
import oracle.jdbc.pool.OracleDataSource;
import java.util.HashMap;
import java.util.Map;

public class OraJDBCTx {
    public static void main(String args[]) throws SQLException {

        OracleDataSource ods = null;
        Connection conn = null;
        ResultSet rset = null;
        // Create DataSource and connect to oracle
        ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:@//192.168.1.103:1521/Ora11GR2");
        ods.setUser("HR");
        ods.setPassword("hr");
        conn = ods.getConnection();

        checkSupportedTransactions(conn);

        int employee_id = 0;
        float salary = 0.0f;
        float commission_pct = 0.0f;

        HashMap<String, Float> employeesSalary = new HashMap();
        System.out.println("\nstep: employee_id / salary / commission");

        try {
            // Query employee salary and commission
            rset = queryEmployees(conn);
            // loop through result set
            while (rset.next()) {
                employee_id = rset.getInt("EMPLOYEE_ID");
                salary = rset.getFloat("SALARY");
                commission_pct = rset.getFloat("COMMISSION_PCT");
            }
        }
    }
}
```

```

        System.out.println("1. before committing: " + employee_id
            + " / " + salary + " / " + commission_pct);
        employeesSalary.put(new Integer(employee_id).toString(),
            new Float(salary));
    }

    // Tx test
    updateEmployees("HR", conn, employeesSalary);
}
// Close the result set, statement, and the connection
finally {
    if (rset != null)
        rset.close();
    if (conn != null)
        conn.close();
}
}

public static ResultSet queryEmployees (Connection conn) {
    Statement stmt = null;
    ResultSet rset = null;
    try {
        stmt = conn.createStatement();
        rset = stmt.executeQuery("SELECT employee_id, salary, "
            + "commission_pct FROM employees where employee_id=100 "
            + "order by employee_id asc");
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rset;
}

public static void updateEmployees (String dbName, Connection conn,
    HashMap<String, Float> employeesSalary) throws SQLException {

    PreparedStatement updateSalary = null;
    PreparedStatement updateCommission = null;

    String updateString = "update " + dbName + ".employees"
        + " set salary = ? where employee_id = ?";

    String updateStatement = "update " + dbName + ".employees"
        + " set commission_pct = ? where employee_id = ?";

    try {
        conn.setAutoCommit(false);
        updateSalary = conn.prepareStatement(updateString);
        updateCommission = conn.prepareStatement(updateStatement);
    }

```

```

for (Map.Entry<String, Float> e : employeesSalary.entrySet()) {
    System.out.println("2. adding 5% salary raise and commit ...");
    updateSalary.setFloat(1, (e.getValue().floatValue() * 1.05f));
    updateSalary.setInt(2, Integer.decode(e.getKey()));
    updateSalary.executeUpdate();
    System.out.println("3. committing ...");
    conn.commit();
    System.out.print("4. after committed: ");
    printResultSet(queryEmployees(conn));

    System.out.println("5. assigning 10% commission and rollback");
    updateCommission.setFloat(1, 0.10f);
    updateCommission.setInt(2, Integer.decode(e.getKey()));
    updateCommission.executeUpdate();
    System.out.println("6. rolling back ...");
    conn.rollback();
    System.out.print("7. after rolled back: ");
    printResultSet(queryEmployees(conn));
}
catch (SQLException e) {
    System.out.println("commit error: " + e.getMessage());
    if (conn != null) {
        try {
            System.err.println("The Tx is being rolled back");
            conn.rollback();
        } catch (SQLException excep) {
            System.out.println("rollback error: " + e.getMessage());
        }
    }
}
finally {
    if (updateSalary != null) {
        updateSalary.close();
    }
    if (updateCommission != null) {
        updateCommission.close();
    }
    conn.setAutoCommit(true);
}
}

```

```

public static void printResultSet(ResultSet rset) {

    try {
        while (rset.next()) {
            int employee_id = rset.getInt("EMPLOYEE_ID");
            float salary = rset.getFloat("SALARY");
            float commission_pct = rset.getFloat("COMMISSION_PCT");

```

```

        System.out.println(employee_id + " / " + salary + " / "
            + commission_pct);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}

public static void checkSupportedTransactions(Connection conn) {

    try {
        DatabaseMetaData dbMetaData = conn.getMetaData();
        System.out.println("Oracle ISOLATION LEVEL in use: "
            + conn.getTransactionIsolation());
        if (dbMetaData.supportsTransactionIsolationLevel
            (Connection.TRANSACTION_READ_UNCOMMITTED)) {
            System.out.println("Isolation Level "
                + "TRANSACTION_READ_UNCOMMITTED is supported.");
        } else {
            System.out.println("Isolation Level "
                + "TRANSACTION_READ_UNCOMMITTED is not supported.");
        }
    }
    if (dbMetaData.supportsTransactionIsolationLevel
        Connection.TRANSACTION_READ_COMMITTED)) {
        System.out.println("Isolation Level "
            + "TRANSACTION_READ_COMMITTED is supported.");
    } else {
        System.out.println("Isolation Level "
            + "TRANSACTION_READ_COMMITTED is supported.");
    }
    if (dbMetaData.supportsTransactionIsolationLevel
        (Connection.TRANSACTION_REPEATABLE_READ)) {
        System.out.println("Isolation Level "
            + "TRANSACTION_REPEATABLE_READ is supported.");
    } else {
        System.out.println("Isolation Level "
            + "TRANSACTION_REPEATABLE_READ is supported.");
    }
    if (dbMetaData.supportsTransactionIsolationLevel
        (Connection.TRANSACTION_SERIALIZABLE)) {
        System.out.println("Isolation Level "
            + "TRANSACTION_SERIALIZABLE is supported.");
    } else {
        System.out.println("Isolation Level "
            + "TRANSACTION_SERIALIZABLE is supported.");
    }
}

```

```

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

The output result of a run of the above program is given below. Note that the Oracle Isolation level was 2 or READ COMMITTED. This is consistent with what we described earlier in this chapter. However, it's surprising that it even returned a result showing that Oracle supports REPEATABLE_READ, which contradicts the Oracle documentation. You might want to check this example program out and see what you would get (make sure you use Oracle JDBC driver `ojdbc6.jar` as was used with this example).

```

Oracle ISOLATION LEVEL in use: 2
Isolation Level TRANSACTION_READ_UNCOMMITTED is not supported.
Isolation Level TRANSACTION_READ_COMMITTED is supported.
Isolation Level TRANSACTION_REPEATABLE_READ is supported.
Isolation Level TRANSACTION_SERIALIZABLE is supported.

```

```

step: employee_id / salary / commission
1. before committing: 100 / 70206.24 / 0.0
2. adding 5% salary raise and commit...
3. committing...
4. after committed: 100 / 73716.56 / 0.0
5. assigning 10% commission and rollback
6. rolling back...
7. after rolled back: 100 / 73716.56 / 0.0

```

10.13 SUMMARY

This chapter discussed Oracle's data consistency and concurrency models from performance and scalability perspectives. Various read phenomena were associated with resultant data inconsistencies. Then we introduced how multi-version concurrency control can help enforce read consistency while providing better concurrency with the benefits of readers and writers do not block each other. We also covered the three Oracle isolation levels: READ COMMITTED, SERIALIZABLE, and READ ONLY, with READ COMMITTED as the default isolation level. Note that if you use the READ COMMITTED isolation level with Oracle, non-repeatable and phantom reads are not prevented.

We then discussed Oracle's row-level locking, which gives the maximum concurrency/scalability while providing an adequate isolation level. Various table lock modes were covered as well. Oracle's table locking does not simply lock up all the rows of a table. Instead, it's more of a protection mechanism to prevent tables from being deleted or altered while their rows are being modified. Besides, Oracle

has many table lock modes that allow table locks to be converted from one restrictive level to the next rather than escalating from row locking to table locking as with other databases.

We have also explained the concepts of latches and enqueues. These are Oracle internal locks and users have no control of them. However, it's very important to understand conceptually what they are, because the statistical metrics associated with them appear frequently in one of the major Oracle performance and scalability troubleshooting tools—Automatic Workload Repository (AWR), which is the main subject of the next chapter. Many concurrency issues are reflected as contentions or wait events in AWR reports, so this is a natural transition to the next chapter of discussing AWR after we have covered the Oracle concurrency models in this chapter.

RECOMMENDED READING

The following two texts explain Oracle data consistency and concurrency well:

T. Kyte, *Expert Oracle Database Architecture*, A Press, New York, 2010.

T. Kyte, *Expert One on One Oracle*, A Press, New York, 2001.

To understand transactions in the context of databases and applications, refer to the following classic text:

J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), Morgan Kaufmann, 1st edition, San Francisco, 1992.

To learn more about JDBC, refer to:

<http://download.oracle.com/javase/tutorial/jdbc/basics/index.html>

You can also consult Chapter 13 of the following Oracle Concept document for more information on Oracle concurrency and transaction models:

Oracle Corp, *Oracle Database Concepts*, 11g Release 1 (11.1) B28318-05 (556 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm

EXERCISES

- 10.1** Pick a sample schema and try out various transaction settings introduced in Section 10.2 at a SQL> prompt.
- 10.2** Describe all read phenomena that may cause data inconsistencies. What isolation levels are proposed in SQL 92 for combating the resultant data inconsistencies? Which ones are implemented by Oracle?

- 10.3** Explain how data consistency and concurrency requirements may go against each other.
- 10.4** Explain how Oracle's row-level locking and table level locking work together to ensure data consistency while providing maximum concurrency/scalability.
- 10.5** What's the difference between a lock escalation and a lock conversion?
- 10.6** Explain the concepts of Oracle's latches and enqueuees.
- 10.7** Explain the ACID properties of a database transaction and what techniques Oracle uses to guarantee these properties with Oracle databases.
- 10.8** Discuss scenarios where the `READ COMMITTED` or the `SERIALIZABLE` isolation level is more favorable.
- 10.9** How does an Oracle `SELECT . . . FOR UPDATE` statement work? How locks are applied with this category of SQL statements?

11

Anatomy of an Oracle Automatic Workload Repository (AWR) Report

It takes a very unusual mind to undertake the analysis of the obvious.

—Alfred North Whitehead

Oracle has taken diagnosing and analyzing performance and scalability issues seriously since the very earliest versions. The relevant tools have gone through three generations, from the UTLBSTAT/UTLESTAT utilities that had been in use since the earliest version up to Oracle 8, to STATSPACK in Oracle 9i, and to AWR in Oracle 10g and 11g. All those tools have been based on the same framework of V\$ tables and snapshots.

As described in Chapter 5, Oracle stores statistic information about the activity of a database in V\$ tables or views in memory. Then, when a snapshot is taken, the accumulative statistic information about the activity of a database up to that point is extracted and inserted into a series of *stats* tables generated with the corresponding scripts. By comparing two snapshots stored in the stats tables, those tools can generate reports based on the performance metrics such as elapsed times and system resource

consumptions, which are associated with various wait events and/or high-load SQL statements. The availability of such information makes it possible to both proactively monitor the performance of a database and reactively investigate a performance problem.

Historically, the UTLBSTAT and UTLETSAT tools were called BSTAT-ESTAT utilities, with the letter “B” in BSTAT standing for “begin” and the letter “E” in ESTAT standing for “end.” The beginning snapshot was taken with the BSTAT utility, while the ending snapshot was taken with the ESTAT utility. Then the ESTAT utility was used to generate a report. In contrast, both STATSPACK and AWR use one command to take snapshots, while having a separate script to generate the report. Each snapshot has a snap_id, and one can generate a report with any two snapshots as long as they were taken in the same period of time since the last time the database was started up. Since the state of a database maintained in memory is reset when a database is restarted, one cannot generate an AWR report with two snapshots taken in two different running periods of a database.

An Oracle performance report generated with the original UTLBSTAT and UTLETSAT tools was divided into the following sections:

- System Wide Stat Totals
- File I/O Stats
- Latch Statistics
- Rollback Segment Stats
- Dictionary Cache Stats
- Init.ora Parameters

STATSPACK and AWR reports include more information in a lot more sections than those generated with the original UTLBSTAT and UTLETSAT tools as shown above. The remainder of this chapter focuses only on exploring what information is contained in an AWR report and how we can use it to diagnose and improve Oracle performance and scalability, as STATSPACK has somewhat become outdated.

11.1 IMPORTANCE OF PERFORMANCE STATISTICS

Oracle uses a large set of statistics to quantitatively characterize the performance of an Oracle database from as many aspects as possible. The best way to understand what those performance statistics are about is to look at what columns are contained in the three V\$ tables of V\$SYSSTAT, V\$SESSTAT, and V\$STATNAME. These three V\$ tables represent the system level statistics, session level statistics, and the names of all

statistics, respectively. The following output from the DESCRIBE command for each V\$ table tells all:

SQL> DESC V\$SYSSTAT

Name	Null?	Type
-----	-----	-----
STATISTIC#		NUMBER
NAME		VARCHAR2 (64)
CLASS		NUMBER
VALUE		NUMBER
STAT_ID		NUMBER

SQL> DESC V\$SESSTAT

Name	Null?	Type
-----	-----	-----
SID		NUMBER
STATISTIC#		NUMBER
VALUE		NUMBER

SQL> DESC V\$STATNAME

Name	Null?	Type
-----	-----	-----
STATISTIC#		NUMBER
NAME		VARCHAR2 (64)
CLASS		NUMBER
STAT_ID		NUMBER

As is seen from the output for the system level statistic V\$ view of V\$SYSSTAT, each statistic is defined with the following attributes:

- *STATISTIC #*. This is the statistic number of the statistic.
- *NAME*. This is the name of the statistic.
- *CLASS*. This is a number representing the class that the statistic belongs to. The following independent class numbers are supported as of Oracle 11g:
 - o 1—User
 - o 2—Redo
 - o 4—Enqueue
 - o 8—Cache
 - o 16—OS
 - o 32—Real Application Cluster
 - o 64—SQL
 - o 128—Debug

An interesting fact is that these class numbers are additive. For example, by adding 8 and 64, a new class number of 72 is derived, which represents a new class of SQL cache.

- *VALUE*. This is the value of the statistic.
- *STAT_ID*. This is the identifier of the statistic.

The session level statistic is defined with three attributes as indicated above: *SID*, *STATISTIC#* and *VALUE*. This set of statistics is maintained on a per session basis. It is valid only for the session currently connected, similar to the set of system statistics that is valid only for the current running period since its startup. The *V\$STATNAME* view defines the name of the statistic with given *STATISTIC#*, *CLASS* and *STAT_ID*. For a complete list of all performance statistics defined in *V\$SYSSTAT*, see Appendix E.

Oracle performance statistics serve as the basis for monitoring and diagnosing the performance of an Oracle database. It's important to keep in mind that Oracle performance statistics are *cumulative*, namely, counted since the startup of an instance. Whenever a database is restarted, the performance statistics are reset. In addition, it's the changes in statistics or *delta values* over a period of time that are most interesting when diagnosing a performance problem. Therefore, it's critical to determine the time period when diagnosing a performance issue. A too large time period may average out the symptoms of a performance issue, while a too small time period may not capture the relevant performance factors. In general, the time period should be limited to from several minutes to no more than one hour. Ideally, one should try to correlate the performance statistics with the driving workloads, which can be characterized as normal, peak and somewhere in-between. Proper snapshot period can be determined accordingly.

As discussed above, the start and end points of a time period are determined by the timestamps of the snapshots taken either automatically or manually. By default, Oracle initiates a snapshot around the transition of an hour. However, in a test environment, it's necessary to take snapshots manually to match the start and end of a test. This is especially necessary for performance regression tests of a product for which constant changes are made over time.

Having explained what Oracle performance statistics are about and the importance of looking at those statistics for a properly determined time period, we are ready to anatomize an AWR report that I generated and used for solving an Oracle performance and scalability issue with a real product. For your reference, the commands for taking snapshots and generating an AWR report are given in Appendix B at the end of this text.

Before we start, I'd like to mention that the performance statistics reflected in an AWR report are organized by category. I have to point out that an AWR report is lengthy and you are encouraged to take a cursory look only at this time. In reality, according to my experience, only about 10% or less of an AWR report is relevant to what I was looking for in resolving an Oracle performance issue associated with the

product being diagnosed or tested. But it's hard to gauge which part of an AWR report would be relevant to a specific reader, so I decided to give a full coverage here, with some sections trimmed to save space.

The all-inclusive categories of an AWR report are listed below (for comparison purposes, those typed in boldface were already available from the original BSTAT/ESTAT utilities, as described earlier):

- Report Header
- Report Summary
- Main Report
- Wait Events Statistics
- SQL Statistics
- **Instance Activity Statistics**
- **IO Stats**
- Buffer Pool Statistics
- Advisory Statistics
- Wait Statistics
- **UNDO statistics**
- **Latch Statistics**
- Segment Statistics
- **Dictionary Cache Stats**
- Memory Statistics
- Streams Statistics
- Resource Limit Stats
- **Init.ORA Parameters**

To assist you going through this lengthy (and perhaps boring) report more smoothly, I'd like to mention that the entire report has been divided into sections following its original order and each section is preceded with corresponding textual descriptions. This is particularly beneficial for those who may have not gotten a chance or an Oracle setup to generate an AWR report themselves. This is an opportunity for them to see what a real AWR report looks like. However, to keep it as concise as possible and as authentic as possible in the meanwhile, I have omitted those parts that take too much space without sacrificing the completeness of this report. Let's start with the header section of this specific AWR report out of one of my real-product experiences next.

11.2 AWR REPORT HEADER

An AWR report begins with a header section. As shown below, the header section identifies the DB name, DB Id, Instance, Instance Number, Oracle

Release version, RAC configuration and host information. In addition, it identifies the `snap id` and time for the `begin` and `end` snaps, elapsed time and DB Time. With this specific example, the time period was 59.74 minutes or about one hour.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ObStore	2563315763	ObStore	1	10.2.0.1.0	NO	snt2k-1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	231	27-Jun-07 14:00:43	80	3.3		
End Snap:	232	27-Jun-07 15:00:27	80	3.3		
Elapsed:		59.74 (mins)				
DB Time:		614.64 (mins)				

11.3 REPORT SUMMARY

The Report Summary section is divided into the following subcategories:

- Cache Sizes
- Load Profile
- Instance Efficiency Percentages
- Shared Pool Statistics
- Top Five Timed Events

These subcategories are described next.

11.3.1 Cache Sizes

In Section 6.1 of this text, the total SGA was decomposed into various sub-areas. The Cache Sizes part of an AWR report provides statistics about some of those sub-areas. The most performance sensitive sub-areas are summarized as follows:

- *Buffer Cache*. This corresponds to the database block buffer cache, which is the largest sub-area. The report shows the buffer cache size at the beginning and ending of the snapshots. In this example, the initial buffer cache size was 736 MB, which ended at 748 MB. This difference is small—only 16%. This is an indication that the buffer cache was properly sized, which can be confirmed further with the buffer hit ratios provided later.

- *Shared Pool Size.* This is the second largest sub-area in the SGA. It started with 260 MB and ended with 248 MB. Note that the change was -5% only, which once again confirmed that the shared pool was properly sized as well.
- *Log Buffer.* This corresponds to the redo log buffer sub-area. It's reported as a fixed amount of about 10 MB, in contrast to the dynamically varied sub-areas of the buffer cache and shared pool size as a result of the Automatic Shared Managed Memory feature of Oracle. Note that 10 MB is a proper size of the log buffer in most cases, which can be verified further with the log-file-related wait events.
- *Standard Block Size.* Similar to the log buffer, this is a fixed parameter that is OS-dependent. The 8K block size is optimal for most applications on Windows.

Cache Sizes

	Begin	End		
Buffer Cache:	736M	748M	Std Block Size:	8K
Shared Pool Size:	260M	248M	Log Buffer:	10,344K

11.3.2 Load Profile

The Load Profile section of an AWR report provides information on the load intensity over the period of the snapshots from multiple perspectives, and in two different metrics: *per second* and *per transaction*. Each of the load type is explained below:

- *Redo Size.* This metric measures the amount of redo log data generated per second or per transaction. In this example, about 235 kB redo log data was generated per second, or about 5.8 kB redo log data generated per transaction on average.
- *Logical Reads, Physical Reads, and Physical Writes.* These three metrics are grouped together here since they are correlated. As was explained previously, the terms *logical* and *physical* imply whether the operations were from memory or disk. Since operations on memory are orders of magnitude faster than those on disk, the values of these metrics vary by orders of magnitude as well. Whether the numbers associated with these metrics are normal should be assessed with the top SQLs and I/O statistics to be discussed later. However, one can make a quick conclusion that disk I/O is not an issue here, as most I/O devices can handle up to 1000 IOs per second, which is far above the 0.07 physical reads per second and about 27 physical writes per second shown in this example. However, the high

number of logical reads of about 235 thousands per second is worrisome, as that is about one logical read every 2.5 microseconds. We will discuss more about this later.

- *Block Changes.* This metric measures the number of data storage blocks changed per second or per transaction. A high number of over 800 block changes per second in this example implies that the workload must have been more write-intensive than read-intensive, as can be verified with the physical read and write metrics.
- *User Calls and Logons.* These two metrics measure the load intensity from the Oracle user perspective. The high ratio of the user calls to logons implies that the application has been programmed efficiently from this perspective in the sense that more user work was done after a user logged on, which is good because logon takes time for most applications. Note that the value of zero for logons is a rounded one, which does not mean that no user logon occurred.
- *Parses and Hard Parses.* As will be explained later, a hard parse of a SQL statement is orders of magnitude slower than a soft parse. Therefore, a small percentage of hard parses out of the total number of parses is favorable from the performance and scalability perspectives.
- *Executes and Transactions.* These two metrics are different but correlated. An Oracle transaction consists of multiple executions of commands or SQL statements; therefore, the ratio of the number of executes to the number of transactions measures relatively the complexity of a transaction. In this example, about nine executes occurred for each transaction on average, which implies a somewhat simpler or moderate composition of a transaction for the application under test.
- *Sorts.* This metric measures the number of sort operations per second or per transaction. Too many sorts within a transaction may warrant recoding of the data logic or SQLs. In this example, less than one sort per transaction is a good indication that both data logic and SQLs were composed properly.
- *% Blocks Changed per Read.* This metric measures potential contention between read and write operations. The smaller the percentage, the better from the performance and scalability perspective.
- *Rollback Transactions %.* This metric measures the transaction failure rate. A zero value is good from all perspectives.
- *Recursive Call %.* This is the flip side of non-recursive calls %. To some extent, it represents the “convolutedness” of a SQL statement execution, and therefore, the smaller, the better.
- *Rows per Sort.* This metric measures the number of rows sorted per sort operation on average. A closer to unity value indicates that the application under test was not sort-intensive.

Load Profile

	Per Second	Per Transaction		
Redo size:	234,605.10	5,769.21		
Logical reads:	392,174.51	9,644.02		
Block changes:	803.90	19.77		
Physical reads:	0.07	0.00		
Physical writes:	26.98	0.66		
User calls:	795.17	19.55		
Parses:	363.27	8.93		
Hard parses:	0.12	0.00		
Sorts:	33.92	0.83		
Logons:	0.00	0.00		
Executes:	364.22	8.96		
Transactions:	40.67			
% Blocks changed per Read:		0.20	Recursive Call %:	7.91
Rollback per transaction %:		0.00	Rows per Sort:	1.03

11.3.3 Instance Efficiency Percentages (Target 100%)

This section provides statistics about how efficiently an Oracle instance operates under the load. The best scenario is that every metric percentage-wise hits 100%. For convenience, some of the metrics are grouped as follows:

- *Buffer Nowait% and Buffer Hit%.* These two metrics measure the efficiency of the database buffer cache. No wait with zero misses is the best one can hope for.
- *Redo NoWait%.* This metric measures the efficiency of the redo log buffer. Once again, no wait means that nothing clogged there.
- *In-Memory Sort%.* 100% sort in memory indicates the highest possible efficiency for sort operations with zero disk accesses.
- *Library Hit%.* The library cache is another sub-area in an SGA that stores SQL statements. A high library hit percentage implies that the SQL statements were already stored in the cache and thus less likelihood for reparsing.
- *Soft Parse%.* Once again, a high soft parse percentage implies less, more expensive hard parses incurred.
- *Execute to Parse %.* If this percentage is large, it implies that less parsing or reparsing activities occurred for each SQL executed whether hard or soft parses. However, it may not be too bad if a SQL statement is soft-parsed, so a low percentage may not necessarily mean something fatal.

- *Parse CPU to Parse Elapsed %*. This metric measures the percentage of the CPU time over a parsing cycle. A value of 100% implies that no wait occurred.
- *% Non-Parse CPU*. The flip side of this metric is the time spent on parsing percentage-wise. Since parsing is some kind of overhead, the smaller the amount of time spent on parsing, the better performance-wise.

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00	Redo NoWait %:	100.00
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	99.92	Soft Parse %:	99.97
Execute to Parse %:	0.26	Latch Hit %:	99.12
Parse CPU to Parse Elapsed %:	93.44	% Non-Parse CPU:	98.88

11.3.4 Shared Pool Statistics

This section provides information about the shared pool usage percentage-wise. Since this sub-area of an SGA is automatically managed, an initial usage of about 85% implies that it started with a well-sized amount, and the end usage of about 91% implies that the shared pool was stable and well sized throughout the entire period of the test.

The metric of % SQL with executions >1 measures the reusability of SQLs, whereas the metric of % Memory for SQL with executions >1 is similar to the first metric of Memory Usage % except that it was averaged over the SQLs that were executed more than once only. Since the percentage of SQLs executed more than once was high, the value of the third metric is comparable to that of the first metric.

Shared Pool Statistics

	Begin	End
Memory Usage %:	85.15	90.94
% SQL with executions>1:	88.13	84.14
% Memory for SQL w/exec>1:	90.65	86.03

11.3.5 Top Five Timed Events

This is the section that should be jumped to first when analyzing an AWR report. From here, one could get an immediate glimpse of what might be causing the database to slow down. And if you see a skewed, very high value of % Total Call Time for a metric, that can be considered the potential number one bottleneck. And from there one can

drill down further to find out the root cause(s). This also is where a performance tuning cycle begins:

1. You identify the top number one event
2. Do your drill-downs
3. Make changes accordingly
4. Rerun your test and measure whether the performance has been improved or unchanged or even worsened
5. And then confirm or do your further analysis by looking at the new top five timed events from the newly taken AWR report

In this example, a very high value of over 96% total call time for the CPU time metric clearly shows that the database server might have been driven busy during the period of the test measured. This is undesirable, but it actually makes Oracle performance tuning easier as it implies that possibly only one factor was responsible. Those metrics that have smaller percentage values mean that the factors associated with them were not responsible for the performance issue under investigation.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		35,539		96.4	
log file parallel write	87,410	1,004	11	2.7	System I/O
db file parallel write	7,551	89	12	.2	System I/O
Streams AQ: enqueue blocked on low memory	1	80	80,354	.2	Configuration
SQL*Net more data to client	303,635	35	0	.1	Network

Next, we jump to the Main Report section, which gives concrete clues associated with the top events listed above.

11.4 MAIN REPORT

The Main Report section summarizes database activities by category with a series of clickable links as shown below. The first link, Report Summary, points back to the sections we have just discussed, whereas each subsequent link points forward to a separate category such as Wait Events Statistics, SQL Statistics, and so on. In the remainder of this chapter, each category is examined to help establish a comprehensive, consistent view of the performance of an Oracle database under load, which is

essential for effective performance bottleneck analysis. We start with the Wait Events Statistics category in the next section.

Main Report

- [Report Summary](#)
- [Wait Events Statistics](#)
- [SQL Statistics](#)
- [Instance Activity Statistics](#)
- [IO Stats](#)
- [Buffer Pool Statistics](#)
- [Advisory Statistics](#)
- [Wait Statistics](#)
- [Undo Statistics](#)
- [Latch Statistics](#)
- [Segment Statistics](#)
- [Dictionary Cache Statistics](#)
- [Library Cache Statistics](#)
- [Memory Statistics](#)
- [Streams Statistics](#)
- [Resource Limit Statistics](#)
- [init.ora Parameters](#)

[Back to Top](#)

11.5 WAIT EVENTS STATISTICS

The Wait Events Statistics section is further categorized logically as shown below. Each sub-category is discussed next.

Wait Events Statistics

- [Time Model Statistics](#)
- [Wait Class](#)
- [Wait Events](#)
- [Background Wait Events](#)
- [Operating System Statistics](#)
- [Service Statistics](#)
- [Service Wait Class Stats](#)

[Back to Top](#)

11.5.1 Time Model Statistics

This part of an AWR report gives you the elapsed time information on each type of database activity, sorted in descendent order in absolute seconds and in percentages. This way, it's easy to identify where the database spent most of its time in processing SQLs. Most items are superfluous, but all the information is there for your consumption.

In this example, the total DB time is 36,878.65 seconds, out of which 96.37% or 35,538.91 seconds were DB CPU time. Out of the total DB CPU time, 34,509.38 seconds were SQL execute elapsed time. Note that the ultimate objective is to find out why so much DB CPU time occurred; therefore, one should continue to drill down further in this report.

Time Model Statistics

- Total time in database user-calls (DB Time) : 36878.7s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
DB CPU	35,538.91	96.37
sql execute elapsed time	34,509.38	93.58
parse time elapsed	439.28	1.19
hard parse elapsed time	10.06	0.03
hard parse (sharing criteria) elapsed time	0.19	0.00
PL/SQL compilation elapsed time	0.19	0.00
hard parse (bind mismatch) elapsed time	0.18	0.00
connection management call elapsed time	0.12	0.00
sequence load elapsed time	0.08	0.00
repeated bind elapsed time	0.01	0.00
PL/SQL execution elapsed time	0.00	0.00
DB time	36,878.65	
background elapsed time	1,475.24	
background cpu time	88.65	

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.2 Wait Class

This section lists wait classes in terms of the total number of waits, % time-outs, total wait time, average wait time, and waits per transaction. In this example, one can see that there were no dominant wait classes relative to the amount of the total DB time. One can be assured that there were no bottlenecks associated with hardware resources such as disk and network or configuration issues or contention, and so on. Such information can be used to thwart such blind claims like faster disks or faster networks would be required.

Wait Class

- s - second
- cs - centisecond - 100th of a second
- ms - millisecond - 1000th of a second
- us - microsecond - 1000000th of a second
- ordered by wait time desc, waits desc

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
System I/O	97,905	0.00	1,120	11	0.67
Configuration	198	0.51	83	418	0.00
Network	2,264,485	0.00	45	0	15.54
Concurrency	24,918	0.04	28	1	0.17
User I/O	118,468	0.00	2	0	0.81
Other	4,163	56.64	2	0	0.03
Commit	6	0.00	0	9	0.00

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.3 Wait Events

This section gives more information about the number of events from each wait class. Once again, it helps eliminate many factors as potential bottlenecks that limit the performance of the application under test. Although it's lengthy, it's useful in helping understand what an elaborate mechanism Oracle has built in to facilitate troubleshooting Oracle performance issues.

Wait Events

- s - second
- cs - centisecond - 100th of a second

- ms - millisecond - 1000th of a second
- us - microsecond - 1000000th of a second
- ordered by wait time desc, waits desc (idle events last)

Event	Waits	%Time - outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file parallel write	87,410	0.00	1,004	11	0.60
db file parallel write	7,551	0.00	89	12	0.05
Streams AQ: enqueue blocked on low memory	1	100.00	80	80354	0.00
SQL*Net more data to client	303,635	0.00	35	0	2.08
control file parallel write	1,609	0.00	27	17	0.01
buffer busy waits	16,586	0.05	17	1	0.11
library cache pin	181	0.00	9	52	0.00
SQL*Net message to client	1,838,866	0.00	5	0	12.62
SQL*Net more data from client	121,984	0.00	5	0	0.84
log file switch completion	32	0.00	2	72	0.00
db file sequential read	208	0.00	2	7	0.00
kksfbc child completion	17	100.00	1	54	0.00
latch: cache buffers chains	7,673	0.00	1	0	0.05
enq: TX - index contention	444	0.00	1	1	0.00
control file sequential read	1,327	0.00	1	0	0.01
direct path write	118,208	0.00	0	0	0.81
os thread startup	2	0.00	0	152	0.00
enq: TX - contention	115	0.00	0	3	0.00
latch free	1	0.00	0	169	0.00
enq: HW - contention	164	0.00	0	1	0.00
LGWR wait for redo copy	1,648	0.00	0	0	0.01
direct path read	50	0.00	0	1	0.00
log file sync	6	0.00	0	9	0.00
latch: shared pool	13	0.00	0	4	0.00
buffer deadlock	2,348	99.70	0	0	0.02
latch: library cache	9	0.00	0	4	0.00
SGA: allocation forcing component growth	1	0.00	0	18	0.00
log file single write	4	0.00	0	4	0.00
enq: FB - contention	33	0.00	0	0	0.00
db file scattered read	2	0.00	0	5	0.00
log file sequential read	4	0.00	0	2	0.00
enq: SQ - contention	1	0.00	0	2	0.00
cursor: mutex X	7	0.00	0	0	0.00
cursor: mutex S	3	0.00	0	0	0.00
SQL*Net message from client	1,838,862	0.00	125,056	68	12.62
Streams AQ: qmn slave idle wait	397	0.00	6,949	17505	0.00
Streams AQ: waiting for time management or cleanup tasks	1	100.00	3,971	3970565	0.00
Streams AQ: qmn coordinator idle wait	263	48.67	3,475	13212	0.00
class slave wait	2	100.00	10	4889	0.00

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.4 Background Wait Events

This section helps evaluate whether the background processes had caused significant overheads. In most cases, it should not, as Oracle itself has been quite fine-tuned to run various management types of tasks efficiently. Unless you want to study the intrinsic performance of Oracle itself, there is no need to spend too much time here.

Background Wait Events

- ordered by wait time desc, waits desc (idle events last)

Event	Waits	%Time - outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file parallel write	87,408	0.00	1,004	11	0.60
db file parallel write	7,551	0.00	89	12	0.05
Streams AQ: enqueue blocked on low memory	1	100.00	80	80354	0.00
control file parallel write	1,609	0.00	27	17	0.01
os thread startup	2	0.00	0	152	0.00
control file sequential read	646	0.00	0	0	0.00
events in waitclass Other	1,649	0.00	0	0	0.01
db file sequential read	15	0.00	0	6	0.00
direct path write	20	0.00	0	4	0.00
direct path read	12	0.00	0	2	0.00
log file single write	4	0.00	0	4	0.00
log file sequential read	4	0.00	0	2	0.00
rdbms ipc message	75,983	18.98	38,910	512	0.52
Streams AQ: qmn slave idle wait	397	0.00	6,949	17505	0.00
Streams AQ: waiting for time management or cleanup tasks	1	100.00	3,971	3970565	0.00
pmon timer	1,197	100.00	3,496	2921	0.01
Streams AQ: qmn coordinator idle wait	263	48.67	3,475	13212	0.00
smon timer	11	100.00	3,036	275966	0.00

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.5 Operating System Statistics

This section helps probe how the Operating System had performed over the period of time the load persisted. In general, OS should be quite optimized, and one should just skip this section.

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	122,641
AVG_IDLE_TIME	235,696
AVG_IOWAIT_TIME	0
AVG_SYS_TIME	3,509
AVG_USER_TIME	119,023
BUSY_TIME	3,928,434
IDLE_TIME	7,546,074
IOWAIT_TIME	0
SYS_TIME	116,166
USER_TIME	3,812,268
LOAD	1
OS_CPU_WAIT_TIME	10,200
RSRC_MGR_CPU_WAIT_TIME	0
VM_IN_BYTES	0
VM_OUT_BYTES	0
PHYSICAL_MEMORY_BYTES	17,099,644,928
NUM_CPUS	32

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.6 Service Statistics

This section gives information on which users or services consumed the database resources most. Note in this example that the application incurred over 1.4 billion logical reads versus only 204 physical reads. It's obvious that the application had spent most of its DB CPU time fetching data from the database buffer cache. Although a logical read is orders of magnitude faster than a physical read, an excessive number of logical reads can sink the database into fetching data from the buffer cache endlessly without doing anything else. This is the root cause of the database problem for this particular test, as will be discussed later.

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
APP	36,883.90	35,544.10	204	1,405,812,669
SYS\$USERS	0.80	0.70	13	692
SYS\$BACKGROUND	0.00	0.00	39	20,526

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.5.7 Service Wait Class Stats

This section breaks down the total wait time incurred by each service by wait classes such as User IO, Concurrency, Admin, and Network. The values of all metrics are insignificant in this example as we already knew from the previous section that the DB CPUs were driven busy by excessive logical reads.

Service Wait Class Stats

- Wait Class info for services in the Service Statistics section.
- Total Waits and Time Waited displayed for the following wait classes: User I/O, Concurrency, Administrative, Network
- Time Waited (Wt Time) in centisecond (100th of a second)

Service Name	User I/O Total Wts	User I/O Wt Time	Concurcy Total Wts	Concurcy Wt Time	Admin Total Wts	Admin Wt Time	Network Total Wts	Network Wt Time
APP	118390	164	24917	2773	0	0	2264458	4533
SYS\$USERS	11	7	0	0	0	0	55	0
SYS\$BACKGROUND	66	28	2	31	0	0	0	0

[Back to Wait Events Statistics](#)

[Back to Top](#)

11.6 SQL STATISTICS

This section lists top SQLs from all major perspectives, for example, by Elapsed Time, by CPU time, by Gets (buffer gets or logical reads), and so on. This is where one can identify hot SQLs that were responsible for the poor performance of an Oracle-based application.

SQL Statistics

- [SQL ordered by Elapsed Time](#)
- [SQL ordered by CPU Time](#)
- [SQL ordered by Gets](#)
- [SQL ordered by Reads](#)
- [SQL ordered by Executions](#)
- [SQL ordered by Parse Calls](#)
- [SQL ordered by Sharable Memory](#)

- [SQL ordered by Version Count](#)
- [Complete List of SQL Text](#)

[Back to Top](#)

11.6.1 SQL ordered by Elapsed Time

From this section one can find out quickly which SQLs incurred the largest amount of elapsed time. In addition to elapsed time and CPU time, the % total DB time metric is more indicative of which SQLs are potential tuning candidates. SQLs are identified by Id, which is convenient for keeping track of hot SQLs moving forward. In this example, the hottest SQL has an Id of *gsmv6*. . ., and let's see how it behaved by other metrics in the next few sections.

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
11,818	11,763	40,514	0.29	32.04	gsmv6cvsysnys	app.exe	SELECT T119.C1, C400079600, C4...
5,890	5,862	20,265	0.29	15.97	624kyhqp82nvm	app.exe	SELECT T119.C1, T119.C1 FROM T...
5,768	5,763	19,986	0.29	15.64	1wfvddxcf841m	app.exe	SELECT T62.C1, C490008000, C49...
5,386	5,380	20,266	0.27	14.60	gfjauwnsqzfx	app.exe	SELECT T62.C1, C179 FROM T62 W...
4,915	4,911	19,680	0.25	13.33	6h79rp4gyts91	app.exe	SELECT T119.C1, C179 FROM T119...
56	56	19,989	0.00	0.15	c070097bggvup	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
55	55	18,778	0.00	0.15	4pb2scqxpjgcn	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
49	49	20,279	0.00	0.13	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
28	28	9,273	0.00	0.08	1gbu3raazg6dy	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
28	28	9,236	0.00	0.08	ch8pbbp3r7xv0	app.exe	SELECT C1, C2, C3, C4, C5, C6,...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.2 SQL ordered by CPU Time

This section identifies hot SQLs by CPU time. It is seen that the SQL with the Id of gsmv6... continues to be the hottest SQL by this metric.

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
11,763	11,818	40,514	0.29	32.04	gsmv6cvsysnys	app.exe	SELECT T119.C1, C400079600, C4...
5,862	5,890	20,265	0.29	15.97	624kyhpq82nvm	app.exe	SELECT T119.C1, T119.C1 FROM T...
5,763	5,768	19,986	0.29	15.64	1wfvddxcf841m	app.exe	SELECT T62.C1, C490008000, C49...
5,380	5,386	20,266	0.27	14.60	gfjauwnsqxzf	app.exe	SELECT T62.C1, C179 FROM T62 W...
4,911	4,915	19,680	0.25	13.33	6h79rp4gyts91	app.exe	SELECT T119.C1, C179 FROM T119...
56	56	19,989	0.00	0.15	c070097bgvup	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
55	55	18,778	0.00	0.15	4pb2scqxpjgcn	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
49	49	20,279	0.00	0.13	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
28	28	9,273	0.00	0.08	1gbu3raazg6dy	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
28	28	9,236	0.00	0.08	ch8pbbp3r7xv0	app.exe	SELECT C1, C2, C3, C4, C5, C6,...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.3 SQL ordered by Gets

This section identifies hot SQLs by the number of buffer gets or logical reads. For this example, this is the most revealing metric that clearly links the excessive number of buffer gets as the underlying responsible cause that drove the DB CPUs busy fetching data repeatedly from the buffer cache. We could just pause and ponder over why excessive buffer gets had occurred with this SQL, but for the purpose of understanding all the parts of an AWR report, let's continue perusing the remainder of this AWR report.

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 1,405,702,082
- Captured SQL account for 99.6% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
487,795,411	40,514	12,040.17	34.70	11762.78	11817.71	gsmv6cvsyns	app.exe	SELECT T119.C1, C400079600, C4...
243,887,965	20,265	12,034.94	17.35	5861.58	5889.68	624kyhpq82nvm	app.exe	SELECT T119.C1, T119.C1 FROM T...
222,103,691	20,266	10,959.42	15.80	5380.30	5385.82	gfjauwnsqxzfx	app.exe	SELECT T62.C1, C179 FROM T62 W...
220,993,259	19,680	11,229.33	15.72	4910.95	4914.69	6h79rp4gyts91	app.exe	SELECT T119.C1, C179 FROM T119...
219,156,990	19,986	10,965.53	15.59	5762.77	5767.73	1wfvddxcf841m	app.exe	SELECT T62.C1, C490008000, C49...
668,571	20,279	32.97	0.05	48.56	49.03	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
272,137	19,651	13.85	0.02	18.99	19.03	d0cp76dp8mfhq	app.exe	UPDATE T331 SET C459 = EMPTY_C...
271,772	19,675	13.81	0.02	18.03	23.92	ffbpi4hpp19ty	app.exe	UPDATE T331 SET C456 = EMPTY_C...
262,946	37,551	7.00	0.02	23.27	23.30	079k3hnw05ppq4	app.exe	SELECT C400079600 FROM T147 WH...
227,517	19,626	11.59	0.02	14.10	14.13	1tdwt74ht67q6	app.exe	SELECT C459 FROM T331 WHERE C1...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.4 SQL ordered by Reads

This section lists SQLs by physical reads. It's interesting that the SQL with the Id of gsmv6... identified as the hot SQL is even not listed here, which should not be surprising given the huge number of about 488 million buffer gets incurred. Because of their relatively smaller percentages of elapsed time compared with the SQLs listed in the previous two sections, one can conclude that none of the SQLs listed here are hot ones.

SQL ordered by Reads

- Total Disk Reads: 265
- Captured SQL account for 3.4% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1	42	0.02	0.38	0.02	0.02	04xtrk7uyhknh		select obj#, type#,
								ctime, mti...
1	20,265	0.00	0.38	5861.58	5889.68	624kyhpg82nvm	app.exe	SELECT T119.C1, T119.C1 FROM T...
1	27	0.04	0.38	0.02	0.02	8swypbbr0m372		select order#, columns, types ...
1	2	0.50	0.38	0.01	0.01	asvzxj61dc5vs		select timestamp, flags from ...
1	27	0.04	0.38	0.02	0.02	cqgv56fmuj63x		select owner#, name, namespace...
1	8	0.13	0.38	0.00	0.02	cvn54b7yz0s8u		select /*+ index(idl_sb1\$ i_id...
1	143	0.01	0.38	0.05	0.05	f8pavn1bvsj7t		select con#, obj#, rcon#, enab...
1	8	0.13	0.38	0.01	0.01	ga9j9xk5cy9s0		select /*+ index(idl_sb4\$ i_id...
1	10,499	0.00	0.38	19.67	19.67	gxftzhd310ahj	app.exe	UPDATE T119 SET C400131300="S...
0	19,995	0.00	0.00	12.31	12.37	052whz4f6cwwd	app.exe	SELECT C400079600, C200000020 ...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.5 SQL ordered by Executions

This section lists SQLs by the number of executions. The hot SQL with the Id of gsmv6... reappeared as the hottest SQL here. Note that many SQLs have been omitted to save space.

SQL ordered by Executions

- Total Executions: 1,305,519
- Captured SQL account for 72.6% of Total

Executions	Rows Processed	Rows per Exec	CPU per Exec (s)	Elap per Exec (s)	SQL Id	SQL Module	SQL Text
40,514	40,531	1.00	0.29	0.29	gsmv6cvsysnys	app.exe	SELECT T119.C1, C400079600, C4...
40,002	39,994	1.00	0.00	0.00	g3m1pv9cpw6h0	app.exe	SELECT C400079600 FROM T119 WH...
37,551	37,548	1.00	0.00	0.00	079k3hnnw05pq4	app.exe	SELECT C400079600 FROM T147 WH...
20,279	20,279	1.00	0.00	0.00	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
20,279	20,279	1.00	0.00	0.00	fazmqt9m11gj7	app.exe	INSERT INTO H62 (entryId, T0, ...
...	...	...	...	...	...	...	...
18,240	18,240	1.00	0.00	0.00	64n9254p0hpkw	app.exe	INSERT INTO T109 (C179, C40012...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.6 SQL ordered by Parse Calls

This section lists SQLs by the number of parse calls. For this example, it proves that no SQLs were parsed more excessively than others. Note that many SQLs have been omitted to save space.

SQL ordered by Parse Calls

- Total Parse Calls: 1,302,081
- Captured SQL account for 72.8% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
40,529	40,514	3.11	gsmv6cvsysnys	app.exe	SELECT T119.C1, C400079600, C4...
40,015	40,002	3.07	g3m1pv9cpw6h0	app.exe	SELECT C400079600 FROM T119 WH...
37,578	37,551	2.89	079k3hnnw05pq4	app.exe	SELECT C400079600 FROM T147 WH...
20,278	20,279	1.56	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
20,278	20,279	1.56	fazmqt9m11gj7	app.exe	INSERT INTO H62 (entryId, T0, ...
...	...	...	...	...	...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.7 SQL ordered by Sharable Memory

This section captures the statistics about the amount of sharable memory in the shared pool consumed by an object. For this example, it contains no data, so we can skip it.

SQL ordered by Sharable Memory

No data exists for this section of the report.

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.8 SQL ordered by Version Count

This section captures the SQLs that had high version counts. To optimize the execution of a SQL statement, Oracle optimizer may rewrite a SQL, which would result in a new version of the same SQL statement. This typically has something to do with the setting of the parameter `CURSOR_SHARING`, which can take one of the values of {EXACT, SIMILAR, FORCE}. If you suspect that some SQLs perform suboptimally because their version count values are too high, you can experiment with the parameter `CURSOR_SHARING` and see which option gives you the best

result. But typically it might be more rewarding to spend time somewhere else in an AWR report to look for root causes that cause your database to slow down.

SQL ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
451	30	8xxw6k8rbv5bs	app.exe	SELECT T482.C1 FROM T482 WHERE...
180	12	0by3hna8a3bax	app.exe	SELECT T451.C1 FROM T451 WHERE...
82	6	0xw0nuxy0s9y6	app.exe	SELECT T553.C1 FROM T553 WHERE...
33	4	g0h1k8rxma68d	app.exe	SELECT T325.C1 FROM T325 WHERE...

[Back to SQL Statistics](#)

[Back to Top](#)

11.6.9 Complete List of SQL Text

This section lists SQLs with their full texts, indexed by SQL Id. You might be interested only in those SQLs that are impacting the performance of your Oracle-based application. After identifying those offending SQLs, you can extract the relevant SQL texts here and decide what you can do to help alleviate the adverse impacts of those offending SQLs. Note that in this section, only those five high buffer gets SQLs are retained to save space.

Before concluding this section, it is pointed out here that you can determine whether the concept of a bind variable is used in your Oracle application by looking for patterns like `=:"SYS_B_n"` which is an indication that bind variable is not used. From the texts of the following SQLs, you can see a lot of such patterns, so you can conclude that this application was not using bind variables. One of the reasons that Oracle bind variables are not used is that the product has to support all major database platforms.

SQL Id	SQL Text
1wfvddxcf841m	SELECT T62.C1, C490008000, C490009000, C179, C400129200 FROM T62 WHERE ((T62.C490009000 = ::"SYS_B_0") AND (T62.C400079600 = ::"SYS_B_1")) ORDER BY ::"SYS_B_2" ASC
624kyhqpq82nvm	SELECT T119.C1, T119.C1 FROM T119 WHERE (T119.C179 = ::"SYS_B_0") ORDER BY ::"SYS_B_1" ASC
6h79rp4gyts91	SELECT T119.C1, C179 FROM T119 WHERE (T119.C179 = ::"SYS_B_0") ORDER BY ::"SYS_B_1" ASC
gfjauwnsqzfx	SELECT T62.C1, C179 FROM T62 WHERE (T62.C179 = ::"SYS_B_0") ORDER BY ::"SYS_B_1" ASC
gsmv6cvsynsys	SELECT T119.C1, C400079600, C400127400, C400129200 FROM T119 WHERE (T119.C179 = ::"SYS_B_0") ORDER BY ::"SYS_B_1" ASC

[Back to SQL Statistics](#)

[Back to Top](#)

11.7 INSTANCE ACTIVITY STATISTICS

This part of an AWR report summarizes the database instance activity statistics, with all activities divided into three categories: a comprehensive report of almost all aspects of an instance, an absolute value section, and a thread section, as is shown below. We'll focus on the first section as that's where one can get a glimpse of how an Oracle database instance performed during the period of the test.

Instance Activity Statistics

- [Instance Activity Stats](#)
- [Instance Activity Stats - Absolute Values](#)
- [Instance Activity Stats - Thread Activity](#)

[Back to Top](#)

11.7.1 Instance Activity Stats

Since the statistics presented in this section are overwhelming, we examine them by category rather than individually. Although one may not need to look at every statistic here in a real world Oracle performance troubleshooting scenario, going over the gigantic list of all the instance activity statistics listed below is helpful for us to understand the various concepts and instruments built into Oracle as it evolved from version to version.

As is shown below, each statistic is appended with three attributes: total, per second, and per transaction. It is hard to make sense of each metric value standalone, so the information presented here is best consumed in conjunction with your application domain and other metrics found somewhere else in this report, for example, hinting a misconfiguration, a poorly written or tuned SQL, undersized hardware, and so on, that contributes to the poor performance of the Oracle application under test. Therefore, we'll focus mainly on the meaning of each statistic, rather than whether a metric value shown here is good, bad, or normal. These statistics are explained as follows:

- *CPU Metrics.* This category includes two metrics: CPU used by this session and CPU used when call started. Both of these two metrics are measured in centiseconds (1 centisecond = 10 milliseconds). The first metric measures the CPU time used by the instance during the period bound by the two snapshots, while the second metric has a cut-off start time of when the user call started. If no new users logged on during the measurement period, then these two metrics would give approximately the same numbers, as was the case with this example. Note that these metrics are cumulative. Since the database server for this example actually had 8 cores and each core had 4 parallel computing threads, the maximum total CPU time could be calculated as 8 [cores] × 4 [threads] ×

$3600 [\text{seconds}] \times 100 [\text{centiseconds}] = 11,520,000 \text{ centiseconds}$. Given the total CPU time of 3,659,941 centiseconds as reported below, we could deduce that the portion of the Oracle CPU time was $3,659,941 / 11,520,000 = 31.8\%$. This was close to the actual average CPU usage of 36% as measured at the OS-level. The extra 4.2% relative to the DB server usage could be attributed to the overhead from the OS and the Oracle background processes.

- *CR Blocks Created.* Here the abbreviation CR stands for Consistent Reads, which refers to the requirement that the values of the attributes of the rows queried should be the same as the original values when the query started. This is relevant to the likelihood that the data the query returned had changed since the query started, which occurs when multiple users read and update the same data concurrently. To circumvent this common issue, Oracle keeps multiple versions of the same data in the buffer cache. The most up-to-date copy of the data is maintained in the current buffer or current block with all recent updates contained. A CR buffer or CR block (CR_BLOCK) contains the version of the data at a particular time prior to the current buffer to help maintain the read consistency that the data in the CR buffer are consistent as measured by the start time of the query. This subject is best clarified in the Oracle 11g Database Reference in the context of a V\$ metric of V\$INSTANCE_CACHE_TRANSFER. Read consistency is achieved through the SCN (system change number), which is the next metric to be examined.
- *SCN Metrics.* An SCN is a sequential counter for identifying precisely a moment in time. To get the current SCN, use the query `SELECT CURRENT_SCN FROM V$DATABASE;`. Besides, each table has an ORA_ROWSCN attribute, which represents the latest COMMIT operation for a particular row. One can use the function `SCN_TO_TIMESTAMP` to convert an SCN to an actual timestamp. The below example shows that 66 Commit SCN cached, which were referenced more than 5000 times.
- *DBWR Metrics.* This set of metrics provides statistical information about the database writer (DBWR) process associated with the checkpoint, transaction and undo operations. It is helpful for evaluating how the DBWR process had performed.
- *IMU Metrics.* The abbreviation IMU stands for In-Memory Undo, which is an Oracle patented performance optimization technique of maintaining undo in memory rather than in segments. Oracle introduced this performance optimization technique in 10g, hoping that the resource stress incurred by segment-based undo operations could be alleviated with IMU. In general, this is less a problem if you don't see the IMU-labeled wait events in your top event list, for example, the wait event of latch: In-memory undo latch. You are encouraged to investigate further if you run into such a situation.
- *SQL*Net Roundtrips to/from Client.* SQL*Net was Oracle's client/server middleware product that was renamed to Net8 in Oracle8. It is responsible for enabling network connections between an Oracle server and its clients, whether they are on the same system or on the two separate systems. It is based on

Oracle's Transparent Network Substrate (TNS) technology, which provides a generic interface to all popular network protocols for connectivity between an Oracle database and its client or another database. Note that SQL*Net is part of the Oracle large-scale scalability building blocks, which allow thousands of concurrent users to connect to an Oracle database.

SQL*Net is configured with the following three files:

- o The `tnsnames.ora` file, which you are already familiar with. It defines the connect string for each instance of an Oracle database for the client to connect to.
- o The `sqlnet.ora` file, which defines basic configuration details such as tracing options, default domain, encryption, and `expire_time` for dead connection detection, and so on.
- o The `listener.ora` file, which configures Oracle listeners with such information as protocols, keys, hosts, and ports. A listener is required to accept connection requests.

In this example, it shows that about 13 round trips occurred between the Oracle server and the application server. If this number is too large, one optimization technique is to use chunking or packing more data into one load to reduce the chattiness between the database server and application server so that the overall network latency can be improved.

- *Cleanout-Related Metrics.* A cleanout is an operation to release the buffer blocks held temporarily for certain purpose such as consistent reads as was discussed previously. A commit cleanout is an attempted cleanout at commit time of a transaction. Note that in this example the metric *active txn count during cleanout* had a value of 2.57 per transaction, which means that on average 2.57 transactions alive at cleanout. This is possible as transactions can run concurrently.
- *Buffer Pinned Count.* A buffer is said to be in a pinned state when it is in use. According to Oracle's database reference document, these metrics associated with buffer pinned are for internal debugging purposes only.
- *Calls to Kcmxxx Metrics.* These metrics are for internal debugging purposes. The three acronyms, `kcmgas`, `kcmgcs`, and `kcmgrs`, stand for the KCM (Kernel Cache Layer Miscellaneous) routines of Get and Advance SCN (GAS), Get Current SCN (GCS), and Get Recent SCN (GRS), respectively. You probably would want to ignore such metrics unless you are interested in probing Oracle internals.
- *Cluster Key Scan Metrics.* These metrics are related to scanning the cluster keys of a table cluster. They are for information purposes only.
- *Concurrency Wait Time.* This metric is a measure of concurrency contention. In this example, it's about 28 seconds in total, which is negligible.
- *Consistent Gets Metrics.* This set of metrics provides information on consistent gets. A buffer in the buffer cache can have one of the many possible states such as

FREE (not currently in use), XCUR (exclusive), SCUR (shared current), CR (consistent read), READ (being read from disk), PI (past image), and so on. The concept of CR is related to a process named buffer cloning, which makes a copy of the buffer at the point of time when the buffer contents are considered consistent. The cloned buffer copies are called CR copies. A consistent get is a logic read operation from one of the CR copy buffers when the buffer is in consistent read state or mode. Note that in this example, an extremely high value of over 1.4 billion was reported for the consistent gets metric. Some texts suggest that this could be quickly fixed by adding more CPU power, which may be specious. A case study to be presented later shows that by adding proper indexes, the number of consistent gets was reduced to about 6 million with everything else the same, which fixed the poor scalability of an Oracle-based enterprise application decisively.

- *DB Block Metrics.* This set of metrics includes four metrics as follows:
 - *DB block changes.* This metric reports the total number of changes due to update or delete operations made to all blocks in the SGA. Such changes result in redo log entries and become permanent if the transaction is committed. It also measures the rate at which buffers are being dirtied.
 - *DB block gets.* This metric reports the number of times a CURRENT_BLOCK was requested. As was explained previously in the context of a CR block, a CURRENT block contains all up-to-date changes to a block since the CR copy was cloned. It is requested when a DML operation is initiated. DB block gets and consistent gets constitute logical reads. A large value of this metric typically results from OLTP type of applications, whereas a small value results from batch-job type of applications. To put it into perspective, the value of about 23 DB block gets per transaction in this example resulted from a batch-job type of application.
 - *DB block gets direct.* This metric reports the number of times a CURRENT block was read directly to the user's process global area (PGA) with the buffer cache bypassed. DB block gets direct favors data that is static and user specific.
 - *DB block gets from cache.* This is the other part of DB block gets in addition to DB block gets direct, as a DB block get can only be either from cache or direct. Note that if a block is not found in the cache, a disk read is initiated, which is called a cache miss. Also note that a 100% cache hit ratio does not necessarily mean a well-performing database, as is illustrated with this real world case study.
- *Enqueue Metrics.* In order to explain the concept of Oracle enqueue, let's explore Oracle locks first. Locking is a mechanism to allow multiple users to modify the same data in a serialized fashion. The data is locked by the transaction of a user until it is committed or rolled back before the next user can modify it. Oracle allows a variety of locks on various database resources such as a single or multiple rows, an entire or multiple tables, and so on. Enqueues are basically database locks that serialize access to database resources. After a user or session

acquires an enqueue on a database resource, other users are prevented from accessing the data locked by the current user. When the user or session is done with its transaction, the next user would acquire an enqueue for its transaction, and other users, if any, will have to wait for the current user's enqueue to be released, and so on.

An AWR report includes the following enqueue-related metrics:

- *Enqueue Conversions.* This metric reports the total number of conversions of the state of table or row lock. In this example, it is seen that 722 enqueue conversions had occurred during the one-hour measurement period.
- *Enqueue Releases.* This metric reports the total number of table or row locks released. In this example, 3.22 enqueues were released per transaction on average.
- *Enqueue Requests.* This metric reports the total number of table or row locks acquired. In this example, the number of enqueues acquired is about the same as the number of enqueues released.
- *Enqueue Timeouts.* This metric reports the total number of table and row locks (acquired and converted) that timed out before the transactions were completed. In this example, no enqueue timeouts occurred, which is desirable.
- *Enqueue Waits.* This metric is self-explanatory, namely, the total number of enqueue waits because of other users' enqueues were still in effect. In this example, only 690 enqueue waits occurred, which is a small fraction (0.15%) of the total number of enqueues acquired and released.
- *Exchange Deadlocks.* When two buffers are exchanged, potentially it could result in a deadlock. This metric reports the number of such potential deadlocks. Exchange deadlocks occur only with index scan operations.
- *Execute Count.* This metric reports the total number of calls (user and recursive) that executed SQL statements. In this example, about nine executes occurred per transaction on average.
- *Free Buffer Inspected and Requested.* The metric of free buffer inspected reports the number of buffers examined and skipped over from the end of an LRU queue in order to find a reusable buffer, whereas the metric of free buffer requested reports the number of times a free buffer or a reusable buffer was requested to create or load a block from disk. In this example, on average, 0.36 free buffers were inspected and 2.68 free buffers were requested per transaction.
- *Heap Block Compression.* This metric reports the number of times heap block compression occurred during the measurement period. As data buffers are allocated over time, a heap may become patched with disjoint blocks. Heap block compression is a technique to compress patched regions so that larger free regions can be allocated more efficiently. In this example, heap block compression occurred at a rate of 23.54 times per second.

- *Hot Buffers Moved to Head of LRU.* This metric reports the number of times that hot buffers were moved from the end to the head of an LRU list to prevent them from being reused. In this example, such moves occurred at a rate of 11.23 per second.
- *Index Operational Metrics.* This set of metrics reports the number of index fast full scans for full segments, index fetch by key, and index scans `kdiixs1` (which is not documented by Oracle), respectively. In this example, it is seen that far more index fetch by key occurred than index full fast scans.
- *LeafNode Metrics.* The first metric of leaf node 90–10 splits is not documented by Oracle. The second metric of leaf node splits reports the number of times an index leaf node was split forced by the insertion of additional values. In this example, leaf node splits occurred at a rate of 1.42 per second.
- *Lob Metrics.* This set of metrics reports the numbers of LOB API read/write operations. The term *unaligned* in the third metric reports the number of LOB API writes whose start offset or buffer size were not aligned to the internal chunk size of the LOB. In this example, it is seen that all LOB writes were unaligned.
- *Logons Cumulative.* This metric reports the total number of logons since the instance started. In this example, only three logons occurred.
- *Messages Received/Sent.* These metrics report the numbers of messages exchanged between background processes. In this example, about 26 messages were exchanged.
- *No Buffer To Keep Pinned Count.* This metric reports the number of times the buffers were not found where expected. It has a zero value in this example.
- *No Work—Consistent Read Gets.* This metric reports the number of consistent gets that required no work of cleanout or rollback. In this example, about 1.4 billion of such consistent gets occurred, which is good that so many consistent gets required no work, but this number itself is too excessive.
- *Open Cursors Cumulative.* This metric reports the total number of cursors opened since the instance started.
- *Parse Metrics.* This set of metrics is self-explanatory, given the concept of parsing explained previously.
- *Physical IO Metrics.* This set of metrics summarizes the physical IO (read and write) operations. Each metric is fairly self-explanatory. Also, a better place to look at IO issues is the IO Stats section, which will be discussed in the next section.
- *Prefetch Metrics.* This set of metrics reports the prefetch activities. It is more useful for Oracle kernel development than for external applications.
- *Recursive Metrics.* This set of metrics reports the number of recursive calls in the database as well as total CPU usage used by non-user or recursive calls. In this example, on average less than two recursive calls occurred per transaction. The user call CPU time can be obtained by subtracting this value from the metric of “CPU used by this session.” It is seen that the recursive CPU time was insignificant relative to the total user call CPU time.

- *Redo Metrics.* This set of metrics reports the statistics about the redo activities. Each of the metrics is self-explanatory, given the concept of redo explained previously.
- *Rollback Metrics.* The first metric of rollback changes—undo records applied reports the number of undo records applied to user-requested rollbacks (not consistent rollbacks), while the second metric of rollbacks only—consistent read gets reports the number of consistent gets that require only block rollbacks, but no block cleanouts.
- *Rows Fetched via Callback.* This metric is only useful for internal debugging purposes.
- *Session Metrics.* This set of metrics reports the statistics about the session activities. Note the 1.4 billion session logical reads in this example. Also note the session PGA and UGA memory statistics.
- *Sort Metrics.* The first metric of sorts (memory) reports the number of sort operations that were performed in memory with no disk write operations, whereas the second metric of sorts (rows) reports the total number of rows sorted. Since sort operations are caused by the where-clause conditions with table join SQL operations, it can be deduced based on the number of sorts per transaction that this example was not a read-intensive application.
- *Switch Current to New Buffer.* This metric reports the number of times the CURRENT block moved to a new buffer, leaving the CR block in the original buffer. In this example, 979 such switches occurred, which was insignificant.
- *Table Metrics.* This set of metrics gives a detailed view of all major statistics associated with table-related activities. Note the first metric of table fetch by rowid that reported that over 1.5 billion rows were fetched by ROWID, which was usually recovered from an index. The other metrics are related to table scan activities.
- *Transaction Metrics.* This set of metrics provides statistics about transaction rollbacks. Note that 189 transactions were rolled back.
- *User Metrics.* This set of metrics reports the statistics about the user I/O wait time, the number of user calls, user commits, and user rollbacks. In this example, the user I/O wait time was 2.11 seconds only over a one-hour measurement period, which was inconsequential.
- *Workarea Executions—optimal.* This metric reports the number of executions of such operations as sort or join performed in memory where the required memory area was available immediately (single pass or optimal) without having to try multiple times (multipass). It's seen that out of 0.83 sorts per transaction performed in memory (c.f. the metric sorts [memory]), 0.62 executions occurred optimally. This further confirms that most sorts were performed in memory efficiently.
- *Write Clones Created in Foreground.* This last metric reports the number of times a foreground process clones a CURRENT buffer that is being written. The clone becomes the new CURRENT buffer, leaving the original buffer to complete writing.

Instance Activity Stats

Statistic	Total	per Second	per Trans
CPU used by this session	3,659,941	1,021.08	25.11
CPU used when call started	3,658,384	1,020.65	25.10
CR blocks created	351,643	98.10	2.41
Cached Commit SCN referenced	5,036	1.40	0.03
Commit SCN cached	66	0.02	0.00
DB time	3,785,150	1,056.01	25.97
DBWR checkpoint buffers written	57,146	15.94	0.39
DBWR checkpoints	2	0.00	0.00
DBWR object drop buffers written	0	0.00	0.00
DBWR tablespace checkpoint buffers written	0	0.00	0.00
DBWR transaction table writes	458	0.13	0.00
DBWR undo block writes	27,599	7.70	0.19
IMU CR rollbacks	235,241	65.63	1.61
IMU Flushes	48,928	13.65	0.34
IMU Redo allocation size	199,358,404	55,618.67	1,367.73
IMU commits	94,903	26.48	0.65
IMU contention	23,592	6.58	0.16
IMU ktichg flush	0	0.00	0.00
IMU pool not allocated	1,927	0.54	0.01
IMU recursive-transaction flush	0	0.00	0.00
IMU undo allocation size	568,225,688	158,528.35	3,898.39
IMU- failed to get a private strand	1,927	0.54	0.01
SMON posted for dropping temp segment	0	0.00	0.00
SMON posted for undo segment shrink	0	0.00	0.00
SQL*Net roundtrips to/from client	1,838,240	512.85	12.61
active txn count during cleanout	374,310	104.43	2.57
application wait time	0	0.00	0.00
background checkpoints completed	2	0.00	0.00
background checkpoints started	2	0.00	0.00
background timeouts	14,864	4.15	0.10
branch node splits	7	0.00	0.00
buffer is not pinned count	1,393,951,627	388,896.27	9,563.40
buffer is pinned count	1,694,955,902	472,872.96	11,628.48
bytes received via SQL*Net from client	633,494,876	176,737.69	4,346.18
bytes sent via SQL*Net to client	1,153,758,576	321,885.21	7,915.52
calls to get snapshot scn: kcmgss	1,847,334	515.38	12.67
calls to kcmgas	517,155	144.28	3.55
calls to kcmgcs	6,504	1.81	0.04
change write time	3,909	1.09	0.03
cleanout - number of ktugct calls	141,191	39.39	0.97
cleanouts and rollbacks - consistent read gets	102,660	28.64	0.70

cleanouts only - consistent read gets	5,587	1.56	0.04
cluster key scan block gets	6,578	1.84	0.05
cluster key scans	1,398	0.39	0.01
commit batch/immediate performed	189	0.05	0.00
commit batch/immediate requested	189	0.05	0.00
commit cleanout failures: block lost	0	0.00	0.00
commit cleanout failures: buffer being written	4	0.00	0.00
commit cleanout failures: callback failure	243	0.07	0.00
commit cleanout failures: cannot pin	31,002	8.65	0.21
commit cleanouts	998,243	278.50	6.85
commit cleanouts successfully completed	966,994	269.78	6.63
commit immediate performed	189	0.05	0.00
commit immediate requested	189	0.05	0.00
commit txn count during cleanout	70,399	19.64	0.48
concurrency wait time	2,803	0.78	0.02
consistent changes	808,557	225.58	5.55
consistent gets	1,402,333,336	391,234.67	9,620.90
consistent gets - examination	3,517,992	981.48	24.14
consistent gets direct	74	0.02	0.00
consistent gets from cache	1,402,333,239	391,234.64	9,620.90
cursor authentications	400	0.11	0.00
data blocks consistent reads - undo records applied	783,688	218.64	5.38
db block changes	2,881,467	803.90	19.77
db block gets	3,368,399	939.74	23.11
db block gets direct	39,515	11.02	0.27
db block gets from cache	3,328,882	928.72	22.84
deferred (CURRENT) block cleanout applications	419,921	117.15	2.88
dirty buffers inspected	0	0.00	0.00
enqueue conversions	722	0.20	0.00
enqueue releases	470,052	131.14	3.22
enqueue requests	470,046	131.14	3.22
enqueue timeouts	0	0.00	0.00
enqueue waits	690	0.19	0.00
exchange deadlocks	2,358	0.66	0.02
execute count	1,305,519	364.22	8.96
free buffer inspected	52,876	14.75	0.36
free buffer requested	390,350	108.90	2.68
heap block compress	84,373	23.54	0.58
hot buffers moved to head of LRU	40,248	11.23	0.28
immediate (CR) block cleanout applications	108,247	30.20	0.74
immediate (CURRENT) block cleanout applications	13,654	3.81	0.09
index fast full scans (full)	24	0.01	0.00
index fetch by key	1,160,266	323.70	7.96
index scans kdiixs1	401,810	112.10	2.76

leaf node 90-10 splits	889	0.25	0.01
leaf node splits	5,094	1.42	0.03
lob reads	88	0.02	0.00
lob writes	39,454	11.01	0.27
lob writes unaligned	39,454	11.01	0.27
logons cumulative	3	0.00	0.00
messages received	94,907	26.48	0.65
messages sent	94,906	26.48	0.65
no buffer to keep pinned count	0	0.00	0.00
no work - consistent read gets	1,397,722,963	389,948.43	9,589.27
opened cursors cumulative	5,768	1.61	0.04
parse count (failures)	0	0.00	0.00
parse count (hard)	423	0.12	0.00
parse count (total)	1,302,081	363.27	8.93
parse time cpu	39,710	11.08	0.27
parse time elapsed	42,498	11.86	0.29
physical read IO requests	261	0.07	0.00
physical read bytes	2,170,880	605.65	14.89
physical read total IO requests	1,598	0.45	0.01
physical read total bytes	23,963,648	6,685.58	164.41
physical read total multi block requests	2	0.00	0.00
physical reads	265	0.07	0.00
physical reads cache	179	0.05	0.00
physical reads cache prefetch	4	0.00	0.00
physical reads direct	86	0.02	0.00
physical reads direct (lob)	38	0.01	0.00
physical reads direct temporary tablespace	0	0.00	0.00
physical reads prefetch warmup	0	0.00	0.00
physical write IO requests	46,972	13.10	0.32
physical write bytes	792,084,480	220,982.35	5,434.21
physical write total IO requests	147,227	41.07	1.01
physical write total bytes	1,763,140,608	491,895.70	12,096.27
physical write total multi block requests	99,742	27.83	0.68
physical writes	96,690	26.98	0.66
physical writes direct	39,527	11.03	0.27
physical writes direct (lob)	39,396	10.99	0.27
physical writes direct temporary tablespace	0	0.00	0.00
physical writes from cache	57,163	15.95	0.39
physical writes non checkpoint	80,950	22.58	0.56
pinned buffers inspected	2	0.00	0.00
prefetch warmup blocks aged out before use	591	0.16	0.00
prefetch warmup blocks flushed out before use	0	0.00	0.00
prefetched blocks aged out before use	245	0.07	0.00
process last non-idle time	18,081	5.04	0.12

recursive calls	244,866	68.31	1.68
recursive cpu usage	6,360	1.77	0.04
redo blocks written	1,742,030	486.01	11.95
redo buffer allocation retries	23	0.01	0.00
redo entries	697,039	194.47	4.78
redo log space requests	32	0.01	0.00
redo log space wait time	230	0.06	0.00
redo ordering marks	14,613	4.08	0.10
redo size	840,913,608	234,605.10	5,769.21
redo synch time	5	0.00	0.00
redo synch writes	771	0.22	0.01
redo wastage	23,011,188	6,419.85	157.87
redo write time	103,584	28.90	0.71
redo writer latching time	9	0.00	0.00
redo writes	87,382	24.38	0.60
rollback changes - undo records applied	693	0.19	0.00
rollbacks only - consistent read gets	248,583	69.35	1.71
rows fetched via callback	680,070	189.73	4.67
session connect time	0	0.00	0.00
session cursor cache hits	1,275,126	355.75	8.75
session logical reads	1,405,702,082	392,174.51	9,644.02
session pga memory	6,505,120	1,814.85	44.63
session pga memory max	13,893,632	3,876.16	95.32
session uga memory	2,510,520	700.41	17.22
session uga memory max	13,761,144	3,839.20	94.41
shared hash latch upgrades - no wait	465,468	129.86	3.19
shared hash latch upgrades - wait	6,625	1.85	0.05
sorts (memory)	121,600	33.92	0.83
sorts (rows)	124,907	34.85	0.86
sql area purged	0	0.00	0.00
summed dirty queue length	0	0.00	0.00
switch current to new buffer	979	0.27	0.01
table fetch by rowid	1,544,240,928	430,825.24	10,594.48
table fetch continued row	9	0.00	0.00
table scan blocks gotten	11,141	3.11	0.08
table scan rows gotten	460,925	128.59	3.16
table scans (direct read)	0	0.00	0.00
table scans (long tables)	0	0.00	0.00
table scans (rowid ranges)	0	0.00	0.00
table scans (short tables)	21,452	5.98	0.15
total number of times SMON posted	0	0.00	0.00
transaction rollbacks	189	0.05	0.00
transaction tables consistent read rollbacks	0	0.00	0.00
transaction tables consistent reads - undo records applied	0	0.00	0.00

undo change vector size	177,047,012	49,394.05	1,214.66
user I/O wait time	211	0.06	0.00
user calls	2,850,182	795.17	19.55
user commits	145,759	40.67	1.00
user rollbacks	0	0.00	0.00
workarea executions - optimal	90,561	25.27	0.62
write clones created in foreground	27	0.01	0.00

[Back to Instance Activity Statistics](#)

[Back to Top](#)

11.7.2 Instance Activity Stats—Absolute Values

This section has three metrics: session cursor cache count, opened cursors current, and logons current. Each of these metrics is explained as follows:

- *Session Cursor Cache Count.* This metric reports the total number of cursors cached. It is closely related to a parameter named `SESSION_CACHED_CURSORS`, which specifies the number of session cursors to cache. If a SQL statement has been parsed multiple times, then the session cursor for that SQL statement is placed into the session cursor cache, which will prevent subsequent parse calls to reopen the cursor. Oracle uses an LRU algorithm to remove entries in the session cursor cache to make room for new entries to be cached. Each time a new cursor is added to the session cursor cache, the session cursor count increments by one. In this example, the session cursor cache count started at 19,076, and ended at 19,315.
- *Open Cursors Current.* This metric reports the total number of current open cursors. It is closely related to the initialization parameter named `OPEN_CURSORS` which specifies the maximum number of open cursors (handles to private SQL areas) a session can have at once. This parameter cannot be set too low, otherwise if the application runs out of open cursors, an error would be thrown; and it cannot be set too higher, otherwise a session may open up too many cursors, which will adversely affect performance. In this example, it started at 260 and ended at 265.
- *Logons Current.* This metric reports the total number of current logons. In this example, it started at 80 and ended at 80.

Instance Activity Stats—Absolute Values

- Statistics with absolute values (should not be diffed)

Statistic	Begin Value	End Value
session cursor cache count	19,076	19,315
opened cursors current	260	265
logons current	80	80

[Back to Instance Activity Statistics](#)

[Back to Top](#)

11.7.3 Instance Activity Stats—Thread Activity

This section contains only one metric—log switches (derived). It reports the number of times the redo log has switched from one log file to another. Since a redo log switch may take seconds, it affects performance if it occurs too frequently. In this example, only two log switches occurred over a one-hour period, which was inconsequential.

Instance Activity Stats—Thread Activity

- Statistics identified by '(derived)' come from sources other than SYSSTAT

Statistic	Total	per Hour
log switches (derived)	2	2.01

[Back to Instance Activity Statistics](#)

[Back to Top](#)

11.8 IO STATS

For enterprise applications, IO STATS is one of the most interesting sections to visit. From here, one can get an immediate glimpse of the IO performance associated with the application data storage, as will be shown below.

IO Stats

- [Tablespace IO Stats](#)
- [File IO Stats](#)

[Back to Top](#)

11.8.1 Tablespace IO Stats

Tablespace IO Stats section reveals the average read time for disk reads and average buffer wait time for disk writes. These numbers in general should be between 5 to 10 milliseconds or not exceed 20 milliseconds. In this example, the first tablespace is the application tablespace, and average read and write buffer wait time were 7.07 and 1.16 milliseconds, respectively. Also one can get the number of physical reads and writes from here, respectively, for instance, 205 physical reads and 44,056 writes for this example during the one-hour measurement period. Standard disk storage can support up to 1000 IOs/s, so based on the number of physical IOs, one can conclude that disk IO was not the bottleneck for this application.

Tablespace IO Stats

- ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	205	0	7.07	1.01	44,056	12	14,913	1.16
UNDO	4	0	7.50	1.00	2,380	1	1,660	0.11
SYSAUX	27	0	6.30	1.07	217	0	0	0.00
SYSTEM	16	0	8.75	1.00	179	0	0	0.00

[Back to IO Stats](#)

[Back to Top](#)

11.8.2 File IO Stats

The File IO Stats gives further information on IO statistics on a file-by-file basis. In this example, it shows that the average read time ranged from 6.30 to 10 milliseconds across all files. One can also check the read intensity on each file, so that more intensely accessed files can be placed on faster disk storage if options are available.

File IO Stats

- ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/data3/obs/app2.dbf	27	0	10.00	1.00	13,101	4	4,561	1.46
APP	/data3/obs/app3.dbf	30	0	6.67	1.00	14,742	4	4,857	0.90
APP	/data3/obs/app1	148	0	6.62	1.01	16,213	5	5,495	1.14
SYSAUX	/data3/obs/sysaux.dbf	27	0	6.30	1.07	217	0	0	0.00
SYSTEM	/data3/obs/system.dbf	16	0	8.75	1.00	179	0	0	0.00
UNDO	/data3/obs/undo.dbf	4	0	7.50	1.00	2,380	1	1,660	0.11

[Back to IO Stats](#)

[Back to Top](#)

11.9 BUFFER POOL STATISTICS

This section provides buffer pool statistics. Here it reports the DEFAULT buffer pool statistics only. The DEFAULT buffer pool has the block-size set by the DB_BLOCK_SIZE initialization parameter, and its pool size is set by the DB_CACHE_SIZE initialization parameters. There are two scenarios if the SGA_TARGET parameter is set: (a) if DB_CACHE_SIZE is set, then it is treated as a minimum memory pool size; (b) if DB_CACHE_SIZE is not set, it's determined internally by Oracle. However, if SGA_TARGET is not set, then the default value of DB_CACHE_SIZE is the greater of 48 MB and 4 MB * number of CPUs.

With this example, SAG_TARGET was set to about 1 GB and DB_CACHE_SIZE was not set. Therefore, the DEFAULT pool's size was determined internally by Oracle. Based on the number of 92,565 buffers and a standard block size of 8192 bytes, one can calculate that the DEFAULT pool size was 758.3 MB. Note that about 1.4 billion buffer gets (logical reads) occurred on this DEFAULT pool versus 181 physical reads and 57,163 physical writes. This excessive buffer gets phenomenon will be discussed later.

Buffer Pool Statistics

- Standard block size Pools D: default, K: keep, R: recycle
- Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k

P	Number of Buffers	Pool Hit%	Buffer Gets	Physical Reads	Physical Writes	Free Buff Wait	Writ Comp Wait	Buffer Busy Waits
D	92,565	100	1,395,211,688	181	57,163	0	0	16,585

[Back to Top](#)

11.10 ADVISORY STATISTICS

Advisory statistics provide further details about estimated instance recovery and various memory pools to aid memory tuning, as is shown below. Let's take a quick look at the instance recovery stats next and then move to various advisories.

Advisory Statistics

- [Instance Recovery Stats](#)
- [Buffer Pool Advisory](#)
- [PGA Aggr Summary](#)
- [PGA Aggr Target Stats](#)
- [PGA Aggr Target Histogram](#)

- [PGA Memory Advisory](#)
- [Shared Pool Advisory](#)
- [SGA Target Advisory](#)
- [Streams Pool Advisory](#)
- [Java Pool Advisory](#)

[Back to Top](#)

11.10.1 Instance Recovery Stats

This section provides ballpark numbers on the mean time to recovery (MTTR) time based on the log file and checkpoint file sizes. In this example, it shows that the MTTR would be in the range of 21 to 31 seconds.

Instance Recovery Stats

- B: Begin snapshot, E: End snapshot

	Target MTTR (s)	Estd MTTR (s)	Recovery Estd IOs	Actual Redo Blks	Target Redo Blks	Log File Size Redo Blks	Log Ckpt Timeout Redo Blks	Log Ckpt Interval Redo Blks
B 0	0	31	4304	202961	203174	2764800	203174	
E 0	0	21	236	1386	439081	2764800	439081	

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.2 Buffer Pool Advisory

This section hints on how one can reduce the number of physical reads with varying buffer pool sizes. The example below shows that by increasing the DEFAULT buffer size from 72 MB to 1.008 GB, the number of physical reads can be reduced from 155,476 to 93,477, or vice versa. The buffer pool advisory provides valuable information on how one can make trade-offs between performance and memory. For example, if the application is less physical IO intensive, then there is no need to oversize the buffer cache, or vice versa.

Buffer Pool Advisory

- Only rows with estimated physical reads >0 are displayed
- ordered by Block Size, Buffers For Estimate

P	Size for Est (M)	Size Factor	Buffers for Estimate	Est Phys Read Factor	Estimated Physical Reads
D	72	0.10	8,910	1.65	155,476
D	144	0.19	17,820	1.32	123,848
D	216	0.29	26,730	1.23	115,302
D	288	0.38	35,640	1.19	112,131
D	360	0.48	44,550	1.12	105,655
D	432	0.57	53,460	1.04	97,536
D	504	0.67	62,370	1.02	95,792
D	576	0.77	71,280	1.01	94,957
D	648	0.86	80,190	1.00	94,441
D	720	0.96	89,100	1.00	94,176
D	752	1.00	93,060	1.00	94,081
D	792	1.05	98,010	1.00	93,864
D	864	1.15	106,920	1.00	93,667
D	936	1.24	115,830	0.99	93,592
D	1,008	1.34	124,740	0.99	93,477
D	1,080	1.44	133,650	0.99	93,178
D	1,152	1.53	142,560	0.98	92,506
D	1,224	1.63	151,470	0.98	91,875
D	1,296	1.72	160,380	0.97	91,515
D	1,368	1.82	169,290	0.97	91,508
D	1,440	1.91	178,200	0.97	91,508

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.3 PGA Aggr Summary

This section has only one row of data: 100% PGA Cache Hit, 207 MB work area processed, and zero MB extra work area read/write. These numbers imply that there were no PGA-related performance issues.

PGA Aggr Summary

- PGA cache hit % - percentage of W/A (WorkArea) data processed only in-memory

PGA Cache Hit %	W/A MB Processed	Extra W/A MB Read/Written
100.00	207	0

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.4 PGA Aggr Target Stats

Since a PGA is a private area to an Oracle server process, the term *PGA aggregate target* or the actual parameter `PGA_AGGREGATE_TARGET` sets a limit for all server processes so that the aggregated PGA from all processes will not exceed what `PGA_AGGREGATE_TARGET` sets. In this example, the PGA Aggregate Target was 1627 MB, which was fixed as this AWR report was taken from a 10g server. The Auto PGA Target was the part of the total PGA that could be automatically managed, mostly work areas. The PGA Mem Alloc was the part that was actually allocated, which was seen to have increased from 149.75 MB (begin snap) to 160.94 MB (end snap). Therefore, we can conclude that the PGA was lightly touched during the one-hour measurement period and should not be the focal point of this report.

PGA Aggr Target Stats

- B: Begin snap E: End snap (rows identified with B or E contain data which is absolute i.e. not diffed over the interval)
- Auto PGA Target - actual workarea memory target
- W/A PGA Used - amount of memory used for all Workareas (manual + auto)
- %PGA W/A Mem - percentage of PGA memory allocated to workareas
- %Auto W/A Mem - percentage of workarea memory controlled by Auto Mem Mgmt
- %Man W/A Mem - percentage of workarea memory under manual control

	PGA Aggr Target(M)	Auto PGA Target(M)	PGA Mem Alloc(M)	W/A PGA Used(M)	%PGA W/A Mem	%Auto W/A Mem	%Man W/A Mem	Global Mem Bound(K)
B	1,627	1,387	149.75	0.00	0.00	0.00	0.00	166,600
E	1,627	1,384	160.94	0.00	0.00	0.00	0.00	166,600

[Back to Advisory Statistics](#)
[Back to Top](#)

11.10.5 PGA Aggr Target Histogram

This section shows the PGA aggregate target histogram. The term *Optimal* means that the size of the work area is optimal that the amount of memory required for a SQL execution is smaller than the work area size so that extra pass of finding additional memory is not needed. This example shows that the work area optimal size was between 2 kB and 4 kB as predominately the majority of SQLs were executed within that range without incurring extra passes.

PGA Aggr Target Histogram

- Optimal Executions are purely in-memory operations

Low Optimal	High Optimal	Total Execs	Optimal Execs	1-Pass Execs	M-Pass Execs
2K	4K	90,221	90,221	0	0
64K	128K	5	5	0	0
256K	512K	2	2	0	0
512K	1024K	31	31	0	0
1M	2M	5	5	0	0

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.6 PGA Memory Advisory

This section shows the advisory for sizing PGA. The optimal PGA target was between 203 MB and 407 MB, depending on the degree of estimated PGA cache hit ratio. This ratio stays at 100% beyond 407 MB for the PGA Target.

PGA Memory Advisory

- When using Auto Memory Mgmt, minimally choose a `pga_aggregate_target` value where Estd PGA Overalloc Count is 0

PGA Target Est (MB)	Size Factor	W/A MB Processed	Estd Extra W/A MB Read/ Written to Disk	Estd PGA Cache Hit %	Estd PGA Overalloc Count
203	0.13	3,668.37	207.62	95.00	0
407	0.25	3,668.37	0.00	100.00	0
814	0.50	3,668.37	0.00	100.00	0
1,220	0.75	3,668.37	0.00	100.00	0
1,627	1.00	3,668.37	0.00	100.00	0
1,952	1.20	3,668.37	0.00	100.00	0
2,278	1.40	3,668.37	0.00	100.00	0
2,603	1.60	3,668.37	0.00	100.00	0
2,929	1.80	3,668.37	0.00	100.00	0
3,254	2.00	3,668.37	0.00	100.00	0
4,881	3.00	3,668.37	0.00	100.00	0
6,508	4.00	3,668.37	0.00	100.00	0
9,762	6.00	3,668.37	0.00	100.00	0
13,016	8.00	3,668.37	0.00	100.00	0

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.7 Shared Pool Advisory

Shared Pool Advisory tries to correlate the shared pool size with the estimated library cache memory object hits. In this example, it seems that the latter was independent of the former. This is a good example that not every piece of information in an AWR report is worth your time to look at. It is there simply because AWR generates so much information. Many sections simply imply that there are no performance issues there and they should be ignored in a real world Oracle performance troubleshooting scenario. However, I have found that going over all the sections of an AWR report is very educational in terms of getting to know every bit of Oracle from the performance perspective.

Shared Pool Advisory

- SP: Shared Pool Est LC: Estimated Library Cache Factr: Factor
- Note there is often a 1:Many correlation between a single logical object in the Library Cache, and the physical number of memory objects associated with it. Therefore comparing the number of Lib Cache objects (e.g. in v\$librarycache), with the number of Lib Cache Memory Objects is invalid.

Shared Pool Size(M)	SP Size Factr	Est LC Size (M)	Est LC Mem Obj	Est LC Time Saved (s)	Est LC Time Saved Factr	Est LC Load Time (s)	Est LC Load Time Factr	Est LC Mem Obj Hits
220	0.89	30	3,827	31,780	1.00	1,206	1.01	1,940,343
248	1.00	57	6,411	31,793	1.00	1,193	1.00	1,945,749
276	1.11	84	9,088	31,802	1.00	1,184	0.99	1,950,066
304	1.23	111	12,783	31,808	1.00	1,178	0.99	1,953,353
332	1.34	138	17,618	31,812	1.00	1,174	0.98	1,956,045
360	1.45	165	20,028	31,816	1.00	1,170	0.98	1,958,380
388	1.56	192	23,770	31,819	1.00	1,167	0.98	1,960,346
416	1.68	219	25,393	31,821	1.00	1,165	0.98	1,961,867
444	1.79	246	26,962	31,822	1.00	1,164	0.98	1,962,931
472	1.90	273	28,662	31,823	1.00	1,163	0.97	1,963,620
500	2.02	300	30,504	31,824	1.00	1,162	0.97	1,964,027

[Back to Advisory Statistics](#)
[Back to Top](#)

11.10.8 SGA Target Advisory

SGA Target Advisory hints at an optimal SGA Target Size. This example hints that beyond 768 MB, the benefits with reduced number of physical reads start to diminish. Therefore an SGA target size of around 800 MB would be adequate.

SGA Target Advisory

SGA Target Size (M)	SGA Size Factor	Est DB Time (s)	Est Physical Reads
512	0.50	44,286	114,150
768	0.75	44,206	96,608
1,024	1.00	44,184	94,059
1,280	1.25	44,180	92,987
1,536	1.50	44,175	92,300
1,792	1.75	44,166	90,654
2,048	2.00	44,166	90,645

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.9 Streams Pool Advisory

Streams Pool Advisory hints of a desirable streams pool size. In this example, all zero values beyond 4 MB indicate that a 4 MB streams pool size is adequate. Note the following table was trimmed to save space.

Streams Pool Advisory

Size for Est (MB)	Size Factor	Est Spill Count	Est Spill Time (s)	Est Unspill Count	Est Unspill Time (s)
4	0.50	1,092	150	0	0
8	1.00	0	0	0	0
12	1.50	0	0	0	0
16	2.00	0	0	0	0
...	...	...	...	...	...

[Back to Advisory Statistics](#)

[Back to Top](#)

11.10.10 Java Pool Advisory

With this example, there were no Java stored procedures running inside Oracle. Therefore, there is no data here.

Java Pool Advisory

No data exists for this section of the report.

[Back to Advisory Statistics](#)

[Back to Top](#)

11.11 WAIT STATISTICS

The Wait Statistics section provides information on buffer wait and enqueue activity. Looking at all the metrics measured by wait time, we can conclude that there was neither buffer contention nor enqueue contention.

Wait Statistics

- [Buffer Wait Statistics](#)
- [Enqueue Activity](#)

[Back to Top](#)

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	13,439	17	1
1st level bmb	1,287	1	1
undo header	878	0	0
undo block	781	0	0
segment header	135	0	0
2nd level bmb	46	0	0
file header block	1	0	0

[Back to Wait Statistics](#)

[Back to Top](#)

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	425	425	0	425	1	1.43
TX-Transaction	152,223	152,262	0	93	0	3.20
HW-Segment High Water Mark	2,759	2,759	0	143	0	1.16
FB-Format Block	1,039	1,039	0	28	0	0.36
SQ-Sequence Cache	45	45	0	1	0	2.00

[Back to Wait Statistics](#)

[Back to Top](#)

11.13 LATCH STATISTICS

What is an Oracle latch? As explained in Chapter 10, a latch is a serialization mechanism (or a lock) to protect shared memory in SGA. When a latch is acquired for an area of an SGA, that area can only be accessed or modified by the current process or the owner of the latch until the latch is released. Latches operate very quickly—typically in nanosecond level. Like an enqueue, a latch is also a kind of lock, but it operates on shared memory rather than data as an enqueue does. Given the multifarious operations on shared memory or SGA, you can imagine that the types of latches are abundant as well.

If you want to learn more about latches, there are two more reliable sources about Oracle latches. One is Oracle’s Database Concepts document, which explains the concepts of latches and internal locks clearly, and the other is Oracle’s Performance Tuning document, which explains latch related wait events.

Let’s explore the latch related statistics next.

Latch Statistics

- [Latch Activity](#)
- [Latch Sleep Breakdown](#)
- [Latch Miss Sources](#)
- [Parent Latch Statistics](#)
- [Child Latch Statistics](#)

[Back to Top](#)

11.13.1 Latch Activity

This section lists all latches by name, followed by the metrics of Get Requests, Percent Get Miss, Average Sleeps/Miss, Wait Time, NoWait Requests, and Percent NoWait Miss. As noted below, there are two types of latches: willing-to-wait latches and no-wait latches. For willing-to-wait latches, an additional metric of Wait Time is given. This metric is an indicator of whether latch contention occurred with a specific latch. Let’s take a look at some of the most active latches as follows:

- *In memory undo latch (Get Requests: 1,697,452; NoWait Requests: 146,533).* This latch is related to Oracle’s in memory undo (IMU) optimization feature. In this example, the wait time was 0 so it was not a performance issue. If it was, it would show up as a top event of *latch: In-memory undo latch*, with non-zero wait time.
- *Cache buffers chains (Get Requests: 2,818,893,720; NoWait Requests: 349,580).* This is called CBC latch. A cache buffer chain is a linked list of cache buffers or a chain of buffers (the terms linked list and chain or hash chain are interchangeable). Apparently, operations on buffer chains such as searching for, adding, or removing a buffer from a chain might need to be serialized and

hence this latch. The wait event associated with this latch is *latch: cache buffers chains*, which could become a top event either because the chain was too long or the same buffer block was competed by many concurrent processes. If this happens, the recommended fix is to find the associated SQL statement and act accordingly.

- *Cache buffers LRU chain (Get Requests: 174,458; NoWait Requests: 728,752)*. This latch protects a buffer chain from being modified by more than one process when a buffer in the chain should be removed, based on the LRU algorithm or some similar algorithms. The associated wait event is *latch: cache buffers lru chain*. As with the CBC latches, the root cause lies in the SQL statement that incurs excessive logical and physical reads.
- *Checkpoint queue latch (Get Requests: 529,538; NoWait Requests: 52,899)*. A checkpoint queue, also known as a buffer checkpoint queue (BCQ), is a data structure (a queue) that contains dirty buffers that need to be written for a checkpoint. This latch is about coordinating access to those dirty buffers for point-checking purposes.
- *DML lock allocation (Get Requests: 573,979; NoWait Requests: 0)*. A DML lock is requested when a DML SQL Statement (INSERT, DELETE, UPDATE, ...) is to be executed, for the obvious reason that DML statements cause changes to data. There is one DML lock for each table modified in a transaction. The initialization parameters `dml_locks` defines the number of dml locks for an instance of Oracle database. If this parameter is set to zero, DML locks are disabled. This latch protects the free list data structure for DML locks.
- *Enqueue hash chains (Get Requests: 941,954; NoWait Requests: 61)*. Enqueue hash chains are linked list data structures that contain enqueue resources hashed with the resource type and identifiers. An enqueue resource is a database resource protected by an enqueue lock. Therefore, this latch simply protects enqueue hash chains from being manipulated inconsistently.
- *Enqueues (Get Requests: 208,427; NoWait Requests: 0)*. We already know that an enqueue is basically a lock that protects a database resource, which could be a row or a table or multiple such items. So this latch protects enqueues.
- *Library cache (Get Requests: 4,383,832; NoWait Requests: 1,066)*. This latch protects library cache concurrency.
- *Library cache lock (Get Requests: 277,529; NoWait Requests: 0)*. This latch protects library cache locks.
- *Library cache pin (Get Requests: 2,789,709; NoWait Requests: 11)*. A pin prevents an object loaded into memory from being aged out. So this latch protects library cache pins pinned on the underlying objects. Note that caching an object does not impose non-aging-out policy.
- *Redo allocation (Get Requests: 564,204; NoWait Requests: 694,714)*. This latch manages redo log buffer space allocation for redo entries.
- *Row cache objects (Get Requests: 600,292; NoWait Requests: 161)*. This latch protects the access to metadata objects in the data dictionary cache. If a contention for this latch occurs, the recommendation is to increase the shared pool size.

- *Session allocation (Get Requests: 316,143; NoWait Requests: 0).* This latch controls session object allocation. If you encounter session allocation latch contention, you might want to experiment with the Oracle initialization parameter named *sessions*.
- *Session idle bit (Get Requests: 5,848,885; NoWait Requests: 0).* This latch controls how the execution of a call from a session is updated between the two states of ACTIVE and INACTIVE.
- *Shared pool (Get Requests: 231,335; NoWait Requests: 0).* This latch controls how memory is allocated and freed in the shared pool.
- *Undo global data (Get Requests: 2,643,281; NoWait Requests: 0).* This latch guards the state information of the Undo or Rollback segments stored in the SGA.

Latch Activity

- "Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for willing-to-wait latch get requests
- "NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
- "Pct Misses" for both should be very close to 0.0

Latch Name	Get Requests	Pct Get Miss	Avg Slps /Miss	Wait Time (s)	NoWait Requests	Pct NoWait Miss
AWR Alerted Metric Element list	16,303	0.00		0	0	
Consistent RBA	87,414	0.00		0	0	
FOB s.o list latch	55	0.00		0	0	
In memory undo latch	1,697,452	0.55	0.00	0	146,533	0.00
JS queue state obj latch	21,480	0.00		0	0	
KMG MMAN ready and startup request latch	1,195	0.00		0	0	
KMG resize request state object freelist	6	0.00		0	0	
KTF sga latch	6	0.00		0	1,197	0.00
KWQMN job cache list latch	1	0.00		0	0	
KWQP Prop Status	2	0.00		0	0	
MQL Tracking Latch	0			0	71	0.00
Memory Management Latch	47	0.00		0	1,194	0.00
Memory Queue Message Subscriber #1	2	0.00		0	0	
Memory Queue Message Subscriber #2	2	0.00		0	0	
Memory Queue Message Subscriber #3	2	0.00		0	0	
Memory Queue Message Subscriber #4	2	0.00		0	0	
OS process	21	0.00		0	0	

OS process allocation	1,205	0.00		0	0	
OS process: request allocation	6	0.00		0	0	
PL/SQL warning settings	17	0.00		0	0	
SQL memory manager latch	1	0.00		0	1,184	0.00
SQL memory manager workarea list latch	79,973	0.00		0	0	
STREAMS LCR	2	0.00		0	0	
STREAMS Pool Advisor	2	0.00		0	2	0.00
Shared B-Tree	133	0.00		0	0	
Streams Generic	2	0.00		0	0	
active checkpoint queue latch	12,342	2.07	0.00	0	0	
active service list	6,463	0.00		0	1,197	0.00
archive control	2	0.00		0	0	
begin backup scn array	39,409	0.02	0.00	0	0	
buffer pool	8	0.00		0	0	
cache buffer handles	13,070	0.11	0.00	0	0	
cache buffers chains	2,818,893,720	0.92	0.00	1	349,580	0.70
cache buffers lru chain	174,458	0.04	0.00	0	728,752	0.21
cache table scan latch	0			0	2	0.00
channel handle pool latch	6	0.00		0	0	
channel operations parent latch	16,081	0.00		0	0	
checkpoint queue latch	529,538	0.00	0.00	0	52,899	0.00
client/application info	11	0.00		0	0	
compile environment latch	146,853	0.00		0	0	
dml lock allocation	573,979	0.55	0.00	0	0	
dummy allocation	6	0.00		0	0	
enqueue hash chains	941,954	0.13	0.00	0	61	0.00
enqueues	208,427	0.17	0.00	0	0	
event group latch	3	0.00		0	0	
file cache latch	923	0.00		0	0	
global KZLD latch for mem in SGA	1	0.00		0	0	
hash table column usage latch	182	0.00		0	6,554	0.00
hash table modification latch	60	0.00		0	0	
image handles of buffered messages latch	2	0.00		0	0	
job_queue_processes parameter latch	60	0.00		0	0	
kks stats	419	0.00		0	0	
ksuosstats global area	241	0.00		0	0	
ktm global data	11	0.00		0	0	

kwqbsgn:msghdr	12	0.00		0	0	
kwqbsn:qsga	130	0.00		0	2	0.00
kwqbsn:qxl	16	0.00		0	0	
lgwr LWN SCN	87,571	0.01	0.00	0	0	
library cache	4,383,832	0.47	0.00	0	1,066	0.75
library cache load lock	862	0.00		0	0	
library cache lock	277,529	0.52	0.00	0	0	
library cache lock allocation	581	0.00		0	0	
library cache pin	2,789,709	0.21	0.00	0	11	0.00
library cache pin allocation	232	0.00		0	0	
list of block allocation	1,031	0.00		0	0	
loader state object freelist	157,758	2.70	0.00	0	0	
logminer context allocation	1	0.00		0	0	
messages	300,332	0.12	0.00	0	0	
mostly latch-free SCN	87,889	0.26	0.00	0	0	
multiblock read objects	6	0.00		0	0	
nocode allocation latch	57	0.00		0	0	
object queue header heap	339	0.00		0	9,968	0.00
object queue header operation	299,339	0.00	0.00	0	0	
object stats modification	1	0.00		0	0	
parallel query alloc buffer	472	0.00		0	0	
parameter table allocation management	8	0.00		0	0	
post/wait queue	31	0.00		0	38	0.00
process allocation	6	0.00		0	3	0.00
process group creation	6	0.00		0	0	
qmn task queue latch	1,037	13.11	0.00	0	0	
redo allocation	564,204	0.15	0.00	0	694,714	1.13
redo copy	0			0	697,206	0.26
redo writing	284,986	0.08	0.00	0	0	
resmgr group change latch	3	0.00		0	0	
resmgr:actses active list	3	0.00		0	0	
resmgr:actses change group	1	0.00		0	0	
resmgr:free threads list	2	0.00		0	0	
resmgr:schema config	1	0.00		0	0	
row cache objects	600,292	0.94	0.00	0	161	0.00
rules engine rule set statistics	100	0.00		0	0	
sequence cache	2,597	0.12	0.00	0	0	
session allocation	316,143	0.25	0.00	0	0	
session idle bit	5,848,885	0.01	0.00	0	0	

session state list latch	12	0.00		0	0	
session switching	57	0.00		0	0	
session timer	1,197	0.00		0	0	
shared pool	231,335	0.05	0.12	0	0	
simulator hash latch	86,858,296	0.00	0.00	0	0	
simulator lru latch	86,812,502	0.83	0.00	0	41,880	0.01
slave class	2	0.00		0	0	
slave class create	8	12.50	1.00	0	0	
sort extent pool	86	0.00		0	0	
spilled messages latch	4	0.00		0	0	
state object free list	2	0.00		0	0	
statistics aggregation	112	0.00		0	0	
threshold alerts latch	138	0.00		0	0	
transaction allocation	1,025	0.00		0	0	
transaction branch allocation	57	0.00		0	0	
undo global data	2,643,281	0.26	0.00	0	0	
user lock	4	0.00		0	0	

[Back to Latch Statistics](#)

[Back to Top](#)

11.13.2 Latch Sleep Breakdown

This section simply lists the top latches ordered by the number of misses or sleeps. In this example, the cache buffers chains latch has the largest number of misses.

Latch Sleep Breakdown

- ordered by misses desc

Latch Name	Get Requests	Misses	Sleeps	Spin Gets	Sleep1	Sleep2	Sleep3
cache buffers chains	2,818,893,720	25,864,614	7,673	25,857,493	0	0	0
library cache	4,383,832	20,785	9	20,778	0	0	0
shared pool	231,335	106	13	94	0	0	0
slave class create	8	1	1	0	0	0	0

[Back to Latch Statistics](#)

[Back to Top](#)

11.13.3 Latch Miss Sources

This section lists sources of latch misses. One needs to further decipher the sources on an as-needed basis.

Latch Miss Sources

- only latches with sleeps are shown
- ordered by name, sleeps desc

Latch Name	Where	NoWait Misses	Sleeps	Waiter Sleeps
cache buffers chains	kcbgctr: kslbegin excl	0	14,345	12,259
cache buffers chains	kcbchg: kslbegin: bufs not pinned	0	5,523	3,872
cache buffers chains	kcbgctr: fast path	0	2,337	336
cache buffers chains	kcbrls: kslbegin	0	1,905	5,704
cache buffers chains	kcbgcur: kslbegin	0	254	86
cache buffers chains	kcbzwb	0	222	47
cache buffers chains	kcbget: pin buffer	0	149	68
cache buffers chains	kcbchg: kslbegin: call CR func	0	127	1,961
cache buffers chains	kcb_pre_apply: kcbhq61	0	8	255
cache buffers chains	kcb_post_apply: kcbhq62	0	5	6
cache buffers chains	kcbcge	0	4	219
cache buffers chains	kcbnlc	0	3	33
cache buffers chains	kcbbxsv	0	2	0
cache buffers chains	kcb_is_private	0	1	129
cache buffers chains	kcbbic2	0	1	0
cache buffers chains	kcbget: exchange rls	0	1	3
cache buffers chains	kcbnew: new latch again	0	1	0
cache buffers chains	kcbzgb: scan from tail. nowait	0	1	0
library cache	kglpndl: child: before processing	0	5	2
shared pool	kghalp	0	7	0
shared pool	kgfhfrunp: alloc: cursor dur	0	5	0
shared pool	kgfhfrunp: clatch: wait	0	4	0
shared pool	kgf: sim resz update	0	1	0
slave class create	ksvcreate	0	1	0

[Back to Latch Statistics](#)

[Back to Top](#)

11.13.4 Parent and Child Latch Statistics

These two sections have no data.

Parent Latch Statistics

No data exists for this section of the report.

[Back to Latch Statistics](#)

[Back to Top](#)

Child Latch Statistics

No data exists for this section of the report.

[Back to Latch Statistics](#)

[Back to Top](#)

11.14 SEGMENT STATISTICS

This part of the AWR report summarizes segment statistics, categorized by Logical Reads, Physical Reads, Row Lock Waits, ITL Waits, and Buffer Busy Waits. Each of the categories is discussed next.

Segment Statistics

- [Segments by Logical Reads](#)
- [Segments by Physical Reads](#)
- [Segments by Row Lock Waits](#)
- [Segments by ITL Waits](#)
- [Segments by Buffer Busy Waits](#)

[Back to Top](#)

11.14.1 Segments by Logical Reads

This part gives a glimpse of the logical reads occurred during the test period, listed in descending order. One can identify which owner/tablespace/object (table or index) incurred the largest number of logical reads, and then drill down to the associated objects that caused those logical reads. In this example, the first two table objects constitute over 90% of the total number of logical reads, which is actually good from performance troubleshooting point of view, because only two tables got involved.

Refer back to the section of SQL ordered by Gets, and one can immediately find out those three SQL statements that resulted from the first table T119 and those two SQL statements that resulted from the second table T62 below the T119 SQLs. By referring further back to the section of Top Five Timed Events, one can correlate the extremely high number of logical reads and those five SQLs to the high CPU time percentage of 96.4% there. This example is further studied later as a case study in Chapter 24, and we'll see how to cure this performance issue caused by excessive logical reads.

Segments by Logical Reads

- Total Logical Reads: 1,405,702,082
- Captured Segments account for 91.2% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Logical Reads	%Total
APPADMIN	APP	T119		TABLE	855,532,864	60.86
APPADMIN	APP	T62		TABLE	415,615,872	29.57
APPADMIN	APP	IT119		INDEX	3,912,704	0.28
APPADMIN	APP	IT62		INDEX	1,714,512	0.12
APPADMIN	APP	I119_400129200_2		INDEX	793,360	0.06

[Back to Segment Statistics](#)
[Back to Top](#)

11.14.2 Segments by Physical Reads

This part sorts segments by physical reads in descending order. It is seen that captured segments account for 9.8% of total, which implies that physical reads were spread across all the objects and no bottleneck occurred with physical reads.

Segments by Physical Reads

- Total Physical Reads: 265
- Captured Segments account for 9.8% of Total

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Physical Reads	%Total
APPADMIN	APP	ACTLINK_MAPPING_IND		INDEX	8	3.02
SYS	SYSAUX	WRH\$_PGASTAT_PK		INDEX	4	1.51
APPADMIN	APP	I103_179_1		INDEX	3	1.13
APPADMIN	APP	IT103		INDEX	3	1.13
APPADMIN	APP	IT61		INDEX	3	1.13

[Back to Segment Statistics](#)
[Back to Top](#)

11.14.3 Segments by Row Lock Waits

This part sorts segments by row lock waits in descending order. One should evaluate the number of row lock waits within the test period, which was one hour in this example. The largest number of row lock waits was 64 over a one-hour period, which implies that roughly on average only one row lock wait per minute occurred with that object. In addition, the % of Capture metric doesn't show skewed values, which indicates that row lock waits were not a bottleneck.

Segments by Row Lock Waits

- % of Capture shows % of row lock waits for each top segment compared
- with total row lock waits for all segments captured by the Snapshot

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Row Lock Waits	% of Capture
APPADMIN	APP	IH331		INDEX	64	15.46
APPADMIN	APP	I119_400129200_2		INDEX	49	11.84
APPADMIN	APP	I119_400129200_1		INDEX	30	7.25
APPADMIN	APP	IH119		INDEX	26	6.28
APPADMIN	APP	IT109		INDEX	25	6.04

[Back to Segment Statistics](#)

[Back to Top](#)

11.14.4 Segments by ITL Waits

Here the term *ITL* means Interested Transaction List. This part contains no data with this example. Typically a too small value for the initialization parameter INITRANS may cause undesirable amount of ITL waits, and its effects can be mitigated by increasing the value of INITRANS if needed.

Segments by ITL Waits

No data exists for this section of the report.

[Back to Segment Statistics](#)

[Back to Top](#)

11.14.5 Segments by Buffer Busy Waits

This part sorts segments by buffer busy waits. Similar to the previous section of Segments by Physical Reads, the values of the metric % of Capture were not obviously skewed, and therefore, the application was not bottlenecked by buffer waits.

Segments by Buffer Busy Waits

- % of Capture shows % of Buffer Busy Waits for each top segment compared
- with total Buffer Busy Waits for all segments captured by the Snapshot

Owner	Tablespace Name	Object Name	Subobject Name	Obj. Type	Buffer Busy Waits	% of Capture
APPADMIN	APP	IH331		INDEX	2,453	16.57
APPADMIN	APP	IT331		INDEX	1,643	11.10
APPADMIN	APP	SYS_LOB0000067881C00016\$\$		LOB	885	5.98
APPADMIN	APP	T331		TABLE	749	5.06
APPADMIN	APP	T119		TABLE	721	4.87

[Back to Segment Statistics](#)

[Back to Top](#)

11.15 DICTIONARY CACHE STATS

This section reports the dictionary cache activity statistics. Since the dictionary cache is rarely the bottleneck, one can safely ignore the information here.

Dictionary Cache Stats

- “Pct Misses” should be very low (< 2% in most cases)
- “Final Usage” is the number of cache entries being used

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage
dc_awr_control	60	0.00	0		2	1
dc_global_oids	60	1.67	0		0	153
dc_histogram_data	667	2.25	0		0	1,206
dc_histogram_defs	6,233	2.45	0		0	10,322
dc_object_grants	84	33.33	0		0	198
dc_object_ids	93,935	0.03	0		0	3,012
dc_objects	1,094	3.38	0		0	2,525
dc_profiles	1	0.00	0		0	1
dc_rollback_segments	1,300	0.00	0		0	46
dc_segments	5,167	0.68	0		708	3,057
dc_sequences	44	0.00	0		44	3
dc_tablespace_quotas	2,768	0.00	0		0	3
dc_tablespaces	4,090	0.00	0		0	6
dc_usernames	53	0.00	0		0	5
dc_users	85,855	0.00	0		0	33
outstanding_alerts	36	0.00	0		0	5

[Back to Top](#)

11.16 LIBRARY CACHE ACTIVITY

This section reports the library cache activity statistics. Note that the SQL AREA namespace was requested 23,768 times, far outnumbered the total of all other namespaces. The value of percent miss was 0.38% only for SQL AREA, which implies that SQL AREA was accessed efficiently.

Library Cache Activity

- “Pct Misses” should be very low

Namespace	Get Requests	Pct Miss	Pin Requests	Pct Miss	Reloads	Invali- dations
BODY	9	0.00	15	0.00	0	0
CLUSTER	9	0.00	16	0.00	0	0
INDEX	12	8.33	20	45.00	8	0
SQL AREA	23,768	0.38	1,310,599	0.05	242	2
TABLE/PROCEDURE	274	6.20	83,857	0.50	350	0

[Back to Top](#)

11.17 MEMORY STATISTICS

This section summarizes the memory statistics with three subsections of Process Memory Summary, SGA Memory Summary, and SGA breakdown difference, as shown below. Let’s take a look at each subsection next.

Memory Statistics

- [Process Memory Summary](#)
- [SGA Memory Summary](#)
- [SGA breakdown difference](#)

[Back to Top](#)

11.17.1 Process Memory Summary

This section summarizes process memory or PGA allocation by category. You can get a finer-granularity view of the PGA memory usage dynamically process by process from querying the V\$PROCESS_MEMORY as follows:

```
SELECT * FROM V$PROCESS_MEMORY;
```

Then you will see the columns of PID, SERIAL#, CATEGORY, ALLOCATED, USED, and MAX_ALLOCATED. Under the CATEGORY column, you will see Freeable, Other, SQL, and PL/SQL—the same categories as shown below. The Freeable memory is the amount of memory that can be freed back to OS. The Other category shows the amount of memory beyond the categories of SQL, PL/SQL, and Freeable. However, there is no easy way to break down the Other category.

Since the information given here was obtained at the two snaps, one can compare the begin and end memory numbers for each category to validate if memory leaks had occurred. Memory leaks are typically caused by programming errors that fail to de-allocate memory taken from the process heap. Memory leaks are more common with user transactions where user or session state is maintained but not fully de-allocated when the transaction is completed. In this example, it's obvious that no memory leaks occurred as the end memory was close to the begin memory for each category.

Process Memory Summary

- B: Begin snap E: End snap
- All rows below contain absolute values (i.e. not diffed over the interval)
- Max Alloc is Maximum PGA Allocation size at snapshot time
- Hist Max Alloc is the Historical Max Allocation for still-connected processes
- ordered by Begin/End snapshot, Alloc (MB) desc

	Category	Alloc (MB)	Used (MB)	Avg Alloc (MB)	Std Dev Alloc (MB)	Max Alloc (MB)	Hist Max Alloc (MB)	Num Proc	Num Alloc
B	Other	107.86		1.35	2.38	22	22	80	80
	Freeable	32.88	0.00	0.57	0.33	1		58	58
	SQL	8.80	4.27	0.12	0.14	0	38	72	71
	PL/SQL	0.26	0.09	0.00	0.01	0	0	80	80
E	Other	111.93		1.40	2.38	22	22	80	80
	Freeable	39.75	0.00	0.71	0.32	1		56	56
	SQL	9.04	4.35	0.13	0.14	0	38	72	71
	PL/SQL	0.26	0.09	0.00	0.01	0	0	80	80

[Back to Memory Statistics](#)
[Back to Top](#)

11.17.2 SGA Memory Summary

This section summarizes the SGA memory by regions. The most noteworthy region is the buffer cache, which took the largest chunk of the SGA. The Variable Size region is the amount of memory for various pools, as is shown next.

SGA Memory Summary

SGA regions	Begin Size (Bytes)	End Size (Bytes) (if different)
Database Buffers	771,751,936	784,334,848
Fixed Size	1,984,080	
Redo Buffers	10,592,256	
Variable Size	289,413,552	276,830,640

[Back to Memory Statistics](#)

[Back to Top](#)

11.17.3 SGA Breakdown Difference

This section shows the SGA breakdowns by pool for the two snaps. The pools listed here include java pool, large pool, shared pool, and streams pool. One can further see the sub-regions of library cache, row cache, sql area, and so on, in the shared pool. Many of the sub-regions in the shared pool are oracular and should be pursued on an as-needed basis only.

SGA Breakdown Difference

- ordered by Pool, Name
- N/A value for Begin MB or End MB indicates the size of that Pool/Name was insignificant, or zero in that snapshot

Pool	Name	Begin MB	End MB	% Diff
java	free memory	4.00	4.00	0.00
large	PX msg pool	1.03	1.03	0.00
large	free memory	2.97	2.97	0.00
shared	ASH buffers	20.48	20.48	0.00
shared	CCursor	14.68	15.47	5.37
shared	Checkpoint queue	3.13	3.13	0.00
shared	Heap0: KGL	5.51	5.79	5.05
shared	KCB Table Scan Buffer	3.80	3.80	0.00
shared	KGLS heap		4.11	
shared	KQR L PO	2.43	2.47	1.53
shared	KQR M PO	8.57	8.20	-4.28
shared	KSFD SGA I/O b	3.79	3.79	0.00
shared	PCursor	7.25	7.45	2.69
shared	PL/SQL MPCODE	3.46	2.59	-25.32
shared	db_block_hash_buckets	4.20	4.20	0.00

shared	dbwriter coalesce buffer	4.02	4.02	0.00
shared	free memory	38.60	22.48	-41.76
shared	kglsim hash table bkts	4.00	4.00	0.00
shared	kglsim heap	7.16	7.16	0.00
shared	kglsim object batch	9.14	9.15	0.12
shared	library cache	22.78	22.80	0.09
shared	private strands		2.27	
shared	row cache	7.13	7.13	0.00
shared	sql area	25.80	26.85	4.06
streams	KGH: NO ACCESS	3.98		-100.00
streams	free memory	7.96	7.99	0.38
	buffer_cache	736.00	748.00	1.63
	fixed_sga	1.89	1.89	0.00
	log_buffer	10.10	10.10	0.00

[Back to Memory Statistics](#)

[Back to Top](#)

11.18 STREAMS STATISTICS

This section summarizes the Streams Statistics. Although it contains no data for most categories, it is included here for the sake of completeness.

Streams Statistics

- [Streams CPU/IO Usage](#)
- [Streams Capture](#)
- [Streams Apply](#)
- [Buffered Queues](#)
- [Buffered Subscribers](#)
- [Rule Set](#)

[Back to Top](#)

Streams CPU/IO Usage

- Streams processes ordered by CPU usage
- CPU and I/O Time in micro seconds

Session Type	CPU Time	User I/O Time	Sys I/O Time
QMON Slaves	36,753	0	0
QMON Coordinator	32,586	0	0

[Back to Streams Statistics](#)
[Back to Top](#)

Streams Capture

No data exists for this section of the report.

[Back to Streams Statistics](#)
[Back to Top](#)

Streams Apply

No data exists for this section of the report.

[Back to Streams Statistics](#)
[Back to Top](#)

Buffered Queues

No data exists for this section of the report.

[Back to Streams Statistics](#)
[Back to Top](#)

Buffered Subscribers

No data exists for this section of the report.

[Back to Streams Statistics](#)
[Back to Top](#)

Rule Set

- Rule Sets ordered by Evaluations

Ruleset Name	Evals	Fast Evals	SQL Execs	CPU Time	Elapsed Time
SYS.ALERT_QUE_R	0	0	0	0	0

[Back to Streams Statistics](#)
[Back to Top](#)

11.19 RESOURCE LIMIT STATS

This section contains no data and is included for the sake of completeness only.

Resource Limit Stats

No data exists for this section of the report.

[Back to Top](#)

11.20 init.ora PARAMETERS

This final section of an AWR report summarizes some of the initialization parameters (dynamic) that are either changed during the snapshot period or set differently from the default values. For this example, the most notable ones from the performance perspective are as follows:

- cursor_sharing: SIMILAR
- db_block_size: 8192
- open_cursors: 500
- pga_aggregate_target: 1,706,033,152
- processes: 150
- session_cached_cursors: 100
- sga_target: 1,073,741,824
- undo_management: AUTO

init.ora Parameters

Parameter Name	Begin value	End value (if different)
_wait_for_sync	FALSE	
compatible	10.2.0.1.0	
cursor_sharing	SIMILAR	
db_block_size	8192	
db_domain		
db_file_multiblock_read_count	16	
db_name	ObStore	
job_queue_processes	10	
open_cursors	500	
pga_aggregate_target	1706033152	
processes	150	
remote_login_passwordfile	EXCLUSIVE	
session_cached_cursors	100	
sga_target	1073741824	
undo_management	AUTO	
undo_tablespace	UNDOObs	
user_dump_dest	/data3/obs/udump	

[Back to Top](#)

End of Report

If you have followed along this far, thanks and congratulations! Although an AWR report is so comprehensive, I have found out that to some extent that's why Oracle has stayed at the top as one of the most highly performing and scalable database platforms—thanks to so many elaborate performance- and scalability-oriented algorithms and features built into it. I have helped solve many Oracle performance and scalability challenges in real use cases using AWR reports, combined with my understanding of queuing theory and software performance and scalability in general. Dealing with a software performance and scalability issue effectively and efficiently requires more scientific, logic, and disciplined thinking than wild guesses. And I hope you would enjoy it as much as I do.

11.21 SUMMARY

In this chapter, we anatomized a real-product-based AWR report out of one of my working experiences with an Oracle-based enterprise application. Numerous metrics and concepts have been explained, which offered a full-gamut overview of what kind of useful information Oracle AWR reports have to offer in facilitating Oracle performance and scalability diagnosing efforts. Although this might be a tough chapter for those who have not gotten a chance to learn much about a database or Oracle specifically, AWR reports are foundationally important in optimizing and tuning Oracle in almost every Oracle performance and scalability situation, as will be demonstrated in the remainder of this text.

RECOMMENDED READING

Appendix E, “Statistics Descriptions,” of the following Oracle product document discusses various database statistics in more detail:

Oracle Corp, *Oracle Database Reference*, 11g Release 1 (11.1) B28320-03 (1132 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28320.pdf.

Part III, “Optimizing Instance Performance,” of the following Oracle product document covers more about AWR in detail:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

Refer to the following text for some quantitative case studies of diagnosing and resolving Oracle-based enterprise application performance and scalability problems using AWR reports:

H. Liu, *Software Performance and Scalability: a Quantitative Approach*, John Wiley & Sons, Hoboken, 2009.

EXERCISES

- 11.1 If you are new to AWR reports and have access to an Oracle server, generate an AWR report and go through each section to get familiar with the metrics and concepts contained in various parts of the report.
- 11.2 How do you determine the snapshot covering period meaningfully for an AWR report? What use scenarios favor a smaller period and what use scenarios favor a larger period based on the nature of an Oracle-based application?
- 11.3 Describe the differences among locks, latches, and enqueues in Oracle’s context.
- 11.4 Will you rely more on absolute or relative statistical metrics to determine bottlenecks? Give a few examples to help prove your point.
- 11.5 If you see the utilization of a resource is high, is it proper to jump to a conclusion immediately that the underlying hardware is the problem and needs to be upgraded? In what circumstances is it adequate to conclude that the associated hardware is the bottleneck and needs to be upgraded?
- 11.6 How do you make a judgment on whether IO is or isn’t the bottleneck based on the IO stats information in an AWR report?
- 11.7 There is one particular type of hardware resource that an AWR report doesn’t provide much information about it. What is it?
- 11.8 If you are knowledgeable about other database platforms, compare Oracle’s AWR tool with similar tools of other database platforms if any.
- 11.9 Assuming that you are seeing a buffer cache size of zero MB from an AWR report as shown below, how would you figure out if the buffer cache size was indeed zero or it is just a format error? How would you verify based on the information contained in the other sections of the AWR report? Use the AWR report presented in this section for this exercise.

Cache Sizes

	Begin	End		
Buffer Cache:	0M	0M	Std Block Size:	8K
Shared Pool Size:	260M	248M	Log Buffer:	10,344K

12

Oracle Advanced Features and Options

Art is Man's nature. Nature is God's art.

—James Bailey

The last chapter discussed in detail what information an AWR report contains to facilitate troubleshooting Oracle performance and scalability issues. The concepts and metrics introduced therein reflect what an elaborate infrastructure Oracle has built in behind the scene under the hook of an AWR report. In this chapter, I'd like to offer a high-level overview of all major Oracle features since 8i through 11g so that we see not only trees but also forests in Oracle's context. This would also give us a complete view of how Oracle has evolved architecturally with time.

A good start point is with Oracle 8i, which still is not too distant as that is when the Internet started to take off and so did Oracle.

12.1 ORACLE 8i NEW FEATURES

With the advent of the Internet age in late 1980's, Oracle quickly embraced the Web technologies in its database management product and so named it 8i with

the letter “i” reflecting its affinity to the Internet. The enhanced/new features Oracle introduced in 8i include the following:

- Java
- Oracle interMedia, Spatial, Time Series, and Visual Image Retrieval
- Oracle Parallel Server
- Optimizer Plan Stability
- Locally Managed Tablespaces
- Online Index Creation and Rebuild
- Online Read-Only Tablespaces
- Temporary Tables
- Non-Blocking OCI
- Function-Based Indexes
- Logical ROWIDs
- Enhanced Partitioning
- Connection Load Balancing
- Client Load Balancing
- Enterprise Manager

Let’s briefly cover each of these new features next.

12.1.1 Java

As a high-level programming language, Java was embraced quickly as the language for Internet computing in late 1980’s. In addition to providing GUI-based, development-oriented tools to support Web programming, at the infrastructural level, Oracle responded accordingly with four JAVA API packages:

- *JAVA AQ API*. This is a Java API package for developing Java-based messaging applications using the Oracle AQ (Advanced Queueing). The package was made available as an `aqapi.jar` file in `$ORACLE_HOME/rdbms/jlib` (`$ORACLE_HOME` represents the Oracle installation directory. Note: I checked my Oracle 11g R2 install, and this jar file still is there).

The way Oracle AQ works is that first you create an AQ user/schema, then a queue table with object type Messages, and after that you can use JDBC to connect to the database and start enqueueing/dequeueing messages with JAVA AQ API. In this case, Oracle database is used as a repository for asynchronous queuing. With Oracle AQ, both point-to-point and publish/subscribe messaging models are supported. This is a powerful mechanism for external applications to communicate with each other by exchanging messages so that enterprise applications can collaborate to fulfill a business service rather than staying as isolated islands. This was once part of a hot market named EAI (Enterprise Application Integration). This package supports both the administrative and

operational features of a typical messaging server. Consult the relevant Oracle documentation if you want to learn more about it.

Note that in Oracle 11g, Oracle AQ is called Streams Advanced Queuing. It's also the foundation for Oracle Data Guard technology, which aids in establishing and maintaining standby databases as alternative repositories to the primary production databases. In the AWR report discussed in the previous chapter, some STREAMS- and AQ-related metrics appeared but with no data associated since this feature was not used in that database.

- *JAVA JMS API*. JMS (Java messaging services) is a standard message protocol for developing Java-based messaging applications. Since it's a specification, it can be implemented by any parties and offered as a product. Oracle implemented JMS as an interface to its proprietary AQ product, so to some extent, we can consider that Oracle JMS API is an extension to its AQ API.
- *oracle.ODCI*. This ODCI (Oracle Data Cartridge Interface) package implements an object model as an extension to the commonly used relational model. It's enabled in two jar files: ODCI.jar and CartridgeServices.jar. These two jar files still exist in Oracle 11g's jlib directory together with the aqapi.jar file mentioned above. However, it seems that it has not become a mainstream feature so we won't cover it further.
- *oracle.xml.parser API*. This was an XML parser API that supports all XML-related operations such as generation, transformation, parsing, storing and retrieving, and so on. This feature was evolved into a new feature named XML DB in Oracle 9i.

In addition to these Java API packages, Oracle once developed Oracle Web Server, Oracle Application Server, and so on, to support various Java-based Web applications. But with the acquisitions of BEA/WebLogic and Sun Microsystems during the past few years, these older products probably would fade away gradually with time. However, because of its generality, Oracle's JDBC driver has survived and become more and more robust as an indispensable interface for Java applications to access Oracle databases.

12.1.2 Oracle interMedia, Spatial, Time Series, and Visual Image Retrieval

This package extended Oracle's main functionality of storing structured data to supporting unstructured data. Each unstructured data type is described as follows:

- *interMedia*. This data type was designed for managing text, documents, images, audio, video, and geographical location. This feature was called for when the computing paradigm shifted from text-centric to media-centric, such as messaging, online music/video streaming, and so on.
- *Oracle Spatial*. The spatial feature was designed for managing spatial data such as GIS (geographical information system) data, CAD (computer-aided design) data and CAM (computer-aided manufacturing) data, and any other types of location-based data. Google Earth is a good example of the usage of spatial data.

- *Time Series*. This feature allows timestamp-based data to be stored in an Oracle database. It provides three kinds of functions:
 1. Calendar functions as a mechanism for defining time-related operations such as scheduling, appointment-making, and so on.
 2. Times series functions for analyzing time series data such as calculating moving average, cumulative sum, and so on, typically encountered in data provisioning and data warehousing applications,
 3. Time scaling functions such as transforming from one time scale to another, for example, from daily aggregation to quarterly summaries.
- *Visual Image Retrieval*. This feature facilitates content-based retrieval for images stored in Oracle 8i. Content-based information retrieval uses loosely defined query criteria like “find me something that looks like . . .” and so on. One can also query images with such visual criteria as color, pattern, texture, and so on. Typical applications that require this feature may include online photo stores, online retail stores, online ads agencies, online search providers, and so on.

12.1.3 Oracle Parallel Server

This was Oracle’s first attempt to provide parallel database processing capability—not only for performance and scalability but also for high availability. This technology had evolved and renamed as Oracle RAC (Real Application Cluster) since 9i. However, the most basic architecture remained: Parallel access to the same shared underlying database was implemented with multiple instances, each of which has its own independent processes and memory cache, as described in Chapter 5.

12.1.4 Optimizer Plan Stability

As will be introduced in Part Three, “Optimizing Oracle Performance and Scalability,” Oracle executes each SQL statement by first parsing it and then choosing an optimal execution plan to execute it. An execution plan is like a blueprint for a SQL execution to achieve the best possible performance. Typically, there might be numerous execution plans for a complicated SQL statement, and what execution plan the Oracle optimizer would choose depends on many factors, so a same SQL could be executed with different execution plans either in the same environment or different environments.

With the feature of Optimizer Plan Stability, an execution plan derived after sufficient and rigorous testing could be stored as an outline, and then shipped with the product. Then the same execution plan will be used even without having to invoke the CBO (cost-based optimizer) at each invocation of the same SQL across an entire customer base. This provides both a performance advantage and simplified maintenance to some extent. However, this may not work with SQLs created dynamically on the fly.

12.1.5 Locally Managed Tablespaces

Prior to Oracle 8i, management of a tablespace’s free and used extents was performed on data dictionary tables. With local tablespace management, all information on free

and used extents is stored and tracked in the same tablespace itself using bitmaps. Since bitmaps manage space allocation very efficiently and the dependence on data dictionary is eliminated, a series of benefits are achieved such as better performance, reliability, and reduced disk fragmentation.

12.1.6 Online Index Creation and Rebuild

Prior to 8i, indexes could be created or built off-line only. By “off-line,” it means that a lock is forced on the base table, which prevents concurrent DML operations on the table. This poses a huge problem if it takes a long time from seconds to hours to create or rebuild an index. Oracle lifted this constraint in 8i with online index creation and rebuild, which does not lock the table and queries on the table are not blocked. According to Oracle, this feature works for partitioned or non-partitioned B*-tree indexes, including index-organized tables.

12.1.7 Online Read-Only Tablespaces

Oracle 8i offered an option of placing a tablespace in read-only mode when there are no transactions outstanding in that tablespace alone. The prior versions required no outstanding transactions in the entire database. This could help improve performance to some extent since some locking/unlocking-related operations are saved. However, my advice is that features like this need to be tested thoroughly before committing to implementing them. They may or may not help, depending on your application. Also, this feature is more relevant to DSS (decision support system) than OLTP (online processing systems) applications.

12.1.8 Temporary Tables

Oracle 8i introduced temporary tables that could be used to store session-specific or transaction-specific data. A temporary table starts with empty, stores intermediate results, and then becomes empty again at the end of a session or transaction. Temporary tables are created in the user's temporary tablespaces. I'll offer a quantitative case study later to help illustrate how temporary tables can help improve the performance of an Oracle-based application.

12.1.9 Non-Blocking OCI (Oracle Call Interface)

Oracle Call Interface (OCI) is a comprehensive, high-performance, native C language based interface to Oracle databases. It was introduced in Oracle 6 originally. Oracle 8i introduced non-blocking OCI to help improve the throughput of OCI calls. With non-blocking OCIs, Oracle would not block on an OCI call so that the server would be available to accept and process other calls. The non-blocking OCI call feature was implemented with non-blocking polling mode, which can be set for an entire application or at the individual call level.

12.1.10 Function-Based Indexes

Oracle 8i started offering function-based indexes that allow the columns of a table with functions applied to be indexed. Typical functions are arithmetic expressions, data type conversion functions, expressions that contain Java or PL/SQL functions, package functions, C callouts, or SQL functions, and so on. I once had an experience in applying function-based indexing to solving a performance issue with a real product that was based on WebLogic on Oracle 9i, and it was very effective. However, I lost the data so I could not make it more quantitative here. The bottom line is that function-based indexes are as effective as regular indexes. So apply function-based indexes to helping solve your performance and scalability issues whenever applicable.

12.1.11 Logical ROWIDs

An Oracle `rowid` is a pointer that points to the location of the row it points to in the database. This provides the fastest possible access to a given row in a given table, as a `rowid` contains the physical address of a row so that the row can be retrieved in a single block access. However, such physical `rowids` apply only to physical objects such as ordinary tables (excluding index-organized tables), clustered tables, table partitions and subpartitions, indexes, and index partitions and subpartitions. In index-organized tables, `rowids` do not have fixed physical addresses. To solve this problem, Oracle 8i introduced logical `rowids` that are based on the table's primary keys of an index-organized table. Physical `ROWIDs` and logical `ROWIDs` were united under the same universal `ROWID` type named `UROWID` so that a column of the `UROWID` datatype can store all kinds of `rowids`—even the `rowids` of foreign tables in non-Oracle databases accessed through a gateway.

12.1.12 Enhanced Partitioning

Prior to Oracle 8i, only one partitioning method of *range partitioning* was available. Oracle 8i introduced two more methods: *hash* and *composite* partitioning methods. Before we explain how these three partitioning methods work, let's first have a general understanding of what partitioning is about.

Partitioning is a method to break a large entity into smaller pieces called partitions. In Oracle's context, such entities could be tables, indexes, or index-organized tables. From an administrator's perspective, a partitioned entity can be managed either collectively or individually. However, from an application's perspective, a partitioned entity is identical to the original, non-partitioned entity, which means that same SQL statements still apply as if the entity was not partitioned.

Partitioning could be done in three different methods in Oracle 8i as described above. Each of these methods is introduced below:

- *Range Partitioning.* This is the most common type of partitioning that is based on the ranges of values of the partitioning key established for each partition. Partitioning is achieved by specifying a `VALUES LESS THAN` clause, which

specifies a non-inclusive upper bound for the partitions. All partitions, except the first partition, assume implicitly a lower bound specified by the **VALUES LESS THAN** clause of the preceding partition. For example, if a table is partitioned on an integer type column with values ranging from 0 to 999, and it is partitioned with a **VALUES LESS THAN 100** into 10 partitions, then the first partition would take the sub-range of 0 to 99 (non-inclusive upper bound), second sub-range 100 to 199 (assumed lower bound), . . . , and the tenth sub-range of 900 to 999.

- *Hash Partitioning.* This method can be applied to a partitioning key that is not historical or has no obvious sequence order to follow. By applying a hash algorithm to an irregular partitioning key, data can be made evenly distributed among all partitions.
- *Composite Partitioning.* This method essentially partitions twice: the first round might use range or hash partitioning, and the second round applies partitioning on the partitions from the first round.

Partitioning is about dividing a larger entity into smaller sub-entities. A question is when to partition a table or index. Oracle recommends a “ballpark” table size threshold of 2 GB, but it depends on many factors so that only your testing with your application can answer the question of when to partition in your situation.

12.1.13 Connection Load Balancing

Connection load balancing was a feature aimed at optimizing connection performance and load balancing among multiple instances deployed on multiple nodes in an Oracle Parallel Server environment. The listener directs a client to the least-loaded instance on the least-loaded node, thus achieving load balancing.

12.1.14 Client Load Balancing

This was a feature applicable when more than one listener supports a service. In such a situation, a client can send requests to the various listeners randomly to achieve loading balancing. I haven’t encountered a situation where the listener was the bottleneck and multiple listeners were needed, but the option is there.

12.1.15 Oracle Enterprise Manager

This is the OEMJC (Oracle enterprise manager Java console) that I have mentioned frequently throughout the first part of this text, so you should be very familiar with it. It’s a very easy-to-use tool, but has been phased out in 11g. However, you can still connect it to 11g Oracle servers.

12.2 ORACLE 9i NEW FEATURES

This section introduces Oracle 9i new features at the heels of the preceding section, which introduced main new features of Oracle 8i. To set the expectation properly,

I have to mention that it's beyond the scope of this text to give a full coverage about how to implement each new feature discussed here. The intention is to help you understand conceptually all major new features of a specific version of Oracle, especially in the context of performance and scalability, so that if some features interest you or apply to your product you can dive deeper by looking up more detailed Oracle documentations or other relevant texts.

Some Oracle 9i new features that will be briefly introduced in this chapter include the following:

- Real Application Cluster (RAC)
- Data Guard
- Performance Tuning Intelligent Advisors
- Actual Operation-Level Query Statistics
- Dynamic Sampling of Optimizer Statistics
- Cloning Production Database with Oracle Enterprise Manager
- Renaming Columns and Constraints
- Dynamic Memory Pools
- Flashback Query
- List Partitioning

Let's begin with Oracle RAC first in the next section.

12.2.1 Real Application Clusters (RAC)

Oracle 9i RAC was aimed at bringing Oracle performance, scalability, and availability to an unprecedented level to meet more and more demanding business requirements. Conceptually, an Oracle RAC operates as a single system to the application while providing transparent performance and scalability under the hook by forming a multi-node cluster that all instances on the clustered nodes share the same database at the bottom level (note: a cluster is a group of independent nodes or servers that function collaboratively as a single system). Apparently, this is a very complex product that challenges state-of-the-art software and hardware technologies.

Figure 12.1 illustrates a typical Oracle RAC deployment. In this scenario, it starts with Web and application servers, and then comes down to the RAC setup. A RAC database deployed on SAN storage are shared by n nodes with an Oracle instance running on each node. Theoretically, a RAC can scale up to 64 nodes, for example, on Linux; but in reality, smaller deployments may have only 2 to 3 nodes. The other limitation is that if the bottleneck is somewhere else (e.g., disk IO or networking), then adding more nodes will not help in terms of performance and scalability other than increasing availability.

There are two critical pieces of technologies associated with a RAC: (1) a fast inter-node Interconnect bus that allows all nodes to exchange *heartbeat* messages so that they can stay synchronized with each other, and (2) a shared cache system known as

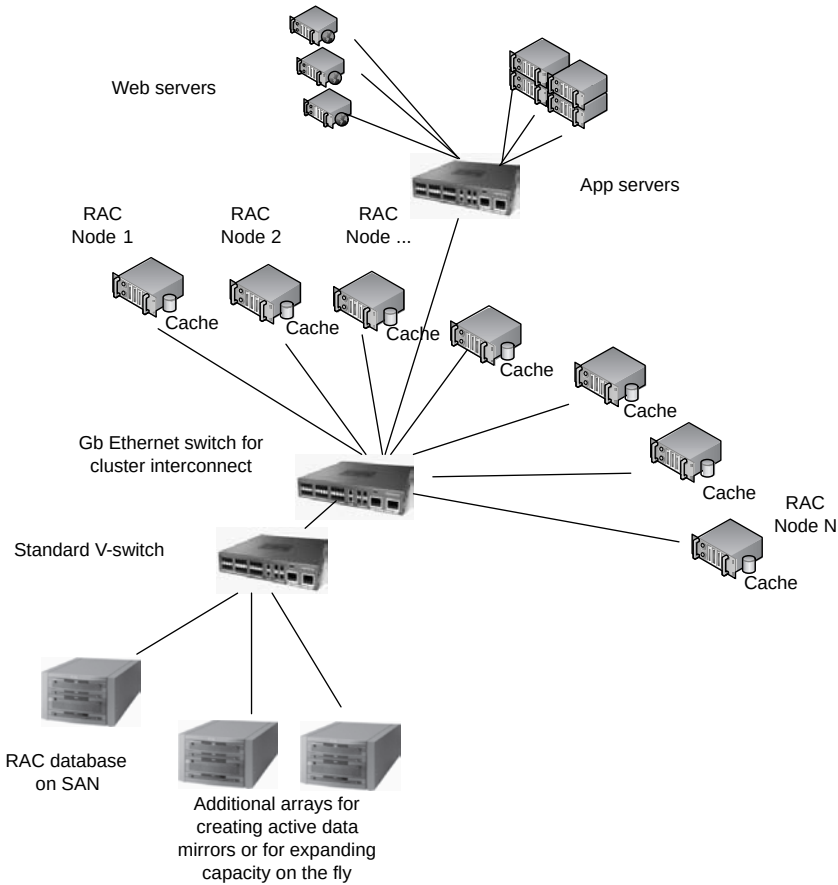


Figure 12.1 An Oracle RAC setup with N RAC nodes and a RAC database on SAN.

Cache Fusion, which uses the collective caches of all nodes in the cluster to satisfy user requests. Detailed discussions on these technologies are beyond the scope of this text, and if you are interested in knowing more about how to actually set up and configure a RAC environment, you should consult Oracle's RAC-related documents or RAC-dedicated texts.

The benefits of the Oracle RAC feature include the following:

- **Elastic Scalability.** You can scale up or down by adding or removing nodes based on your needs. On some platforms, you can even add nodes dynamically while your cluster is running, just like the hot-swappable disk technology used in RAID5s.
- **High Availability.** High availability benefit refers to the fact that if one node goes down, other nodes can continue to work while the affected node is being repaired. This non-stopping property of a RAC setup makes it very appealing for mission-critical business applications.

- *Transparency.* The concept of transparency means that a RAC is presented to an application as a single system and there is no need to modify an application in order for it to work on a RAC. This is a very desirable benefit as it simplifies application deployment on a RAC tremendously.
- *Buffer Cache Management.* With the Cache Fusion technologies, buffer caches of all nodes are coordinated at the system level. This minimizes the probability of fetching data from disks significantly, thus improving system performance. An RAC uses the Global Cache Service (GCS) to orchestrate all buffer cache related operations. In addition, a RAC uses the Global Enqueue Service (GES) to manage inter-node communications to keep all nodes in sync.
- *Improved Concurrency and Throughput with Row Locking and Multi-version Read Consistency.* Contention caused by competing for the same resource is often a performance hindrance. With the row locking optimization technique, operations can be performed on multiple, independent rows without having to wait for each other. Multi-version read consistency allows read and write operations to be performed concurrently without blocking each other. It's implemented by creating snapshots or read consistent versions of blocks that have been dirtied by a transaction but have not been committed.

For Oracle RAC performance and scalability best practices, refer to the texts recommended at the end of this chapter. Next, we introduce Oracle's Data Guard feature in 9i.

12.2.2 Data Guard

There are two common stringent requirements for enterprise applications: (1) the system must be up and running all the year around, and (2) one cannot afford to lose data stored in databases. The first requirement is translated into the high availability (HA) requirement. Typically, HA is measured with the number of digit 9's. For example, two 9's or 99% availability corresponds to a $365 \times (1 - 0.99) = 3.65$ day down-time a year, three 9's or 99.9% availability corresponds to an 8.76 hour down-time a year, four 9's or 99.99% corresponds to a 52.56 minute down-time a year, and finally five 9's or 99.999% corresponds to a 5.256 minute down-time a year. In reality, three 9's are common, and five 9's are not uncommon with really mission-critical enterprise systems.

Oracle's Data Guard feature is designed just around the above two requirements. It ensures high availability, data protection, and disaster recovery for enterprise systems and enterprise data. At this point, let's digress a little to explain how high availability is achieved in general.

High availability typically is achieved via clustering, which makes multiple copies of a system so if one goes down another can take over and the system can continue to function. In this regard, there are two clustering modes: active/active and active/passive. For convenience, let's assume that there are two nodes with a cluster. In

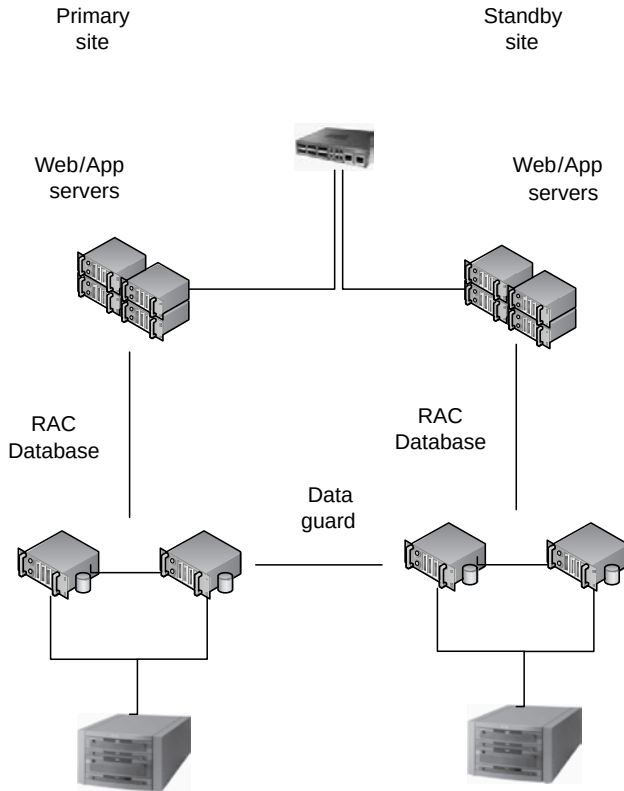


Figure 12.2 Oracle's Maximum Availability Architecture (MAA) consisting of a RAC and a Data Guard.

active/active clustering mode, both nodes are actively processing requests, while in active/passive clustering mode, one node is active and the other is passive or in a standby state. In active/passive mode, the standby node doesn't participate in processing requests. However, if the active node goes down, the passive node takes over immediately. A RAC is typically configured to run in active/active mode, while a Data Guard setup is configured to run in active/passive mode.

Figure 12.2 shows how the concept of a data guard fits into a maximum availability setup with a production RAC environment mirrored to a standby environment. It's also called maximum availability architecture (MAA), because of the enhanced availability from a RAC.

In a smaller scale without a RAC, a data guard configuration can be as simple as shown in Figure 12.3, which consists of one production database and one or more standby databases. The primary and standby databases in a Data Guard configuration don't have to be physically located side by side—they are connected by Oracle Net so standby databases can be located anywhere remotely as long as they are reachable via networking.

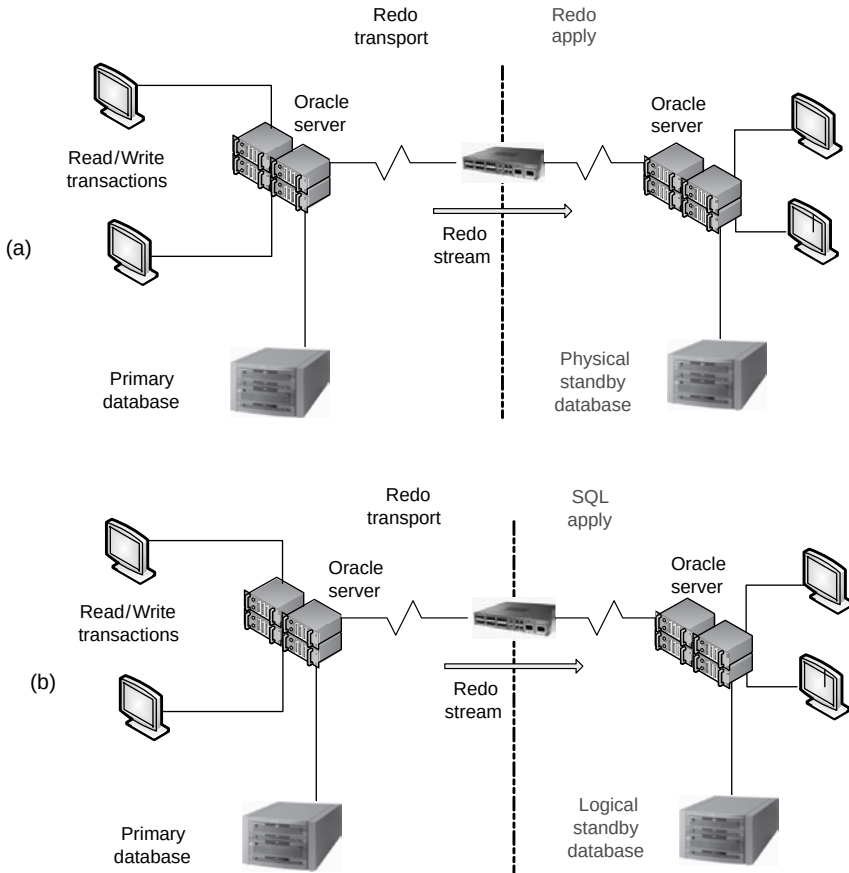


Figure 12.3 Oracle Data Guard Configuration with (a) a physical standby database, and (b) a logical standby database. Note the Redo Transport service used in both configurations and Redo Apply service in the physical standby database configuration and SQL Apply service in the logical standby database configuration.

In a typical data guard configuration, one can have a physical standby database or a logical standby database. The differences between the two are discussed below:

- **Physical Standby Database.** In this case, the standby database is an exact copy of the primary database on a block-by-block basis. Primary and physical standby databases are synchronized through a service called *Redo Apply*, which recovers the redo data from the primary database and applies the redo to the standby database. See Figure 12.3 (a) for a Data Guard configuration with a physical standby database.
- **Logical Standby Database.** In this case, the standby database contains the same logical objects as the primary database, but the physical storage of the objects doesn't have to be the same. Synchronization between the primary and standby

databases is achieved through a service named SQL Apply, which transforms the redo data from the primary database into SQL statements and then executes the SQL statements on the standby database. See Figure 12.3 (b) for a Data Guard configuration with a logical standby database.

In summary, a Data Guard mirrors a primary production Oracle database to a physical or logical standby Oracle database. The difference between physical and logical standby databases is whether the same objects from the primary database is moved to identical physical storage on the standby database side. Both configurations operate on primary redo logs, which contain transactional redo data that represents the state of a database at a point of time. Redo data transfer is performed by the Redo Transport service with the help of redo streams. On the standby database side, one of the two services is initiated to write incoming redo data: Redo Apply service for a physical standby database or Redo SQL service for a logical standby database.

12.2.3 Performance Tuning Intelligent Advisors

Oracle started offering performance tuning intelligent advisories with 9i. Noteworthy advisories from this version were:

- *Shared Pool Usage Advisory.* This advisory probes shared pool usage and attempts to improve parse time and to minimize CPU usage. It also analyzes SQL execution memory to improve SQL execution performance and to minimize unnecessary CPU and IO utilizations.
- *PGA Aggregate Target Advisory.* This advisory optimizes dynamically the amount of PGA memory allotted to SQL work areas within the limit set with the parameter `PGA_AGGREGATE_TARGET`.

Note that these advisories have been further improved in 10g and 11g. In addition, other features added to the OEM in 9i were: XML DB (a set of built-in high-performance XML file storage and retrieval technologies), Oracle Streams (a distributed messaging technology for replicating databases), the Data Guard SQL Apply Database, and so on.

12.2.4 Actual Operation-Level Query Statistics

This was another intelligent Oracle performance tuning feature that could help identify most heavily accessed tables, indexes, and partitions. Once again, such intelligent features have been significantly enhanced in 10g/11g, and we will cover more about them later. The purpose of introducing them here is to give you an idea when they got started so that you can estimate their maturity with the version of Oracle you are using.

12.2.5 Dynamic Sampling of Optimizer Statistics

In Chapter 16, the Oracle Optimizer will be introduced in detail. In Chapter 27, a quantitative case study will be presented to demonstrate how critical it is to keep

optimizer statistics accurate and updated. Optimizer statistics can be updated manually. However, it can be set to be updated dynamically so that Oracle can take care of it automatically by itself.

Dynamic sampling is enabled by default. What statistics are gathered with dynamic sampling is controlled by the level of dynamic sampling specified with the parameter `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter. The criteria that trigger dynamic sampling are described as follows:

- *Level 0.* Dynamic sampling is disabled.
- *Level 1.* Dynamically sample each of all unanalyzed tables if all these conditions are satisfied: (1) there exists one or more unanalyzed, non-partitioned tables in the query, (2) this unanalyzed table has no indexes, (3) this unanalyzed table has more than 32 blocks (sample size) to be dynamically sampled.
- *Level 2 (default level).* Use dynamic sampling if at least one table in the SQL statement has no statistics. The sample size is 64 blocks.
- *Level 3.* Use dynamic sampling if either of these conditions is true: (1) the SQL meets level 2 criteria or (2) the statement has one or more expressions in its WHERE clause predicate. The sample size used for this level is 64 blocks.
- *Level 4.* Use dynamic sampling if either of these conditions is true: (1) the SQL meets level 3 criteria or (2) the statement uses complex predicates (an OR or AND operator between multiple predicates on the same table). The sample size used for this level is 64 blocks.
- *Level 5.* Same criteria as level 4 except using a sample size of 128 blocks.
- *Level 6.* Same criteria as level 4 except using a sample size of 256 blocks.
- *Level 7.* Same criteria as level 4 except using a sample size of 512 blocks.
- *Level 8.* Same criteria as level 4 except using a sample size of 1024 blocks.
- *Level 9.* Same criteria as level 4 except using a sample size of 4096 blocks.
- *Level 10.* Same criteria as level 4 except using all blocks.

That's all dynamic sampling is about. You can revisit this feature briefly introduced here after you study Chapters 16 and 27.

12.2.6 Cloning Production Database with Oracle Enterprise Manager

One dilemma with developing an enterprise application is lacking realistic production data for testing the performance and scalability of the product in a controlled internal environment. Starting with 9i, Oracle offered a feature for cloning a subset of a production database (both data and statistics) so that the cloned dataset could be imported into another testing environment for analyzing and testing further. This feature is not only helpful for development testing but also for resolving customer escalations. However, one must make sure that comparable hardware and configurations are replicated in the testing environment or such factors have been taken into account when evaluating the test results.

12.2.7 Renaming Columns and Constraints

Starting with 9i, one could rename columns and constraints, which might be necessary during the development cycle of a product. However, renaming may cause problems in the application code and corresponding changes should be made as well.

12.2.8 Dynamic Memory Pools

Starting with 9i, a few memory pools in an SGA (buffer pool, shared pool, and large pool) could be manually changed dynamically, which means that there is no need to restart the database. Note that these pools can be automatically managed by Oracle in 10g and 11g, so manual adjustments may not be necessary unless you are sure that you can do a better job than Oracle, based on the peculiarities of your application that you know more than Oracle does.

12.2.9 Flashback Query

With this feature, you can run a query against your database traced back to a past point of time so that you could compare the performance of the query in certain before/after scenario. It was based on the same mechanism of multi-version read consistency that was introduced earlier.

12.2.10 List Partitioning

In Section 12.1.12, we introduced range partitioning, hash partitioning, and composite partitioning. In 9i, one more partitioning method of list partitioning was introduced. The concept of list partitioning is simple. For example, you partitioned your sales data based on the range of years. Then on each partition, you could further partition based on regions, for example, North America, Asia, Europe, and so on, and that is list partitioning. List partitioning can be used jointly with range or hash partitioning to form a composite partitioning.

12.3 ORACLE 10g NEW FEATURES

At the heels of the previous two sections on Oracle 8i and 9i new features, we continue our exploration of new features in 10g. Note that the letter “i” is now replaced by letter “g” which implies “grid computing.” Although it’s as simple as a single letter change, it signified a new vision on enterprise computing at Oracle. We will cover what grid computing is about later in this chapter.

Some Oracle 10g new features that will be briefly introduced in this chapter include the following:

- Automatic Storage Management (ASM)
- Asynchronous Commit

- Database Replay
- Read Performance Statistics Directly from the SGA
- Automatic Workload Repository (AWR)
- Automatic Database Diagnostic Monitor (ADDM)
- Automatic Shared Memory Tuning
- Automatic Optimizer Statistics Gathering
- SQL Tuning Features
- Grid Computing

Let's begin with the feature of Automatic Storage Management (ASM) next.

12.3.1 Automatic Storage Management (ASM)

Oracle manages data and data must be stored on physical storage devices in the smallest units of blocks. As we emphasized in Chapter 8, mostly IO or storage would eventually turn out to be the dominating bottleneck for an Oracle database. Storage is different from physical memory. Memory is not quite configurable externally—you have either a lot more or less installed on your database server. However, storage is externally configurable and how you configure your storage can critically determine the performance and scalability of your Oracle-based enterprise application.

In order to help you understand the concept of ASM better, I'd like to step back a little and show you one screenshot taken at the time when a 10g R2 database was being created. As shown in Figure 12.4, there are three options for choosing a storage mechanism for your database being created:

- *File System.* This is the simplest choice. You simply choose a pre-configured file system, which could be as simple as an OS-level file system such as NTFS for Windows or as more advanced as a third-party, database storage-oriented file system like Veritas. In this case, you just use the file system and all management tasks are taken care of by the underlying OS or the file management system. Since our subject here is about ASM, let's move on to ASM next.
- *ASM.* This is the second option shown in Figure 12.4. It says "*ASM simplifies database storage administration and optimizes database layout for I/O performance. To use this option you must either specify a set of disks to create an ASM disk group or specify an existing ASM disk group.*" We'll expand on this introduction after describing Raw Devices option next.
- *Raw Devices.* As you see, it says if you do not use a clustered file system (like Veritas) and if you don't use ASM, then use this option for the required shared storage for your RAC. If you choose this option, then raw partitions or volumes must be created *a priori*. That typically is done by very experienced professional administrators, but it suffices to say that raw devices provide the best possible I/O performance at the cost of complexity. Some research has demonstrated that one could get near-raw-device I/O performance if all the performance tunings at both

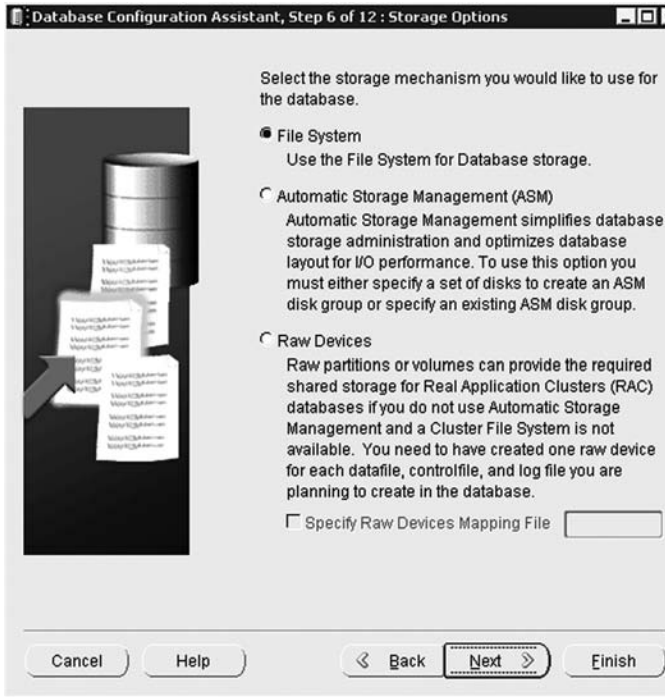


Figure 12.4 Available storage options in Oracle 10g R2.

the OS level and file system level were fully understood and exploited. But that's beyond the scope of this text. So let's get back to ASM next.

Now let's explain how ASM actually works. Here is how:

- The start point with ASM is ready-to-use *disks*, which are always partitions on Windows or partitions of a logic unit number (LUN) or network-attached files on all other platforms.
- Then those disks as described above are configured into *disk groups*. From this point on, each disk group is managed as a storage unit. Note that each disk group has a *type* associated with it. In order to understand disk group types, we have to digress a little to review the concepts of disk *striping* and *mirroring*. Let's review these concepts in ASM's context next.
- *ASM disk striping*. There are a variety of methods to stripe data across multiple individual physical disks, but here is how ASM does it uniquely: ASM divides files into 1 MB (fixed) extents and spreads each file's extents evenly across all disks in a disk group. This is good for I/O performance because data is striped across all disks, eliminating potential I/O contentions if a single disk were hit all the time. But striping does not solve data protection problem. That's what mirroring would do, as we discuss next.

- *ASM disk mirroring.* There are three options for ASM mirroring configurations:
 - *Unprotected.* ASM provides no mirroring. But this really is not an accurate description, because if you are using a RAID already, then disks could be already mirrored at the RAID level. If the underlying RAID were not configured with mirroring or if you are using independent physical disks underneath, then it's truly *unprotected*.
 - *2-way mirroring.* In this case, each extent has 1 mirrored copy.
 - *3-way mirroring.* In this case, each extent has 2 mirrored copies.
- Now after disk groups have been *typed* with proper mirroring configurations, they are ready to be used for your database creation and after that Oracle would take care of everything else, which is the drive behind ASM that Oracle wanted to free you from doing daunting storage management yourself.

At this point, it should be clear to you what ASM is about. ASM is not another file system like an OS-specific file system or OS-agnostic commercial file system like Veritas. Instead, it is more of a file system manager that takes care of some general I/O configuration tasks such as striping and mirroring on your behalf.

Next, the Oracle 10g feature of Asynchronous Commit is introduced.

12.3.2 Asynchronous Commit

Oracle 10g introduced a performance driven technique, which is called asynchronous commit. This technique allows other operations to be performed concurrently while redo data is being written to disk. This feature reduces commit time and thus improves transaction throughput.

12.3.3 Database Replay

In an IT environment, changes are more common than anything else. Typical changes may include hardware and/or software upgrades, and so on. Very often, when an upgrade is performed, performance and scalability of the application are degraded noticeably. So typically, companies exercise caution by testing thoroughly before initiating an upgrade. One of the frequently used testing methods is to use synthetic workloads that may or may not simulate the complexities of the production workloads such as complicated workload profiles and the degree and patterns of user concurrency. Another method is to use some commercial capacity planning software products to predict the performance and scalability with various if-scenarios. Such products are even much less reliable than testing with synthetic workloads.

To mitigate risks in performance and scalability caused by planned upgrades, Oracle 10g's Database Replay feature allows production workloads to be captured and then replayed in a testing environment. It can also report potential functional errors in addition to performance and scalability issues. Currently, this feature supports capturing on 10g and replaying on 11g only.

12.3.4 Read Performance Statistics Directly from the SGA

With 10g's new HTTP-based Enterprise Manager, performance statistics of a database are populated directly from the SGA and then displayed on the console. This is a very useful feature when a database is hung or slow and it's impractical to query such performance statistics by issuing SQL statements from a command line. In other words, when the database is hung or slow, the EM may not suffer the same resource contentions as the database does, and therefore, the channel for probing the system performance issues may remain open. However, this may not work all the time, depending on many factors understandably.

12.3.5 Automatic Workload Repository (AWR)

We have covered AWR heavily in Chapter 11, so we only reiterate here that AWR perhaps is the most significant improvement in the suite of Oracle's performance troubleshooting tools. Many quantitative case studies to be presented later testify how useful AWR is for diagnosing and resolving Oracle performance and scalability issues.

12.3.6 Automatic Database Diagnostic Monitor (ADDM)

ADDM is a feature designed to automatically detect performance bottlenecks and recommend solutions to resolve them. Using the statistical data stored in the AWR, ADDM performs analysis and creates reports on a regular basis (hourly by default). This feature was expanded in 10g to include more components such as RAC, AQs, Streams, and so on. Here I just want to raise your awareness of this feature, and it will be covered in depth in a later chapter about more Oracle auto-tune features.

12.3.7 Automatic Shared Memory Tuning

We have covered Automatic Shared Memory Management (ASMM) in detail in Chapter 6. It is listed here again simply because this is one of the most important performance and scalability features introduced in 10g. If you want to review this feature again, refer to Chapter 6 of this text or relevant Oracle documentations available from Oracle's Web site.

12.3.8 Automatic Optimizer Statistics Gathering

We have emphasized throughout the text about how important it is to keep optimizer statistics both up-to-date and accurate as possible from performance and scalability perspectives. Although manually gathering optimizer statistics is an option, it's impractical to completely rely on such a manual process to keep optimizer statistics up-to-date and accurate. To resolve this issue, Oracle has automated optimizer statistics gathering in 10g.

Automatic optimizer statistics gathering is scheduled by default to run during a nightly maintenance window from 10:00 PM to 6:00 AM weekdays and all day on

weekends. The name of the job is GATHER_STATS_JOB. As soon as it is started, it continues until it finishes regardless of the limits set with the maintenance window. However, both the window period and gathering behavior can be reconfigured.

Note that this job gathers statistics only on the objects (tables, indexes, and columns) in the database that have either missing statistics or stale statistics. Coupled with the scheduled maintenance window, there are potential issues. For example, day-time and nightly workloads could be very different, and the optimizer statistics gathered during the nightly maintenance window may not apply to day-time workloads. This is especially true if you have highly volatile tables in your database. Oracle recommends two approaches to remedying such a situation:

- Manually gather the optimizer statistics on the identified volatile objects during their representative periods (e.g., during day-time or a peak load period) and then lock the gathered statistics so that they will not be changed during the maintenance window by the scheduled automatic statistics gathering job.
- Set optimizer statistics on those volatile objects to NULL. Then when Oracle encounters an object with no statistics, it dynamically gathers the optimizer statistics as part of query optimization. However, you need to make sure the OPTIMIZER_DYNAMIC_SAMPLING parameter is set to a value of 2 (default) or higher, as discussed previously. The optimization statistics can be set to NULL and locked as follows:

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS ('<schema>', '<table>');
  DBMS_STATS.LOCK_TABLE_STATS ('<schema>', '<table>');
END;
/
```

There are two other special scenarios that need special arrangements: bulk-load jobs and system statistics. For bulk-load jobs, optimizer statistics should be gathered either manually or as part of the same script or program that initiated the jobs within a few minutes after they are started. This was what I had done all of the time several years ago with an enterprise application I worked on as a performance engineer.

Regarding system statistics such as CPU- and IO-related performance counters and metrics, they are not gathered automatically and need to be gathered manually. The optimizer requires such information to estimate the CPU and IO costs when determining the optimal execution plan for a query. The system statistics are gathered with the DBMS_STATS.GATHER_SYSTEM_STATS procedure, which analyzes system activity in a specified time period. It's necessary to make sure that system statistics are available when the optimizer needs them. However, according to Oracle, unlike statistics of the database objects, already parsed SQL statements are not invalidated when system statistics get updated. Only new SQL statements are parsed with updated system statistics taken into account.

12.3.9 SQL Tuning Features

SQL tuning has always been an important part of database performance and scalability optimization for all database products. Oracle 10g introduced new features in this area to facilitate SQL tuning. This feature set is summarized as follows:

- *Portable SQL Tuning Sets.* A SQL tuning set is defined as a set of SQL statements together with their associated execution context (user schema, application module name and action, list of bind variables, etc.) and the associated execution statistic metrics (elapsed time, CPU time, buffer gets, disk reads, rows processed, etc.). Oracle 10g enables SQL Tuning Sets to be exported from one system and imported into another system. This is especially convenient for tuning SQLs in a production environment where developers may not have or are not allowed to have direct access to the environment. In this case, one can just export the offending SQLs via SQL Tuning Sets and import them into a separate environment for investigation. However, make sure that the two environments are comparable in terms of hardware and configurations, or the disparities between the two must be taken into account when evaluating the performance of the problematic SQLs.
- *Display SQL History in Enterprise Manager.* This is a useful feature when one is interested in studying the performance of a SQL statement before and after a tuning technique is applied.
- *Interruptible SQL Access Advisor.* A SQL Access Advisor analyzes a SQL statement and recommends tunings with predicted improvements. I have used this feature with one of the real products I worked on and it's quite effective. However, it's important to make it interruptible as sometimes for whatever reasons it might get stuck and you just want to stop it.
- *SQL Access Advisor Recommends Function-Based Indexes.* This is just one more type of indexes that was added to the SQL Access Advisor.

12.3.10 Grid Computing

Grid computing was Oracle's vision for the then next computing paradigm shift. It was anticipated that large computing grids would become a norm or mainstream that computing needs can be satisfied just like utility services such as electricity, gas, and so on. To some extent, I would say it resembles the now reality, new computing paradigm of so-called cloud computing, but many purists may think otherwise. Anyway, my purpose here is not to argue about grid computing versus cloud computing. Instead, I'd like to help you understand what Oracle 10g has to offer to facilitate grid computing, as the letter "g" has a special place here after all.

Oracle 10g's new features in the grid computing area include:

- *Data Provisioning.* This feature refers to the data operations such as archiving, moving, and copying large datasets, and so on. New features in 10g had made

such operations easier and faster. Also the effects of performing such bulk data provisioning operations on production databases were minimized.

- *Grid Management.* This feature set includes the following features:
 - *Configuration Plug-ins and Assistants.* This is a subset of grid management features that enable the Enterprise Manager Database Control (EMDC) to be configured out of the box either using a seed database or a customer database. It also enabled post-install configuration of an EMDC with either Database Configuration Assistant or custom scripts. Migrating Non-ASM databases to ASM-based bases through an EMDC was made possible as well.
 - *Database Management.* This is a set of database management features that enables: (1) automatically discovering or manually adding host cluster targets to the Enterprise Manager, and (2) monitoring the performance of up to 64–128 nodes or instances in a RAC environment, and so on.
 - *Patching.* This is a patching tool that helps administrators apply large scale patching to an Oracle grid environment.
 - *Software Install and Cloning.* This is a subset of the grid management features that enables: (1) adding a node to a RAC or cloning a RAC, (2) silently installing a RAC, and (3) streamlined ASM and RAC interface installation processes, and so on.

This concludes our introduction to new features of Oracle 10g. In the next section, we introduce the new features of Oracle 11g.

12.4 ORACLE 11g NEW FEATURES

Oracle 11g is the newest version as of this writing. Similar to how we introduced some of the new features of the previous versions of Oracle in the previous sections, in this section, I'll introduce some of the new features of 11g mainly from performance and scalability perspectives. These features include:

- Automatic Memory Management (AMM)
- Intelligent Cursor Sharing
- Server Result Cache
- Database Smart Flash Cache
- Database Replay SQL Performance Analyzer (SPA) Integration
- I/O Calibration
- Partitioning Enhancements
- SQL Plan Management
- Zero-Size Unusable Indexes and Index Partitions
- Invisible Indexes
- Virtual Columns

Let's begin with AMM next.

12.4.1 Automatic Memory Management

In Chapter 7, we introduced in detail about how Oracle 11g has automated the management of the entire memory (SGA + PGA) allocated to Oracle. It's listed here as the number one new feature in 11g based on my opinion that this is the most important feature to keep in mind from all perspectives, from database creation, post-install configuration, administration, development, and performance and scalability tuning, and so on. This feature has a cascading effect on many aspects of Oracle, especially on those auto-tune features. Anyway, if you are doing Oracle performance optimization and tuning work whether in a production or development environment, it's necessary that you spend some time fully understanding how it works, preferably enhanced with some hands-on exercises as well.

Since this feature has been fully covered in a previous chapter, we move on to the Intelligent Cursor Sharing feature next.

12.4.2 Intelligent Cursor Sharing

First of all, properly setting `CURSOR_SHARING` is extremely important for those Oracle-based enterprise applications that do not use bind variables. A quantitative case study will be presented in a later chapter to show the significant impact of this parameter on the performance and scalability of one of the enterprise applications I worked on.

This feature is called adaptive cursor sharing as well. It's called "intelligent" or "adaptive" because the cursor adapts to different execution plans with different values of the bind variables. However, keep in mind that this feature is a new feature that: (1) it applies to SQL statements that use bind variables only, (2) it is independent of the `CURSOR_SHARING` initialization parameter we mentioned throughout this text, (3) it is turned on by default and it cannot be disabled, and (4) it does not apply to those SQL statements that use more than 14 bind variables. So far, I haven't seen quantitative case studies showing how effective this adaptive cursor sharing feature could be, and it would be hard to quantify because by default it's turned on and it cannot be disabled.

12.4.3 Database Resident Connection Pool (DRCP)

DRCP (Database Resident Connection Pool) is a new feature introduced in Oracle 11g. It's more thoroughly described in Oracle 11g R2's *Concept* documentation. Its utility is well described as it "provides a connection pool of dedicated servers for typical Web application scenarios." This statement can be parsed in two parts: (1) *a connection pool of dedicated servers* and (2) *for typical Web application scenarios*. Based on this division, we can infer about a DRCP that:

- It is designed to work with a special, limited scenario of a Web application connected to an Oracle server running in dedicated server mode. In such a scenario, the workload is characterized by a series of user request bursts that keep opening, using, and closing connections to Oracle, incurring excessive resource

utilizations for doing so repeatedly. With a DRCP, such overheads can be saved by pooling connections to the dedicated server, thus improving performance and scalability.

- A DRCP is more effective if the Web application does not implement its own connection pooling itself, such as a PHP Web server running in single-threaded mode communicates to an Oracle database server running in dedicated mode. Or in other words, a DRCP is not helpful if an application is non-Web type, for example, with the scenarios of enterprise batch jobs where the application runs in single-threaded mode but each single thread uses an open connection for hours before finishing the job and closing the connection.

In summary, a DRCP is useful only for *OLTP* type of applications which *do not* implement their own connection pools. If your application happens to fall into this category, it's worthwhile to give it a try in a test environment first as it is claimed that it can potentially help your application scales up to tens of thousands of connections (or users) rather than hundreds of or thousands of connections or users with your Oracle server running in dedicated mode. Consult Oracle's *Concept*, *Administration*, and *Performance Tuning* documentations for the concrete mechanics about how to enable, configure, and tune your DRCP for the best possible performance and scalability for your application.

12.4.4 Server Result Cache

It's well known that Oracle caches cursors, parses SQLs, executes plans, and so on. In 11g, Oracle goes one step further and caches query results with a newly added memory pool within the shared pool. Here is how it works:

- When Oracle receives a SQL query, it first checks the result cache for the result cached. If the result exists in the cache, it would use the cached result right away without having to rerun the same query again, thus improving performance.
- If the result is not found in the result cache, then the query is executed, and the result is both cached and returned to the client.

The server result cache feature is enabled on a SQL by SQL basis by adding a hint of `/* + RESULT_CACHE */` immediately after the `SELECT` keyword. In addition, there are a series of initialization parameters, such as `RESULT_CACHE_MAX_SIZE`, `RESULT_CACHE_MAX_RESULT`, and so on, that can be specified to control how the result cache memory pool would work. Besides, there is a corresponding Client Result Cache feature that works on the client side. Consult the relevant Oracle documentation if you feel these features might help you. However, be warned that excessive caching may hurt performance adversely, as will be demonstrated in a quantitative case study presented in Chapter 23 of this text on double buffering.

Apparently, this feature would be more effective for DSS type of applications than for OLTP type of applications, because SQLs are much more static in the former case than in the latter case.



Figure 12.5 Solid State Devices (SSDs) from Toshiba (Courtesy of Toshiba).

12.4.5 Database Smart Flash Cache

This is a new feature introduced in 11g R2. The Database Smart Flash Cache is a transparent extension of the traditional database buffer cache using solid state device (SSD) technology. In Oracle's term, SGA is like an L1 cache, while SSD is like an L2 cache to an Oracle Server. But before introducing SSD in Oracle's context, let me help you understand what SSD is about in case it's somewhat new to you.

First, I'd like to show you what an SSD looks like. Figure 12.5 shows two SSDs from Toshiba. It seems that they look both like traditional RAM chips and traditional hard drives, but actually they are somewhere in between. They are not like traditional RAM chips because they are non-volatile, namely, the data stored on them will persist even after power is turned off; and they are not like traditional hard drives because they operate completely electronically without electro-mechanical spindles inside. The RAM-chip like SSD is Toshiba's 32 nm Blade X-gate SSD module, while the disk-like SSD is Toshiba's 32 nm high-performance SSD. There are many unique advantages with an SSD, including (1) high I/O performance (e.g., up to 220 MBps in sequential read mode and 180 MBps in sequential write mode), (2) small form factor as is obvious, and (3) reliability (some of it was claimed to be able to run for over 100 years!).

Oracle made this feature available at this time on Linux and Solaris only. This feature has a strong affinity to its Exadata product, which is a "purpose-built" large-scale data warehouse appliance competing with TeraData, which is another company specialized in providing large-scale data warehouse solutions to help support intelligent decision making.

We won't go into too much detail about how Smart Flash Cache works with Exadata, but it's interesting to get a glimpse of how an Oracle database on Exadata would know SSD storage is available for its use. There are two methods:

- *Pinning Objects in the Flash Cache.* This method uses a new storage attribute, `CELL_FLASH_CACHE`, which can be assigned to a database object such as a table, an index, a partition, a LOB column, and so on. There are three settings to this attribute: `DEFAULT`, `NONE`, and `KEEP`. `DEFAULT` means that caching would be managed automatically, `NONE` means never cached, and `KEEP` means

cached. For example, a CUSTOMER table can be pinned to the ExaData Smart Flash Cache with the execution of the following SQL command:

```
ALTER TABLE customer STORAGE (CELL_FLASH_CACHE KEEP)
```

Note that the above storage attribute can also be specified when the CUSTOM table is created.

- *Creating Flash Drives Out of the Flash Cache.* With this method, a flash cache would be used just like regular disks except that it provides much higher I/O performance.

Since this is a very new feature at this time, keep up with Oracle to know its most up-to-date progress.

12.4.6 Database Replay SQL Performance Analyzer (SPA) Integration

In Oracle 11g, Database Replay is integrated with SQL Performance Analyzer (SPA), which combined two separate processes into one. You may want to explore this feature further if optimizing Oracle database performance and scalability is part of your daily job. More details about this feature can be found in Oracle's *Real Application Testing User's Guide*.

12.4.7 I/O Calibration

This feature helps isolate whether an I/O-related performance issue is caused by Oracle configuration or the underlying I/O subsystem. Note that the I/O calibration activity adds a significant overhead so that it should be used only when the database is idle. Secondly, finding out whether I/O is the bottleneck is much painless from an AWR report as we discussed in Chapter 11. If at some point you feel that you have to use this feature to pinpoint down your I/O-related performance issue, consult Oracle's documentation about the concrete mechanics of making it work.

12.4.8 Partitioning Enhancements

Oracle 11g introduced a new partitioning method of *interval partitioning*. Interval partitioning is an extension to the regular range partitioning method that a range is not specified with a less-than (<) expression but is calculated with a given function. It also extended the composite partitioning methods from the previous versions, allowing all permuted composite partitioning methods as listed below:

- Range—Hash (since 8i)
- Range—List (since 9i)
- Range—Range
- List—Range

- List—Hash
- List—List
- Interval—Range
- Interval—Hash
- Interval—List

In addition, a *Partition Advisor* is provided as a new member of the SQL Access Advisory. It does not only help you carry out a partitioning on a database object but also predicts how much performance improvement you would get if you partition, which could be taken as a valuable reference. Once again, if you decide that you want to implement partitioning, refer to Oracle's relevant documentation or texts that cover this topic in more details.

12.4.9 SQL Plan Management

SQL Plan Management is a feature that prevents the execution plan of a SQL from changing abruptly. It helps achieve this objective by making sure only known-to-be-good plan is used and it does not adopt a new plan unless it's verified to have better performance than the current plan. The core concept behind SQL Plan Management is a SQL plan baseline set, which stores the known, acceptable plans. Rather than focusing on how to create and manage plan baselines (such detailed technical bits are always available from Oracle's documentations), let's next explain what change scenarios can leverage SQL Plan Management to prevent Oracle performance regressions from happening.

SQL Plan Management can help prevent performance regressions with the following change scenarios:

- From development environment to production environment. Enterprise applications are typically tested heavily in internal development environments with realistic workloads. Often times performance regressions arise in customer environments after a product is released. By shipping established SQL plan baselines with a product, some performance regressions can be avoided, although it may not prevent all performance regressions from happening.
- It's very common that performance regressions occur when a software product including Oracle database is upgraded. SQL Plan Management can help prevent performance regressions caused by software upgrades.
- Even in a production environment, changes in systems and workloads may cause execution plan changes to some SQL statement, and thus causing performance regressions. With SQL Plan Management in place, an enterprise application is provided a path to adapt to the newly changed environment without incurring performance regressions.

Next, we introduce the feature of Zero-Size Unusable Indexes and Index Partitions introduced in Oracle 11g.

12.4.10 Zero-Size Unusable Indexes and Index Partitions

From an administrative perspective, one often is concerned with the space occupied by unusable indexes and unusable index partitions. Developing a script to find out and clean such unusable objects may become a time-consuming job. Now with 11g, Oracle removes this burden from DBAs and frees such unusable objects by itself.

12.4.11 Invisible Indexes

Oracle 11g allows invisible indexes to be created by adding the keyword `INVISIBLE` to a regular create-index SQL statement. Independently, you can make a visible index invisible with the command

```
ALTER INDEX <your_index> INVISIBLE;
```

An invisible index can be made visible with the command

```
ALTER INDEX <your_index> VISIBLE;
```

What's the use of an invisible index? An index is ignored by the optimizer when it's in an *invisible* state and is considered by the optimizer when it's in a visible state. This is a helpful feature to help a user evaluate the performance impact of an index, especially in an internal test environment. Prior to 11g, I had to manually create and delete an index in order to test out whether the index under test could help improve the performance of my application or not. To some extent, this shows Oracle's commitment to making its database platform more and more user friendly.

12.4.12 Virtual Columns

A virtual column is similar to a conventional physical column except that it does not occupy space on a disk. It's a column with its values computed dynamically on demand based on other physical columns it's defined on in an expression or function. Or from another perspective, you may think that some business logic is moved into database with virtual columns, because computations associated with virtual columns could be performed in an application after a result set is returned from a query resulting from a table with all physical columns. Note that the performance of a query like `SELECT * FROM <table>...` may suffer an unnecessary overhead with a table that contains virtual columns.

Also note that virtual columns can be used in queries, DML, and DDL statements. Other operations such as indexing, gathering statistics, and applying integrity constraints, and so on, also apply to virtual columns. However, exercise caution that you may have made it more complicated than necessary if too many virtual columns were defined on a table.

12.5 SUMMARY

In this chapter, we introduced all major new features from Oracle 8i, 9i, 10g and 11g from performance and scalability perspectives. Some features were intended for increasing performance and scalability, and some were for facilitating diagnosing performance and scalability issues. However, we are not done yet. In the next chapter, I'll offer an aggregated view of the top 10 performance and scalability features supported in the latest version of Oracle: 11g R2 as of this writing.

RECOMMENDED READING

For Oracle 8i features and options, refer to the following Oracle document:

Oracle Corp., Getting to Know Oracle 8i Release 2 (8.1.6), December 1999, Part No. A76962-01, available online at: http://download.oracle.com/docs/cd/A87860_01/doc/server.817/a76962/preface.htm

For Oracle 9i features and options, refer to the following Oracle document:

Oracle Corp., Oracle 9i Database New Features Release 2 (9.2), October 2002, Part No. A96531-02, available online at: http://download.oracle.com/docs/cd/B10501_01/server.920/a96531/ch2_9ir2.htm

For Oracle 10g features and options, refer to the following Oracle document:

Oracle Corp., Oracle® Database New Features Guide 10g Release 2 (10.2), January 2008, Part No. B14214-04, available online at: <http://www.oracle.com/pls/db102/homepage>

For Oracle 11g features and options, refer to the following document:

Oracle Corp., Oracle® Database New Features Guide 11g Release 1 (11.1), October 2008, Part No. B28279-03, available online at: <http://www.oracle.com/pls/db111/homepage> and <http://www.oracle.com/pls/db112/homepage> (11g R2—the latest as of this writing)

EXERCISES

- 12.1** In your opinion, what could you benefit from having a comprehensive overview of all major Oracle performance and scalability features since 8i?
- 12.2** List a few examples to illustrate the utility of Oracle unstructured data types.
- 12.3** What's the relationship between Oracle Parallel Server and RAC? Are they two separate products supported in the latest version of Oracle?
- 12.4** Explain conceptually all available partitioning methods supported in Oracle 11g. What are the differences among the range, hash, list, and interval partitioning methods? Is partitioning more performance or maintenance driven?

- 12.5** Is it accurate to say ASM is Oracle's file system management product like NTFS for Windows, UFS for Solaris, or Veritas as a non-OS specific file system?
- 12.6** Explain the relationship between AWR and ADDM.
- 12.7** Does Intelligent Cursor Sharing make the conventional Cursor Sharing settings of EXACT, SIMILAR, and FORCE obsolete? Why or why not?
- 12.8** Explain how the server result cache feature may help or hurt performance.
- 12.9** Explain the differences between solid state devices (SSDs) and traditional hard drives. How does Oracle take advantage of SSDs?
- 12.10** Explain what for I/O Calibration is. Is it safe to run it in production?
- 12.11** What's the purpose of the invisible indexing feature? Does it help improve query performance?
- 12.12** Does the virtual column feature help improve query performance? Does this feature break the normalization level of a database?
- 12.13** Explain quantitatively how *high availability* is defined.
- 12.14** What's the difference between RAC and Data Guard conceptually? Give a scenario where both might be needed.
- 12.15** What's the difference between active/active and active/passive clustering configurations? Could we have a passive/passive configuration? (No kidding here. I was once asked this question seriously when I was working on setting up a SQL Cluster about 10 years ago).

13

Top 10 Oracle Performance and Scalability Features

Play the music, not the instrument.
—Anonymous

You might wonder why we need this chapter, given that we have introduced so many Oracle performance and scalability features in the preceding chapter, all the way from 8i to 11g. Well, I don't feel I have done my job if I do not offer you a list of top 10 performance and scalability features from my point of view, and most importantly, based on my experiences in all Oracle versions from 8i to 11g.

However, I have to mention that this chapter serves more like a checklist for all major performance and scalability features that any Oracle-based application development team should take seriously. Besides, they are cumulative in the sense that not all features listed here are new in Oracle 11g – most of them are enhanced in 11g. To set a proper expectation, I have to say that this chapter is not a *how-to* guide about how to implement these features technically. Typically, before I try to find out how to carry out a specific task, I would first ponder whether I am moving in the right directions, and that's the purpose I attempt to fulfill in this chapter.

My list of the top 10 Oracle performance and scalability features available in 11g include:

- Real Application Clustering (RAC)
- Dedicated versus Shared Server Models
- Proven Transaction and Concurrency Models
- A Highly Efficient SQL Optimization Engine
- Efficient Parallel Processing with Modern Multi-Core CPUs
- Partitioning
- An All-Encompassing, Powerful Performance, and Scalability Troubleshooting Tool—AWR
- The Most Comprehensive Set of Internal Performance Metrics
- Database Resident Connection Pool
- Times-Ten In-Memory Database

Next, let's start with RAC first.

13.1 REAL APPLICATION CLUSTERING (RAC)

An Oracle RAC promises 24×7 high availability, high performance, and high scalability. However, it's a very complex software product. Typically, almost all enterprise software applications are complex and challenging to develop. The complexities lie at both layers: the application layer and the underlying run-time environment layer. However, an Oracle RAC has taken a very radical approach to making it 100% transparent to the application layer while leaving 100% RAC-related complexities to itself, namely, an Oracle RAC externally manifests itself like a single system, and there is no need to make any changes to an application in order for it to run on an RAC.

There is no question about the great benefits of an RAC. The question is that a company must fully understand what it takes to have a RAC up and functioning in its IT environment. The best scenario is you don't think about it if you really don't need it. Making such a decision would call for a careful ROI (return on investment) analysis, which is beyond the scope of this text.

Now let's say it is well justified and you need to pursue it. Next let's review the complexities related to a RAC implementation from various technical perspectives. A RAC stack would require the following hardware and software components:

- *Shared Storage.* Since all RAC nodes access the same database, a common shared storage is needed. This shared storage is typically provided with a SAN (Storage Area Networks), which has a RAID array configured with a specific RAID level. The SAN used with a RAC should be placed on a dedicated subnet and connected to each node via fiber channel switches and HBAs (Host Bus Adapters) installed on each node. This is a necessary investment and it is not recommended to make a compromise here.

- *High-Speed Interconnect Connectivity.* A RAC does not only need faster access to the shared storage, but also faster interconnect connectivity among the RAC nodes. A RAC needs to keep all nodes synchronized by sending heartbeat messages over a fast inter-node communications channel. Secondly, fetching data from the cache of each node via Oracle's Cache Fusion technology critically depends on the speed of the underlying interconnect link. Simply letting all nodes communicate with each other over a corporate intranet network is not suitable for a RAC. The interconnect must be a private network fully dedicated to a RAC. Using an InfiniBand is a more proper choice. An InfiniBand is a switched fabric communications link matured with the advent of HPC (high-performance computing). It is capable of providing high-speed connectivity ranging from a few Gbits/s to hundreds of Gbits/s/. It's necessary to decide where you want to settle down within this range.
- *Commodity Servers.* A RAC is designed to take advantage of the commodity servers for its nodes. Two of the most critical specs with all nodes (preferably all identical) are the CPUs and RAM. If you are starting new, some of the latest multi-core blade servers would be a good choice. Carefully sizing the number of CPUs at the highest possible CPU GHz with a proper amount of RAM is important, but that's beyond the scope of this text.
- *Storage Options.* A storage option refers to how data would be read off and written to disk. The following options are available for a RAC:
 - *Raw Devices.* With this option, the underlying raw devices are directly presented to Oracle without going through a file system between Oracle and the disks. Oracle manages all I/O operations. Since this option totally eliminates the overhead of a file system, theoretically the best possible I/O performance is achieved. However, it's extremely complex.
 - *LVM (Logical Volume Manager).* An LVM is a software product that is used to combine all physical disks into a logic disk volume. It can also dynamically add new disks to a volume while the volume is being used. In addition, it allows a user to take snapshots of data as backups. An LVM is an acceptable option for a RAC.
 - *ASM.* This option is Oracle's version of an LVM, and it's fully backed up by Oracle for RACs to run on all platforms, so this probably is the most viable choice for a RAC.
 - *CFS (Cluster File System).* A CFS is a file system that allows a shared storage to be connected to multiple servers. This is a popular option not only for RACs but also for all other types of clustered software products in general.
 - *OCFS (Oracle CFS).* This is Oracle's CFS, and also the only CFS that supports both Linux and Windows.
- *Oracle Clusterware Software.* An Oracle RAC requires all files such as data files, control files, redo logs, server parameter files, and so on, to reside on the shared storage. This task is coordinated with Oracle Clusterware. The Oracle Clusterware shares the same private Interconnect as all nodes do. There are

specific network requirements, for example, each node must have at least two NICs (network interface cards), and so on, in order for an Interconnect to work with Oracle Clusterware and a RAC. Consult Oracle's documentations on RACs for further details.

Note that you need to get a lot more training and even consulting services if you are charged to get a RAC environment up and running for your testing or production environments. Be prepared to deal with many unforeseeable issues down the road.

Let's move to the next performance and scalability feature of Dedicated versus Shared Server Models in Oracle 11g.

13.2 DEDICATED VERSUS SHARED SERVER MODELS

As introduced earlier, when you deploy your enterprise application on Oracle, a choice must be made between running your Oracle server in dedicated or shared mode. Simply speaking, in dedicated mode, each user has a dedicated connection to the Oracle server; while in shared mode, users share a connection pool to connect to the Oracle server. The choice is clear when you have only hundreds of users or even thousands of users, in which case, the dedicated mode (default) is more appropriate.

But what if you have an extremely large number of users? In this case, if you are using a RAC, the dedicated mode remains the preferred choice, although one can configure a RAC to run in both dedicated and shared mode, according to Oracle. This practice conforms to the principle of a RAC—you scale horizontally by adding more nodes to the cluster, not by *sharing* on a single server. However, some people just go by “paper,” which says you should use shared mode if the number of users gets really large. Some point out that even for a large number of users, dedicated mode remains the choice and that's what has been proven in the field with real customers.

Unfortunately, I haven't seen quantitative cases studies showing one way or the other with a large number of users. All my experiences and observations have been limited to using the dedicated mode. So here is my take: I would choose the dedicated mode if I have fewer than 10,000 users; if I have a lot more users, I would still stick to the dedicated mode and use a RAC to scale horizontally for the reason that the requirement for supporting more users might better justify the need for a RAC.

In the meanwhile, I hope to see more compelling and convincing case studies from the field about this controversial subject of dedicated versus shared mode for Oracle, more in formally documented case studies than just hearsays.

13.3 PROVEN TRANSACTION AND CONCURRENCY MODELS

Oracle has strengthened its transaction and concurrency models over several decades with more and more innovative techniques. In general, when you apply some optimization and tuning techniques to a database, you would expect that the integrity of the underlying transaction and concurrency models remain intact. This principle has never been broken with my Oracle experience since 8i, but it went the other way

with a same enterprise application on a different database system while I was trying to resolve a significant performance gap between that other database system and Oracle (to be clear, Oracle was significantly better).

Oracle's robust transaction and concurrency models have been implemented with such technologies as locking, latching, multi-versioning, to name a few. To learn more about Oracle's transaction and concurrency models, I'd like to delegate you to Chapter 10 or Oracle's own documentations or Tom Kyte's text (Kyte, 2010) as recommended at the end of this chapter.

13.4 A HIGHLY EFFICIENT SQL OPTIMIZATION ENGINE

For mission-critical enterprise applications, performance and scalability are the key requirements among others. To achieve high performance and scalability at the system level, then at the lowest level, SQL statements must be executed as efficiently as possible (a database server is a SQL chewing machine after all). Oracle never stopped refining its SQL optimization engine. And based on my limited experience, this perhaps is one of the few best SQL optimization engines. It's not only highly efficient by itself, but also leaves the door open for inputting external tuning hints from experienced Oracle performance professionals. It probably is true that Oracle has more performance and scalability tuning knobs than all others combined.

We'll cover more about Oracle SQL Optimizer in a later chapter, and many quantitative case studies will be presented to demonstrate how Oracle meets real world performance and scalability challenges based on my experiences using real products.

13.5 EFFICIENT PARALLEL PROCESSING WITH MODERN MULTI-CORE CPUS

Most server software products have limitations on how many CPUs they can run. There are two reasons for this: one is related to the licensing model designed by the vendor, and the other is related to the technical capability of the product. Specific to Oracle 11g, as shown in Figure 13.1, Enterprise Edition has no limit technically on how many CPUs it can run, while other lower editions are limited to no more than four sockets (note that conventionally a socket is meant to be an entire package for a processor, which could contain a single or more cores. The concept of a CPU is ambiguous here, as some meant a socket and some others meant a core or even a logical compute thread or a virtual CPU like in a VMWare slice. My convention is that a socket is equivalent to a processor, whereas a CPU could be just an independent computing unit, be it a physical core, or a logical thread, or a vCPU. The importance is to understand what's behind that thing that is called a CPU, because of the obvious performance disparity among a physical core, a logical thread, and a vCPU).

The real question here is whether a server software product can scale up as it is claimed with more CPUs. For server products like Oracle that have huge real customer

Key Feature Summary	Express Edition 10g	Standard Edition One	Standard Edition	Enterprise Edition
Maximum	1 CPU	2 Sockets	4 Sockets	No Limit
RAM	1GB	OS Max	OS Max	OS Max
Database Size	4GB	No Limit	No Limit	No Limit
Windows	●	●	●	●
Linux	●	●	●	●
Unix		●	●	●
64 Bit Support		●	●	●

Figure 13.1 Oracle 11g R2 resource scalability limitations.

bases, there are benchmarks designed specifically to prove one way or the other. With Oracle, I haven’t experienced a single case that it did not scale with more CPUs. I had an impressive experience that when an Oracle-based enterprise application was moved from a four single-core processor server to a two quad-core processor server while keeping the total CPU GHz power and the workload the same, the throughput of the test load doubled. This to some extent helped prove that Oracle could execute efficiently on modern multi-core CPU servers.

13.6 PARTITIONING

Partitioning is a standard technique aimed at facilitating the maintenance of a large database and in the meanwhile can potentially improve the performance and scalability of an enterprise application. Almost all database products more or less support partitioning, but the variety of partitioning methods built into Oracle is comprehensive. Range partitioning, hash partitioning, list partitioning, interval partitioning, and composite partitioning as the permutations of these partitioning methods, as we introduced in the preceding chapter, should satisfy the needs of most of the today’s enterprise applications.

13.7 AN ALL-ENCOMPASSING, POWERFUL PERFORMANCE, AND SCALABILITY TROUBLESHOOTING TOOL—AWR

The AWR feature available since 10g has been covered in depth in Chapter 11. It is a very useful tool not only for optimizing and tuning the performance and scalability of an Oracle-based enterprise application internally but also for effectively and efficiently resolving customer production performance escalations. Part Four, “Case Studies: Oracle Meeting Real World Performance and Scalability Challenges,” is a reflection of how useful it has been in helping me resolve the performance and scalability issues of the Oracle-based enterprise products I have worked on.

This AWR tool could be used more efficiently if combined with a good understanding of queuing theory and a data-based performance and scalability troubleshooting approach. Otherwise, a great tool that is considered very useful by some people could be deemed totally useless by some others, depending on one’s background and experience.

13.8 THE MOST COMPREHENSIVE SET OF INTERNAL PERFORMANCE METRICS

The most striking Oracle feature to me is its comprehensive set of internal performance metrics built in and enhanced with each release. These metrics are available either through the V\$ performance dynamic views or AWR reports, as we introduced previously. This makes me feel Oracle is more like science than black art whenever I work on an Oracle performance and scalability issue. To some extent, this also makes Oracle a preferred educational platform to inspire today's college students to solve tomorrow's computing challenges in database arenas.

13.9 DATABASE RESIDENT CONNECTION POOL

This new feature in Oracle 11g was introduced in the previous chapter. It's worthwhile to consider because potentially it can help your Web application support more users. However, first make sure that it's applicable to your application, and secondly test it thoroughly in your test environment before pushing it to your internal test environment or external customer's production environment, as this is a fairly new feature. In addition, it could potentially help you save the need for a RAC if it's really applicable to your Web application.

13.10 IN-MEMORY DATABASE CACHE (IMDB)

Oracle In-Memory Database Cache (IMDB) is an Oracle database product option that enables a critical subset of an Oracle database preloaded into the memory of an application to totally eliminate disk access for this subset. An IMDB is supported by the Oracle Times Ten In-Memory Database technology, which serves as an IMDB's RDBMS engine. Since an IMDB is a subset of a regular Oracle database, most of the RDBMS concepts and principles still apply. For this reason, we won't cover more about it here, and you should consult Oracle's IMDB documentations if you are interested in knowing more about it.

13.11 SUMMARY

In this chapter, I offered a top 10 list of Oracle performance and scalability features available as of 11g R2 from my point of view. These features range from the top notch RAC technology to common features such as proven transaction and concurrency models, a highly efficient SQL optimizer, a powerful performance and scalability troubleshooting tool, and so on. I'll present more quantitative case studies later to help corroborate some of the performance and scalability features introduced here.

In the next two chapters, we'll focus on actually developing an Oracle-based secure online banking application (SOBA) with an end-to-end, piece-by-piece approach.

This sample application will not only help demonstrate the full development life cycle of an Oracle-based application but also serve as a valuable educational and experimental tool for exploring Oracle performance and scalability features further. I have decided to take this sample application—SOBA—much further than just another Oracle sample schema or a stand-alone backend tier: The application tier and Web tier are coded with one of the most widely used Java development platforms, Spring Source (version 3.0), in conjunction with some of the standard Web technologies available today. It would be a very exciting project, as I will even bring you into some of the very hot new software technologies such as RESTful Web services, and so on. By getting our hands dirty with a fully fledged project, we'll learn more than just reading and doing a few exercises or trying out a few SQLs at a SQL*Plus command line.

RECOMMENDED READING

The best texts are Oracle's own documentations accompanying each release. These documentations are not only available online but also much more authentic. For the latest version of Oracle 11g R2 as of this writing, all documentations are accessible from the following URL:

<http://www.oracle.com/pls/db112/homepage>

From this Web site, you can get a list of all 11g R2 documents. The following documents are strongly recommended for understanding Oracle performance and scalability features:

- *Concepts* document in HTML or PDF. (It's always important to understand the concepts first, since they are the basic elements for associative and creative thinking)
- *Administrator's Guide* in HTML or PDF. (I like Oracle's Admin's guide very much, because it actually starts out by explaining clearly Oracle architecture first before giving procedures on how to carry out certain administrative tasks.)
- *Performance Tuning Guide* in HTML or PDF. (This document contains all Oracle performance tuning tips.)
- *Oracle Real Application Clusters* from the left frame under Grid Computing

To learn more about Oracle's transaction and concurrency models, refer to Chapter 6 (Locking and Latching), Chapter 7 (Concurrency and Multi-versioning), Chapter 8 (Transactions), and Chapter 9 (Redo and Undo) of the following text:

T. Kyte, *Expert Oracle Database Architecture: 9i, 10g and 11g Programming Techniques and Solutions*, A Press, New York, 2010.

EXERCISES

- 13.1** List all major components for setting up a RAC environment. Explain why each component is needed.
- 13.2** List the pros and cons of using a RAC. What would be the most-anticipated obstacles in setting up and maintaining a RAC?

- 13.3** How do you decide whether you would choose dedicated or shared server mode for your Oracle database? If you have experience in using shared mode, what's your view on the controversial subject of dedicated mode versus shared mode?
- 13.4** Explain the roles of row-level locking and multi-versioning in Oracle's transaction and concurrency models.
- 13.5** If you have been working with Oracle covering several generations of hardware, what's your experience in how Oracle keeps up with the advances in hardware technologies?
- 13.6** When you encounter an Oracle performance or scalability issue, which tool will you rely on to resolve the issue?
- 13.7** What are the differences among dedicated server, shared server, and DRCP (database resident connection pool)? If an Oracle-based application is maxed out with hundreds or thousands of users, what would you recommend to help scale it up to supporting tens of thousands of users or even more?
- 13.8** What are the pros and cons of using an IMDB (In-Memory Database)?

14

Oracle-Based Application Performance and Scalability by Design

“It’s different to be in a ring.”
—Anonymous

The scalable performance of an Oracle-based enterprise application just doesn’t come as an accidental windfall. A more carefully designed Oracle-based product with performance and scalability taken into account from the beginning is more likely to perform and scale after it is built and delivered to customers. Furthermore, the later a performance or scalability issue is discovered in the development cycle of a product, the more costly it is to fix. Think of such a scenario: A large-scale enterprise application consisted of multiple subsystems; whereas all edge subsystems were multi-threaded, the subsystem acting more like a hub for the entire system was single-threaded. Was this a scalable design? Probably not. A complicated software system is similar to an ecosystem that all parts need to support each other in order to coexist. An equilibrium condition is formed when no single part could easily become a single point of failure or bottleneck. Such an equilibrium condition is necessary so that the whole system can sustain stably.

Now the question is how one can design the performance and scalability into an Oracle-based application by following a proven software engineering methodology. At the highest level, one needs to reach a proper balance among the three axes for a software project: *cost*, *schedule*, and *quality*. How much weight one should put on each axis depends on many factors, such as how innovative the idea about the product is, how fast the competitors will be able to catch up, the size of the potential market, and the expected turnaround in terms of cost recovery and profit margin, and so on. Although most of these aspects, especially *cost* and *schedule*, belong to the managerial category, the *quality* of the actual product is more of a technical issue—both from the *process* and *development* perspectives. While we let a great management team make calls on cost and schedule, let's see what it takes for a capable technical team to make a high quality product—with both performance and scalability taken into account through the entire development life cycle of a product.

Taking aside the gory technical details, let's first explore the traditional, waterfall, spiraling software development methodologies versus more recent rapid development methodologies. It appears that the traditional methodologies are more suitable for developing platform-type products such as operating systems, middleware systems, and database systems, and so on, that are foundations on which applications are built, whereas the more recent rapid development methodologies are more suitable for developing application-oriented products. It also appears that what development methodology to use has something to do with the maturity of the product under development. It's less pressing to release version $N + 1$ from version N , say, with $N > 5$, with a more mature product than to release version 0 or 1 with a new product. Let's next explore the advantages and disadvantages of the various rapid development methodologies, which are most pertinent to developing Oracle-based applications. This would serve as a precursor to the remainder of this chapter, which will cover how to design performance and scalability into an Oracle-based application.

The following subjects are discussed in the remainder of this chapter:

- Rapid Development Methodologies
- Planning
- Requirements Gathering
- Conceptual Design via Data Modeling
- Logical Design via Normalization
- Physical Design
- Implementation
- Release to Market (RTM)
- Continuous Improvements

Note that this chapter and the next chapter will be heavy on programming—both in SQL and in Java. With Java, we'll rely mostly on one of the most popular open source Java development platforms—*Spring Source*. We will also step upon some of the latest hot technologies such as RESTful Web services. Spring has a very large base of

software developers, RESTful Web services is the moving trend, and Oracle is the best database technology, so we are all set to have fun (also consider this an opportunity for acquiring a valuable set of skills in developing Java and Oracle-based applications).

Let's start with Rapid Development Methodologies next.

14.1 RAPID DEVELOPMENT METHODOLOGIES

To set the expectations properly, I have to mention that this section is not a detailed introduction to every major, modern rapid software development methodology. The overview of each rapid development methodology will be concise, yet sufficiently clear for you to realize which development methodology is followed for your product under development. Knowing what development methodology is followed with your project might help position yourself better in working with your team and making great contributions to your project.

Here is a brief introduction to each of the major rapid development methodologies (you might have heard a lot about the first three but less about the last three):

- *Agile Development*. This is an iteration-based, or drop-based development methodology, measured chronologically in roughly every two weeks. Because of its agility nature, it depends more on less formal communications such as oral or face-to-face communications than formal documentations. With this methodology, formal documentations are worked on during the final stage of the product release. The advantage is that it's *agile* so that it could be more effective and efficient; but it could be a two-edge sword that it may turn out to be less effective and efficient if the team size is too large or the team members are less self-disciplined.
- *Extreme Programming (XP)*. The main goal of XP is to reduce the cost of change. This methodology encourages starting with the simplest solution and adding more features phase by phase with time. Its focus is on designing and coding for the needs of today instead of those of tomorrow or the future. Designing and coding for uncertain future needs implies the risk of spending resources on something that might not be needed. The advantage is that it addresses the needs of today, but it might miss big in the long run.
- *Scrum*. This is a role-based methodology with three main roles:
 - The "Scrum Master" who acts as the process orchestrator in the role of a project manager.
 - The "Product Owner" who represents the business.
 - The "Team" who includes all members responsible for requirements analysis, design, implementation and testing, and so on.

The scrum methodology measures the project progress in *sprint*, which is typically 2 to 4 weeks. The goal of each sprint is to create a shippable product increment, but it's very unlikely to achieve such an over-optimistic goal if the size of the project is sufficiently large. The other characteristic of this

methodology is that the entire product backlog is divided into multiple sprint backlogs to make it more manageable and better prioritized. This methodology is probably the most widely used one today.

- *Lean Development.* This methodology emphasizes delivering faster with a trade-off of less functionality—the thinking that 80% today is better than 100% tomorrow. This is typical with those products that are released in *generations*. This is a strategy of snapping the market share first, but it may backfire if the product is less attractive because of the missing features.
- *Joint Development.* This methodology engages the customer throughout the entire life cycle of the product development. Typically, a resident customer liaison is placed at the vendor's site so that what is developed is exactly what is needed by the customer. This methodology works well for custom-made products, either new or legacy replacement, such as billing software products for cable or mobile phone companies.
- *Prototyping Development.* This methodology is more appropriate for proof-of-concept type of projects. It typically involves a long development cycle, large up-front cost and longer ROI (return on investment) period. It helps mitigate the risks of adopting newer technologies to solve extremely challenging tasks. Replacing a large-scale application built with the older technologies of 20 to 30 years ago falls into this category.

As is seen, there is no one single development methodology that would fit all types of projects. However, it doesn't mean that what methodology to use is not a critical issue. Perhaps a hybrid methodology that combines the relevant advantages of each methodology would work best, but that's beyond the scope of this text.

In the remainder of this chapter, let's take a look at all the necessary phases and measures for developing a highly performing, scalable Oracle-based enterprise application, regardless of which development methodology is adopted. Let's start with *Planning* first in the next section.

14.2 PLANNING

The key to planning is to plan well ahead of time. What to plan will be described in the next few subsections, from forming a vision, through defining objectives, conducting ROI analysis, carrying out a feasibility study, and, finally, forming a project team. These are non-technical issues, but to some extent, they could be much more critical than technical issues, because a well-planned project could contribute as much as up to 80% of the success of a project. Let's expand into each subject next.

14.2.1 Vision

A great vision is perhaps the single, most important element of a great product family. Well-known examples in this regard include Bill Gates' vision on PCs and Steve Jobs'

vision on Apple's i-everything (i.e., iPod, iMac, iPhone, iPad, . . .), both of which have overwhelmed the entire world in their respective time frames. Behind these visions is the exceptional execution ability of such exceptional individuals. The other equally impressive example is Google, which was not the first to start the Web-based search business but finished the last with an unshakable position in its territory. All these examples have proven one thing: A great vision with exceptional execution ability can lead to unprecedented successes.

However, for most software projects, the scope and scale are much smaller. But still, a very capable high-level management team with the right vision is the key to the success of a product. Without a right vision behind a product, the project would eventually become an unsuccessful, costly exercise. Unfortunately, there is no written prescription about how to come up with a great vision, which, in my opinion, is more of inborn than can be taught or learned. So let's just assume that the right vision is there and how we can help make it true.

14.2.2 Objectives

Defining objectives is about what you set out to achieve with a driving vision behind a project. This should encompass:

- Targeted customer or user base
- The size of the potential market
- The place as one of the core components in a bigger product portfolio of your company
- All necessary functionality features
- All major non-functional requirements (for example, supporting up to N users or a certain transactional volume, and so on.)
- And other relevant exit criteria

Note that defining objectives is a kind of activity decoupled from actual technical implementation. It is driven more by the envisioned potential commercial prospect. Therefore, the next natural step is to conduct an ROI analysis, as described next.

14.2.3 ROI Analysis

Apparently, developing a software product is a costly exercise in all aspects, from covering manpower, to testing equipment, to acquiring other necessary software products, and so on. Therefore, a careful analysis on the cost and expected return is indispensable for all software projects. It would be disastrous if a project runs out of funding midway, or takes forever to recover the investment and start to see profit.

Like the software performance and scalability engineering approach proposed in this text, an ROI analysis should take a data and fact-based approach as well, rather than an opinion-based approach. At the end, it all comes to whether the projected ROI proves to be sufficiently accurate. However, how to conduct an ROI analysis is a

non-technical issue, and we'll defer it to the expertly professionals to deal with. What we need to know, though, is that an ROI analysis makes various assumptions, and therefore a supporting feasibility study needs to follow, as is described next.

14.2.4 Feasibility Study

The non-technical aspects of a project feasibility study include the preliminary estimates of time, staff, and materials with given time frame and budget. On the technical side, a critical task is to identify potential technological barriers to overcome in order to achieve some of the non-functional requirements, for example, what will it take for the product to be capable of supporting 10,000 concurrent users or 40 million customer accounts? More tedious issues may include what hardware platform and OS platform to run the product, what middleware and database platforms to build the product on, what development platform to settle on (for example, Java or .NET versus other programming platforms), and so on.

This is another classical quandary. In order to prove something is feasible, you need to build it first; but the game is to prove it *a priori* without actually building it. My opinion is that there is no silver bullet, but sufficient knowledge about the state of the art capabilities of modern hardware, real experiences with all relevant, building-block, supporting software technologies, and a professional level mastering of some analytical methods such as queuing theory, can all work much better than purely blind guesses. Some of the quantitative case studies in this text and many other product specific benchmark reports might be very helpful in this regard.

Finally, a great vision needs exceptional execution to make it successful; and the core part of the exceptional execution of a great vision is a very capable project team. This subject is lightly touched upon next.

14.2.5 Project Team Formation

How effective, efficient, or productive a project team is crucially determines how successful the project will be. Whether forming a new team or adapting an existing team, the following factors should be taken into account:

- *Expertise Coverage.* Developing a sufficiently complicated software product requires multiple types of skill sets. It's necessary to identify at least one expert on an area-by-area basis. Such experts will not only do some actual work, but also lead less experienced developers who are able to flawlessly carry out the assigned tasks with given instructions or guidance otherwise. Note that some common math doesn't apply here: A real expert might be able to solve a problem in minutes or hours, which may take much longer if the same problem were given to 5 or 10 inexperienced members, so the inequalities of $1 > 5$ or $1 > 10$ could miraculously hold true here.
- *The Proper Team Size.* It's generally agreed that it's more effective and efficient to have a team with fewer but highly professional members. There are lessons learned that keeping adding more people to an already late project will only

delay it further. The goal should be to maximize the productivity while keeping the team size as small as possible.

- *Team Culture.* Software development is a kind of human activity that an invisible team culture can crucially determine whether a team would eventually succeed as expected or fall apart unexpectedly. Ultimately, it's the major responsibility of a team manager to set up an amiable atmosphere for everyone to work in. According to my personable experience, this is a much more important quality a manager should have than being more technical about something. Keep in mind that a failed project will eventually translate into the loss of the company, so everyone on a team, whether the manager or individual members, should always take an objective approach to various technical issues, and work together to the common goal of finishing the product successfully for the company everyone is committed to.

Perhaps this is a good turning point that we get more technical and specific about the goal of this chapter, namely, how we can design performance and scalability into an Oracle-based enterprise application. To achieve this goal, I have conjured up a sample application named *Secure Online Banking Application* (SOBA) as a concrete base for demonstrating the entire process of building an Oracle-based application. This is a more effective learning approach, as it's less effective to keep talking at the 10,000-foot-high level without touching anything real. I'll take a step-by-step, end-to-end approach to building this online banking example so that it will be clear how SOBA is built at each step and what functions it has. We'll adopt some exciting technologies such as RESTful Web services based on one of the most popular Java development platforms—Spring Source. In the meanwhile, I'd like to set the expectation properly that this will not be a full-blown, commercial-grade online banking application (a real one may take tens of millions of dollars to build).

Let's start with the requirements gathering next.

14.3 REQUIREMENTS GATHERING

A formal requirements gathering process would consist of such activities as interviews, surveys, observations, document reviews, and so on. Since this online banking example is only a sample application, we'll only do what's required minimally here to help you understand what will eventually be implemented into this sample application.

The functional requirements for this sample online banking application are really simple, as described below:

- It would allow a user to log in and check balance and account activities.
- It would allow a user to transfer funds among different accounts (e.g., between checking and savings, etc.).
- It would allow a user to submit online bill payments.

Some of the non-functional requirements may include, if it were a real product:

- Supporting up to N concurrent users where N could be as large as millions
- Generating monthly statements for all accounts within a nightly window of, for example, from 2:00 AM to 5:00 AM.

But since this is a sample application, such non-functional requirements will not be stretched, partially because of lacking the required proper hardware for testing.

Next, we'll present a few use cases for this sample application.

14.3.1 Use Cases

A use case is a schematic diagram or a script that conveys a specific usage of a product. A collection of use cases gives a customer or an executive a comprehensive view about what can be achieved with the system to be built. It also serves as a guideline on devising design, implementation, and testing strategies during the life cycle of a product.

A formal method of creating a use case is to use the Unified Modeling Language (UML). The first step to create a use case diagram in UML is to identify three basic elements: the actor, the system, and actions, as depicted below:

- An *actor* is an external initiator of a series of actions exchanged between a user and a system. For example, in our online banking example, a user may log onto the system, perform some tasks such as checking account balance, checking the status of an online bill payment, transferring funds between the checking and savings accounts, and so on, and finally logging out after done. In this case, the user is the actor.
- The *system* that receives user requests and sends responses back to the user. In our online banking example, the application itself is the system.
- The *actions* between a user and the system from a user's perspective. In our online banking example, actions may include login/logout as well as various typical activities associated with an online banking account as listed above.

Also note that in modern terms, an *actor* is also called a *persona* and a *use case* a *user story*. No matter how the terms change, the semantics remain the same.

With this sample banking application, a typical use case may consist of three actions: login, check balance, and logout. Other use cases can be derived based on other functions of the same application. We will not repeat all those nuances as they are too intuitive. But for a real product, all use cases must be created and documented clearly, as real development activities depend on the valid use cases defined for the product under development. For example, two very different types of tests need to be devised and executed: verification and validation. Verification confirms that the system behaves as designed, whereas validation assures that the developed functionality is what customers expect.

After all the use cases are clearly defined, the next step is to develop user views based on the use cases defined, as introduced next.

14.3.2 User Views

A user view can be considered a more detailed elaboration of the use cases defined. It can be expressed in Web pages, screens, forms, or reports that what a user would see after the product is put in use. For example, a user view might show the Web page after a user logs into this online banking sample application. Other user views may include the login/logout pages, bill payment page, transfer page, and so on. Since this is a simple, sample application, listing the actual user views in a series of Web pages would occupy too much space, so they are omitted here.

A collection of user views can also be considered a *user interface model*. By examining a user interface model, we can extract business processes and business entities, which are discussed next.

14.3.3 Business Processes, Entities, and Business Rules

In a broader sense, a business process is a series of activities or tasks that is exposed to a customer as a service when performed together. In a typical organization, there are three types of business processes: (1) the operational processes that constitute the core business embodied as multiple revenue generation streams, (2) the management processes that govern how the business is run under an effective model, and (3) the supporting processes that support the core business processes and management processes as a necessary overhead. However, as we are dealing with a simple sample online banking application here, we will not be concerned with the complexities associated with all those types of business processes.

Specific to our sample online banking example, a business process can be derived from a user view as discussed in the preceding section. A business process can be best depicted and visualized with a flowchart that describes a series of activities involved. Figure 14.1 shows a flowchart representing an online bill payment submission process.

There are two dependent concepts without which a business process cannot exist: *entities* and *business rules*. An entity is a concrete object that a business process can work on, whereas a business rule restricts what business tasks can be operated on in a business process. Using the online bill payment process depicted in Figure 14.1 as an example, we can identify such entities as a *customer*, an *account*, a *biller*, a *bill*, and so on. Using the same bill payment process, we can identify such business rules as that *a customer's account must be active in order to be accessible online*, and that *an account must maintain a proper balance in order to pay a bill*, namely, no overdraft, and so on.

Figure 14.2 illustrates how entities and business rules can be derived from a business process. As will be seen in the next few sections, database design is based on the concept of entities, whereas database implementation is based on not only entities but also business rules. Business entities and business rules play pivotal roles in designing a database at various levels.

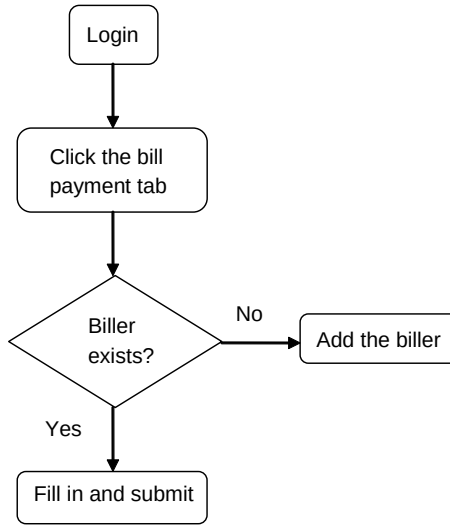


Figure 14.1 A bill payment business process.

With given use cases, user views, business processes, entities and business rules, the next step is to carry out the design of a database. As one of the database design best practices, one needs to follow a three-step design process, namely, the *conceptual design*, *logical design*, and *physical design*. Each of these design steps is described in the next three sections. Let's begin with the conceptual design via data modeling next.

14.4 CONCEPTUAL DESIGN VIA DATA MODELING

Data modeling is the main theme of the conceptual design phase of a database. It is a process of creating a conceptual data model to capture the semantics of the data in a more formal format. Then a conceptual data model is transformed into a logical data

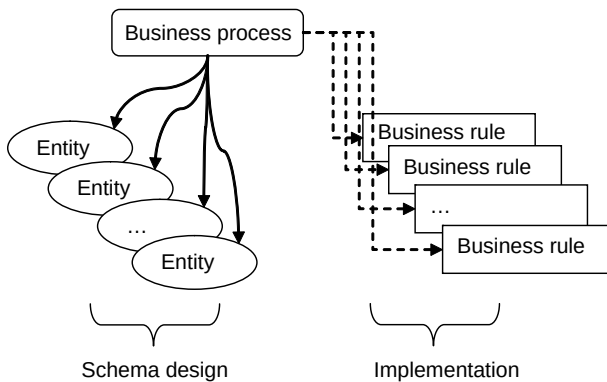


Figure 14.2 Database schema design and implementation based on entities and business rules derivable from a business process.

model, which is further normalized and refined during the logical design phase of a database. A logical data model is eventually converted into an implementable physical data model that is database system specific (Oracle, SQL Server, DB2, MySQL, etc.).

Conceptual design is also called external design, as this step only provides a black-box view of what the data model should look like at the high level. Although a conceptual data model can be expressed with flow charts, storyboards, and screen flow diagrams as well, the entity-relationship diagram (ERD) is the most standard format. Therefore, in this section, we will focus on how to use the ERD format to describe the conceptual data model associated with our sample online banking application.

14.4.1 Entity-Relationship Diagramming

An ERD is an abstract and conceptual representation of data. How to compose a data model using the ERD format is the main task of data modeling. The major objective of data modeling is to reflect the proper relationships among various participating entities so that the essence of the problem domain is captured in the data model produced. Therefore, creating ERDs is the process of data modeling itself. Next, we apply data modeling to our sample online banking example. Let's begin with a review of some of the basic concepts and conventions related to an ERD in general first.

First, there are three basic elements with an ERD: *entities*, *relationships*, and *attributes*. In the previous section, we already explained that an entity is a concrete object that a business process can work on. However, we have not explained that an entity can have attributes that define what an entity is about. For example, with our sample online banking application, we could have the customer and account entities. The customer entity can have attributes like name, social security number (SSN), address, email, phone number, and so on, whereas the account entity can have attributes like balance, status, and so on. An entity would be meaningless without accompanying attributes.

The *relationship* part in the term *entity-relationship diagram* associates various entities in a meaningful way. A bunch of stand-alone entities would be useless if they were not associated with each other in proper business context. For example, the two entities mentioned above, a *customer* and an *account*, can form a relationship or bind together in the format of *a customer accesses his account* where the verb *accesses* is the "relationship" in this case.

Note that the definition of a relationship is incomplete without imposing a *cardinality* constraint. The cardinality constraint of a relationship quantifies how many entities at both ends can be associated with each other. For example, a customer can have one banking account or more, whereas a banking account can be owned jointly by more than one person. In this case, the customer-account relationship is said to be a many-to-many or *N:N* relationship. However, to make it simple, we'll restrain it to a one-to-many relationship with this sample application, namely, one customer can have multiple accounts, but one account can be assigned to one customer only. This one-to-many relationship is the most common one out of all possible types of relationships.

Not surprisingly, there are many formats for depicting an ERD. The most intuitive format is called Chen's format proposed by Peter Chen in 1976. Some of the general conventions with this format are as follows:

- An entity is drawn as a *rectangle*. However, some use the singular form and some use the plural form for an entity. In this text, the singular form is used.
- A relationship is drawn in a *diamond*. It could be expressed in one way or both ways. In this text, one way is used, since the other way is implied and self-explanatory.
- The attributes of an entity are drawn as *ovals* connected with a line to the owning entity. Note that a relationship can have its own attributes as well.

In addition to the concepts of entities, relationships, and attributes, the cardinality and ordinality of a relationship are specified in an ERD as well, typically in the format of *min:max* where *min* denotes the ordinality (whether the relationship is mandatory ($\min \geq 1$) or optional ($\min = 0$)), and *max* denotes the cardinality (the maximum number of instances an entity can be associated with). Figure 14.3 illustrates the ERD for the *customer-has-account* relationship that is pertinent to our online banking example. According to the conventions explained above, this ERD denotes a bi-relationship between a customer and an account that (read from left to right) *a* customer has 1 or N accounts, whereas (read from right to left) an account is owned by one and only one customer.

The next ERD format that is very popular is called the Information Engineering (IE) format. It was developed in the 1970s to 1980s. It is briefly introduced in the next section.

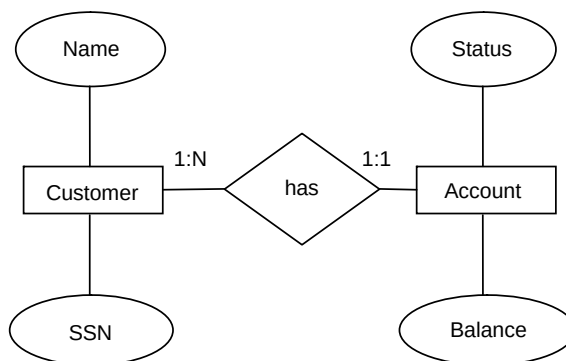


Figure 14.3 An ERD representing the relationship of a customer has an account with proper ordinality and cardinality labeled in *min:max* notation as well.

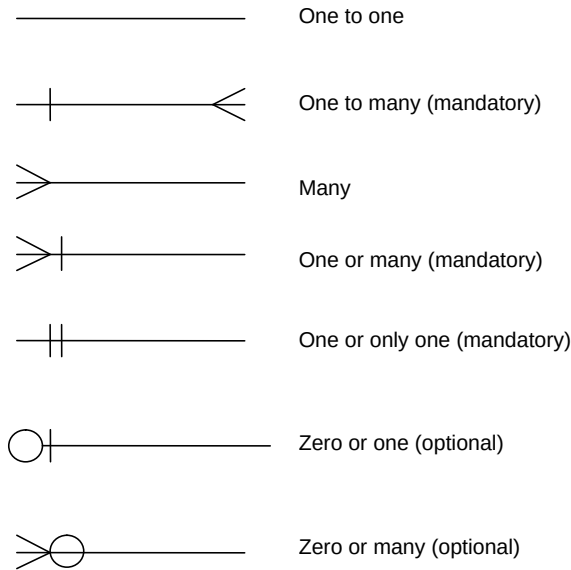


Figure 14.4 IE notations for ERDs.

14.4.2 The Information Engineering (IE) Format for ERDs

Figure 14.4 shows the IE notations for ERDs. The noticeable notational implications are:

- A crow's foot represents *many*
- A bar represents *one* and two parallel bars represent *one and only one*
- And a circle represents *zero*. Note the ordinality with each relationship, which represents whether the relationship is mandatory or optional.

See Figure 14.5 for the same customer-account relationship drawn in IE format. However, rather than struggling with many different ERD formats, it makes sense to

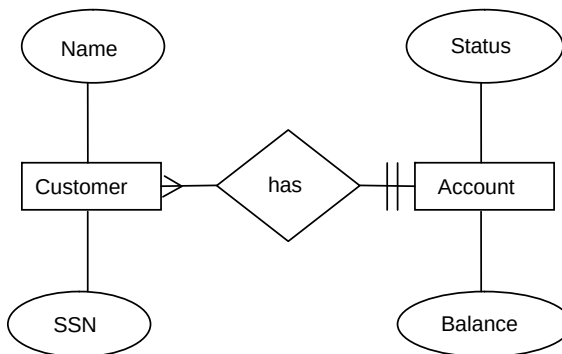


Figure 14.5 Same customer-account relationship drawn in IE format.

settle down on using UML (Unified Modeling Language). This is the subject of the next section.

14.4.3 UML Format for ERDs

There are a few advantages with UML format for ERDs:

- A database eventually needs to interact with an application server, and the application that runs on the application server is typically designed using the object-oriented programming (OOP) model. Using UML to describe a data model provides a natural integration path into the OOP paradigm.
- Compared with the IE format, UML format uses numbers to denote cardinality, which is more intuitive and precise than using symbols.
- UML is a more standard method, which means it would be easier for others to understand. Besides, your UML skill will be more reusable in the long run.

Table 14.1 describes how ERD terms can be mapped to UML terms. Anything that can be expressed in ERD can be expressed in UML as well.

Now the ERD shown in Figure 14.3 can be redrawn in UML. See Figure 14.6. Note that unlike an ERD, the cardinality or multiplicity near a class refers to the destination class. Now it reads (from left to right) a customer has zero to many accounts, or (from right to left) an account is owned by one to many customers. In addition, “0..*” can be simply denoted as “*”, which means “zero or more”. Another convention is that if no multiplicity is specified, then it is one (1) by default.

To know more about the various standard notations with UML, refer to the UML reference book listed at the end of this chapter or any other UML texts. Next, let’s see how a data model can be expressed in relational format.

14.4.4 Relational Format for ERDs

Some of the very basic ERD mapping rules to be observed are as follows:

- An *entity* is mapped to a *relation* or a *table*
- *Attributes* are still mapped to *attributes* or *columns*
- *Relationships* are mapped to *foreign keys*

Using the above mapping rules, we can come up with the relational data model shown in Figure 14.7, which is equivalent to the data models illustrated in Figures 14.3

Table 14.1 Mapping between ERD and UML

ERD Term	UML Term
Entity	Class
Relationship	Association
Attributes	Attributes
Instance of an entity	Object
Supertype/Subtype	Generalization

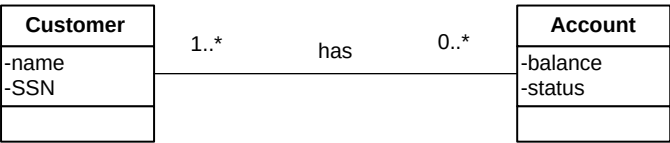


Figure 14.6 Data model expressed in UML.

through 14.5. Note that we have added an ID attribute to each entity to help illustrate the relationship between the two. These two IDs are labeled PK (primary key) on each of the tables. One additional label FK (foreign key) links the account table to the customer table. Also, in terms of the cardinality and ordinality, the arrowhead of the connection line denotes *many* while the plain end of the connection line denotes *one*, which is omitted by default.

You might wonder if this is all what we are going to implement with this sample online banking application. Isn’t it over-simplified if we just have two tables and a few attributes for each table? No. The purpose of having such an over-simplified model is to make it easier to illustrate the concepts associated with the underlying data models, which doesn’t depend on the number of tables or attributes to some degree. In reality, a table may have tens of attributes or even hundreds of attributes, which is too big to be manageable in an ERD during the conceptual design stage.

Next, we’ll demonstrate how the logical design of a database should be carried out. The outcome during this phase should be a properly normalized data model that can be implemented during the physical design stage. Let’s explore how we can come up with a normalized data model for this online banking application.

14.5 LOGICAL DESIGN VIA NORMALIZATION

The conceptual design phase focuses on identifying entities and the inter-relationships among various entities, depicted as ERDs as viewed from a user’s perspective or externally, whereas the logical design phase focuses more on *data consistency* so that the data model will function properly after implemented and deployed in production. Logical design is also known as *internal design*, as the gory details of it will never be seen by customers other than by developers.

The major activity in the logical design phase is *normalization*, a procedure that is necessary to help avoid many types of operational anomalies associated with

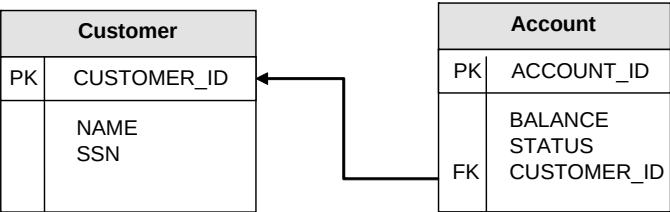


Figure 14.7 The relational data model for the online banking sample application.

INSERT, DELETE and UPDATE SQLs. In order to demonstrate a full normalization process, we'll deviate from the conceptual design introduced in the previous section and starts with a single big table named ACTIVITY that records the daily transactions in all accounts of a customer (note that this big table spans three business entities: CUSTOMER, ACCOUNT, and TRANSACTION). This big table is shown below, with the first column assigned as the primary key:

```
ACTIVITY: ActivityID (PK), CustomerID, CustomerName, CustomerAddress,
         CustomerCity, CustomerState, CustomerZipCode, CustomerPhone,
         TransactionDate, CheckNumber, TransactionType,
         TransactionDescription,
         Debit, Credit, AccountID, AccountName, AccountType,
         AccountDescription,
         AccountTerms, AccountStatus, AccountBalance
```

Next, let's see what operational problems or anomalies may occur with this one big table and thus justify the need for performing normalizations.

14.5.1 Operational Anomalies

Just by looking at the columns of the preceding big table, we can anticipate the following anomalies:

- *Delete Anomaly.* It is seen that none of the activities could be deleted or moved off no matter how long ago it occurred, as that would delete the customer completely from the database. The underlying cause for this delete anomaly is that all the customer data is resident in the same ACTIVITY table.
- *Update Anomaly.* The same underlying cause for the delete anomaly as mentioned above applies to the update operations as well: If a customer moved to a new place, then *each row* of the ACTIVITY table for that customer would have to be updated. This would result in a serious performance problem if the database were large.
- *Insert Anomaly.* One could not create a new account for a customer unless the customer already has an account, since the account entity and the customer entity are embedded in the same ACTIVITY table.

So, how could we resolve these anomalies? The general logic is that *anomalies come from redundancies and redundancies are caused by improper dependencies*. The common practice is to split those entities clogged in the same table into multiple tables through a normalization procedure (a normalization process is also a factorization or decomposition process). The more we decompose, the higher the level of normalization we go. However, decomposition of a table should not be an aleatory exercise. All decompositions must preserve the information in the original table, which means that one must obey certain rules when normalizing.

Normalization is a procedure that can be rigorously described mathematically. The next section provides a brief review of the relation theory, which governs a

normalization process. The purpose of providing an introduction to relation theory is that mathematics is the only format that can help define things unambiguously.

14.5.2 Review of Relation Theory

Relational database management systems are built on the data models described in relation theory published by E. F. Codd in 1970. Relation theory is based on the first-order logic, which is a formal logical system used not only in computer science but also in linguistics, mathematics, and philosophy, and so on. First order logic deals with predicates and quantification. It provides us with a method for reasoning about properties shared by many related objects. To understand how it works, let's take a look at the following example.

Let's state an arbitrary fact that *a cat is an animal*. This statement is a proposition. To make it generic, let's use a variable x to represent a *thing*. Now, we can more formally say that *if the thing that x represents is a cat, then it is an animal*. In first-order logic, it is expressed as $Cat(x) \rightarrow Animal(x)$, which asserts that if x is a cat then it is an animal. Note that this proposition could have a value: true or false, meaning that it's exact or quantitative.

It may not be too useful if x is limited to one thing or a few things only. We want to extend it to all things that x represents, namely, no matter what x represents, so that if x is a cat, then x is an animal. The above proposition can be further expressed in a first-order logic sentence as follows:

$$\forall x (Cat(x) \rightarrow Animal(x))$$

Here the symbol " $\forall$ " is called a universe quantifier which means all possible values of x when it is applied to x like $\forall x$ (note that " $\forall x$ " is read as "for all x ").

Just to remind you, going from the left side to the right side with the above proposition is called a *sufficient* condition that *if x is a cat then x is an animal*. However, the reverse may not hold as a *necessary* condition, namely, *if x is an animal then x is not necessarily a cat*—it could be a dog or some other animal from a different species.

To see how first-order logic is related to relational databases, let's illustrate how a key is defined for a table. If an attribute or an attribute set uniquely defines a row of a table, then that attribute or attribute set is a *superkey* of that table. Let's further state the following:

- *A Trivial Superkey.* If a table has to use all of its attributes to uniquely identify every row, then it's said that it's an *all-key* table. The entire attribute set in this case is called a *trivial superkey*.
- *A Candidate Key.* If a superkey is composed of a single attribute or a least combination attribute set, then the superkey becomes a *candidate* key. In other

words, a candidate key does not contain the attributes that are not required for uniquely identifying every row of a table. Also, a candidate key would become a non-key if one attribute is removed from it.

- *A Simple Key.* If a candidate key has only one attribute, then it becomes a *simple key*.
- *A Compound Key versus a Composite Key.* If every attribute of a candidate key is a simple key in its own right, then the candidate key is called a *compound* key. In contrast, if not every attribute in a candidate key is a simple key in its own right, then the candidate key is called a *composite* key.
- *A Secondary or Alternative Key.* If a candidate key is designated as a preferred key, then it's called a *primary key*. All other candidate keys are called *secondary* or *alternative* keys.
- *A Foreign Key.* If an attribute of a table is a primary key of another table, then this attribute is called a *foreign key*. A foreign key is used to enforce referential integrity between two tables.

Having seen that all of the above statements are in the format of *if X then Y*, we are convinced that all relational databases are indeed deeply rooted in first-order logic, although it's true that the concept of a key is not what a relational database all about.

We have to state that the symbol " $\rightarrow$ " used in the first-order logic example above is reserved for something else in relation theory, in which case, it represents a functional dependency (FD). In relation theory, if an attribute of a table, say X, uniquely determines another attribute of the same table, say Y, then we say "attribute X determines attribute Y" or "attribute Y is functionally dependent on attribute X." This functional dependency is expressed as " $X \rightarrow Y$ " in relation theory. Note that both X and Y can be an attribute or an attribute set.

Functional dependency is a central concept in relation theory. It's the foundation for defining various normalization levels. But before moving on to describing normalization in relation theory, we need to review and reconcile all the terminologies used to name the various parts of a database table.

Let's start with the bottom-line that a relation is a table and a table consists of rows and columns. In the following discussion, when two or more terms are given at the beginning of each bulleted item, the first term is in the context of a logical data model whereas the remaining part is the corresponding counterpart in the context of a physical data model. All part-counterpart pairs are listed as follows:

- *Relation or Table.* These two terms mean the same thing, namely, a table in a more explicit way. The convention is that the term *relation* is used to mean a *table* when discussing normalization of a logical data model whereas the term *table* is used to mean exactly the same thing—*relation*—in the context of a physical model.

- *Tuples or Rows or Records.* All these three terms mean the same thing, that is, a row of a table. The term *tuple* is used in the context of a logical data model, whereas the terms *row* and *record* are used in the context of a physical model.
- *Attributes or Columns.* Once again, the term *attribute* is used in the context of a logical data model, whereas the term *column* is used in the context of a physical model.

The other less explicit parts of a relation or table are named as follows:

- *Degree.* The number of attributes of a relation or table. Theoretically, the degree of a table could be from zero to infinity. Practically, however, it's limited to a range from one to a large N, since a table with zero attributes has no practical meaning and cannot be implemented in SQL; and it's equally true that a table cannot have an infinite number of attributes. Relations with a degree of one, two, three or more are called *unary*, *binary*, *ternary* or *n-nary* relations, respectively. For the example relation shown in Table 14.2, the degree is three. This concept of *degree* defines the horizontal dimension of a relation.
- *Cardinality.* The total number of rows of a table, which is 11 with the example relation shown in Table 14.2. This concept of *cardinality* defines the vertical dimension of a relation.
- *Domain.* The value set of an attribute. With the same example relation, the domain for the AccountType attribute is {Business, Checking, Savings}.
- *Content.* The whole value set of the attributes and tuples or rows of a relation or table. With this example relation, it has $3 \times 11 = 33$ data points as the content of the relation.

Finally, one needs to make a distinction between a relation and a relationship. The term *relation* implies the association among the attributes of a table, whereas the term *relationship* implies the association among the tables of a database.

Apparently, multiple, interchangeable terms coexist due to historical reasons. One needs to take a more accommodating attitude rather than get frustrated.

Table 14.2 An Example Table for Showing Properties of a Table

Customer	AccountType	Term
Jon	Business	ATM
Jon	Business	Fee
Odie	Checking	CreditLimit
Odie	Checking	ATM
Odie	Checking	Fee
Odie	Checking	MinBalance
Garfield	Checking	ATM
Garfield	Checking	MaxWithdraw
Garfield	Business	ATM
Garfield	Business	Overdraft
Garfield	Savings	WireTransfer

The convention in this text is that we use *relation* in describing normalizations and *table* elsewhere. We'll try to use *row* in place of *tuple* whenever possible.

Now we are ready to discuss normalizations, starting with the concepts of functional dependences and lossless-join decompositions. Note that we are using an approach much less formal than a pure mathematical one, so be flexible and exercise common sense whenever necessary.

14.5.3 Functional Dependencies and Lossless-Join Decompositions

A relation has many attributes and there are certainly various kinds of dependencies among those attributes. In fact, all kinds of anomalies and redundancies arise from the dependences of some attributes on some other attributes. In this section, we introduce the concept of a functional dependency, which is the most basic and important dependency that must be dealt with when performing normalizations in the logical database design phase.

The general definition of a functional dependency (FD) is given as follows.

Suppose we have a relation R , which has attributes of $a_1, a_2, \dots, a_n$, or $R = \{a_1, a_2, \dots, a_n\}$. Next, suppose we have two subsets of the attributes from relation R : X and Y , each of which could contain an arbitrary number of attributes in relation R . We also assume that X and Y may even contain overlapping attributes or that one can encompass the other entirely. For a given value set of X , if the value set of Y is uniquely determined, we say X determines Y or Y is functionally dependent on X . In relational algebra, it is expressed as $X \rightarrow Y$. Think of X as a key for the moment, and think of $X \rightarrow Y$ is equivalent to $Y = f(X)$, namely, Y is functionally dependent on X .

Using the previous ACTIVITY relation as an example, we see that for every ActivityID value, there is a corresponding CustomerID value. However, a same CustomerID value may correspond to many different ActivityID values. Thus, we say ActivityID determines CustomerID or $\text{ActivityID} \rightarrow \text{CustomerID}$, but not the other way around.

A theoretically interesting question is how one would arrive at all the FDs for a given relation. This question is termed computing the closure (all FDs) of a given relation. The following rules (first three are called *Armstrong's Axioms*) are helpful for deriving new FDs based on known FDs:

- *Reflexivity.* If $Y \subseteq X$, then $X \rightarrow Y$, which means if Y is part of X , then Y is functionally dependent on X . Actually, FDs satisfying this reflexivity property are called *trivial* FDs, which will be used to define certain level of normalizations as will be described later. Just keep this in mind for the moment.
- *Augmentation.* If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z , where Z is another subset of the columns of the table and XZ means a union of all columns from X and Z (same for YZ).
- *Transitivity.* If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$, which means if X determines Y and Y determines Z , then X determines Z .
- *Union.* If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.

- *Decomposition.* If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$. This rule will be used a lot in attaining higher and higher normalization levels later.

Next, let's consider what is a lossless decomposition: Simply put, a decomposition is *lossless* if the information contained in the original relation can be fully recovered with a natural join of all decomposed relations. A natural join is an equijoin that has all equalities from all common attributes of the two relations to be joined taken into account (an *equijoin* is defined in Chapter 17 of this book). The term *lossless* is interchangeable with the term *non-additive*, which means that when the decomposed relations are rejoined, it should not end up with more rows that the original relation did not have.

One must be careful that a natural join of the decomposed relations does not create extra information that was not in the original relation as we stated above. Figure 14.8 shows an example of a lossy decomposition, in which two extra rows are created when the two decomposed relations were naturally joined together.

As we emphasized, all decompositions intended to eliminate redundancies and thus anomalies must be lossless. There exists a theorem for testing if a decomposition is lossless, as introduced below:

Let F be a set of FDs of relation R . The decomposition of R into R_1 and R_2 is lossless if and only if $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$, where the symbol $\cap$ means intersection of the two sets.

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c3

(a)

A	B
a1	b1
a2	b2
a3	b1

B	C
b1	c1
b2	c2
b1	c3

(b) (c)

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c3
a1	b1	c3
a3	b1	c1

(d)

Figure 14.8 An example of a lossy decomposition (a) original table, (b) and (c) decomposed tables, and (d) re-joined table.

Based on the above theorem, this statement is true: *Given an FD $X \rightarrow Y$ for a relation R , if $X \cap Y = \emptyset$ (empty), then the decomposition of R into $R - Y$ and XY is lossless.* Note that $R - Y$ represents all attributes except those in subset Y and that XY represents the subset with all attributes from the subsets X and Y . Since X is in both decomposed relations, it's a key, with $X \subseteq Y$.

However, a lossless decomposition does not guarantee that the dependencies are preserved. And a decomposition that preserves dependencies may not be a lossless decomposition. Such issues should be governed by the actual application logic, and a more detailed discussion is beyond the scope of this text.

Now we are ready to discuss normalization. Let's start with the first normal form (1NF) next.

14.5.4 First Normal Form (1NF): Avoiding Multi-Valued Columns

A relation is said to be in 1NF if it contains zero multi-valued attributes. Take the preceding big relation **ACTIVITY** for example. For a given customer and a given activity ID, only one row (unique) can be in the database, but multiple transactions may occur. Therefore, the attributes highlighted in boldface as shown below may contain multiple value sets or repeated N times if the ActivityID given contains N transactions (if we use C to represent a customer attribute set and use T to represent a transaction attribute set, it would be in the form of ActivityID, $C, \{T_1, T_2, \dots, T_n\}$. T is also called a *repeating group*).

```
ACTIVITY: ActivityID (PK), CustomerID, CustomerName, CustomerAddress,
          CustomerCity, CustomerState, CustomerZipCode, CustomerPhone,
          TransactionDate, CheckNumber, TransactionType,
          TransactionDescription,
          Debit, Credit, AccountID, AccountName,
          AccountType, AccountDescription,
          AccountTerms, AccountStatus, AccountBalance
```

If this relation is not normalized properly, some queries such as search an activity based on the ActivityID and TransactionType may be problematic, since within the same row, there could be multiple transaction types and the query results cannot be determined deterministically. Applying the first level normalization by avoiding multi-valued attributes can cure such problems. To achieve this objective, the measure to take is to split the multi-valued attributes into a separate relation as follows:

```
ACTIVITY: ActivityID (PK), CustomerID, CustomerName, CustomerAddress,
          CustomerCity, CustomerState, CustomerZipCode, CustomerPhone,

TRANSACTION: TransactionID (PK), ActivityID (FK), AccountID,
             TransactionDate,
             CheckNumber, TransactionType, TransactionDescription,
             Debit, Credit, AccountName, AccountType,
```

```
AccountDescription, AccountTerms,
AccountStatus, AccountBalance
```

Note that we have made TransactionID a primary key in the new TRANSACTION relation, and the primary key of the original relation is copied over as a foreign key for referential integrity purposes. Next, let's proceed to the second normal form (2NF) through second level normalization.

14.5.5 Second Normal Form (2NF): Eliminating Partial Dependencies

Applying the previous FD definition to the TransactionID and AccountStatus attributes of the TRANSACTION relation, we can see that AccountStatus is functionally dependent on TransactionID, because for a given TransactionID value, there exists no more than one AccountStatus value that is associated with that TransactionID value. This is undesirable, as changing the status of the account of a customer would require updating all transactions of the customer. The 2NF breaks this kind of dependency, thus resulting in the following relations with our online banking example:

```
ACTIVITY:    ActivityID (PK), CustomerID, CustomerName, CustomerAddress,
             CustomerCity, CustomerState, CustomerZipCode, CustomerPhone
```

```
TRANSACTION: TransactionID (PK), AccountID (FK),
             TransactionDate, CheckNumber, TransactionType,
             TransactionDescription, Debit, Credit,
```

```
ACCOUNT:    AccountID (PK), ActivityID (FK), AccountName,
             AccountType, AccountDescription, AccountTerms,
             AccountStatus, AccountBalance
```

Looking at the ACTIVITY relation, we see that the customer entity is embedded there. So we are not done with normalization yet. Let's proceed to the third normal form (3NF) next, which eliminates transitive dependencies.

14.5.6 Third Normal Form (3NF): Eliminating Transitive Dependencies:

While the 2NF resolves the partial dependency problem, the 3NF resolves the transitive dependency problem. As described previously, we would have a transitive dependency if attribute Z depends on attribute Y ($Y \rightarrow Z$) and attribute Y depends on attribute X ($X \rightarrow Y$), thus attribute Z depends on attribute X ($X \rightarrow Z$). In such a scenario, typically X is a primary key, whereas Y and Z are not, but Y could be a potential primary key if the entity it represents is defined as a separate table. This is exactly the case with the ACTIVITY relation in 2NF as described in the previous section, namely, there is a transitive dependency of $\text{ActivityID} \rightarrow \text{CustomerID} \rightarrow \text{CustomerName}$. Therefore, the 3NF would be achieved if we move out the customer

entity into a separate relation while keeping the CustomerID as one of the non-key attribute of the ACTIVITY relation.

In summary, the 3NF requires that:

1. There is no multi-valued column in the relation (1NF)
2. There is no partial dependency of a non-key attribute on one of the key attribute (2NF)
3. There is no transitive dependency of a non-key attribute on another non-key attribute that depends on the primary key of the relation (3NF).

What is the 3NF really about? Simply put, it requires that the only functional dependencies allowed are those between the primary key (the whole key, not part of the key if the key is composed of multiple attributes) and the non-key attributes. In other words, for any functional dependency of $X \rightarrow Y$, X is a key and Y is a non-key attribute. Using this principle, the three tables that conform to the 2NF described previously can be split into the following four relations that conform to the 3NF:

ACTIVITY: ActivityID (PK), CustomerID (PK)

TRANSACTION: TransactionID (PK), AccountID (FK), TransactionDate,
CheckNumber, TransactionType, TransactionDescription,
Debit, Credit

ACCOUNT: AccountID (PK), CustomerID (FK), AccountName,
AccountType, AccountDescription, AccountTerms,
AccountStatus, AccountBalance, TransactionID (FK)

CUSTOMER: CustomerID (PK), CustomerName, CustomerAddress,
CustomerCity, CustomerState, CustomerZipCode, CustomerPhone

A legitimate question is if we need to normalize further beyond 3NF, or when we should stop splitting the relations for a database. The answer is that most database designs stop at the 3NF. But it's beneficial to know what higher level normal forms exist and if it's necessary to even perform a full normalization. In general, the following three more levels exist, depending on the situations with a specific database design: Boyce-Codd normal form (BCNF), fourth normal form (4NF), and fifth normal form (5NF). Each of these normal forms is briefly touched upon next.

14.5.7 Boyce-Codd Normal Form (BCNF): Eliminating Key—Non-Key Dependencies

This is considered a stronger version of 3NF. The issue it addresses occurs when a key attribute is functionally dependent on a non-key attribute. Let's use the above

ACCOUNT table in the 3NF to illustrate such a possibility. Suppose that we add an extra attribute PersonalBanker that an account type has only one personal banker assigned to (a personal banker helps customers in office face-to-face). Suppose further that we use the AccountName and AccountType attributes as the primary key without the surrogate AccountID key. Then we would have a situation that the key attribute AccountType would depend on the non-key attribute PersonalBanker and we would have many redundant combinations of the values from these two attributes.

BCNF breaks this kind of dependency by requiring setting up a separate relation with the AccountType and PersonalBanker attributes. On the other hand, if we use the surrogate key as is without introducing the key composed of the AccountName and AccountType attributes, then the dependency between the two non-key attributes of AccountType and PersonalBanker would break the 3NF due to the transitive dependency of AccountID $\rightarrow$ AccountType $\rightarrow$ PersonalBanker for the ACCOUNT relation. Either way, the solution is to set up a separate relation for the AccountType and PersonalBanker attribute with its other attributes included as well.

14.5.8 Fourth Normal Form (4NF): Trivializing or Keying Multi-Valued Dependencies

Up to BCNF, functional dependency is used to guide the normalization process for a table. Fourth normal form (4NF) is defined with the help of the concept of multi-valued dependency (MVD). To illustrate what a MVD is, let's use Table 14.3 as an example. With the three attributes of Customer, Biller, and PayMethod, the following observations hold true:

- The table has to use all three attributes to uniquely identify a row. That is to say, all attributes are key attributes or there is no non-key attribute. Therefore, this relation actually is in BCNF.
- For every Customer, multiple PayMethod values exist regardless of the Biller. For example, John can pay Credit One and Mortgage One either by check or by wire-transfer. That means that there is no functional dependency between any two of the three attributes because of the multiple-value property of an

Table 14.3 An Example Table for Illustrating Multi-Valued Dependencies (MVDs)

Customer	Biller	PayMethod
John	Credit One	Check
John	Credit One	Wire Transfer
John	Mortgage One	Check
John	Mortgage One	Wire Transfer
David	Credit One	Check
David	Credit One	By Phone
David	Credit One	Wire Transfer

attribute against the other two attributes. Note that the concept of an FD is based on the comparison between two single attributes or two attribute sets or a single attribute and an attribute set, whereas the concept of an MVD is extended to compare more than two attributes or attribute sets against each other.

- The underlying cause for the multi-valued dependency is that the data model represents a ternary relation. If we split this single ternary relation into two separate binary relations of Customer-Biller and Customer-PayMethod, then the redundancy and the multi-valued dependency would be eliminated. Then both relations are said to be in 4NF.

An MVD is a more complicated and confusing concept than an FD. Fortunately, Date and Fagin had devised a method for verifying 4NF using FDs without getting involved with MVDs. In order to introduce this method, we need to review the concepts of a simple key and a compound key, as introduced in Section 14.5.2 previously. A key is a *simple key* if it is only a single attribute of a relation; and a *compound key* consists of more than one attributes, each of which is a simple key in its own right. Then the Date-Fagin method states that a relation is in 4NF if:

- It is already in BCNF
- At least one key is a simple key.

The above method is based on the fact that every FD is an MVD but the reverse may not always be true. Because of this, the above conditions are sufficient but not necessary. This can be proved with the two decomposed relations shown side by side in Table 14.4 as follows.

To give a more precise definition of 4NF, we need to define further trivial MVD. Suppose we have two subsets of attributes: X and Y , each of which could be just a single attribute or an attribute set of a relation. The MVD $X \twoheadrightarrow Y$ (which reads *Y has a multi-valued dependency on X* or *X multi-determines Y*) is a *trivial* MVD if Y is part of X (or $Y \subseteq X$ in relational algebra notation) or a natural join of X and Y gives the full representation of the entire relation (or $XY = R$ where R represents the entire relation).

Table 14.4 Example Tables Demonstrating MVDs

Relation Customer-Biller		Relation Customer-PayMethod	
Customer	Biller	Customer	PayMethod
John	Credit one	John	Check
John	Mortgage One	John	Wire Transfer
David	Credit One	David	Check
		David	Wire Transfer
		David	By phone

Now we are ready to define 4NF using the concepts of a trivial MVD and a superkey. A relation R is in 4NF if either of the following conditions is true for every MVD $X \twoheadrightarrow Y$ in the relation:

- Every MVD $X \twoheadrightarrow Y$ is a trivial MVD. In other words, it does not contain non-trivial MVDs. This condition is equivalent to trivializing MVDs.
- X in every MVD $X \twoheadrightarrow Y$ is a superkey. This condition is equivalent to (super) keying MVDs.

It's obvious that the above two relations, Customer-Biller and Customer-PayMethod, obeys the above definition, and therefore by definition they are in 4NF.

Next, let's explore fifth normal form (5NF).

14.5.9 Fifth Normal Form (5NF): Trivializing or Keying Join Dependencies

Fifth normal form (5NF) is considered the “ultimate” level of normalization that no further decompositions are allowed or possible. It's also called projection-join normal form (PJFM), since at this level, only projection and join operations are allowed. By definition, a projection is a unary operation expressed as $\pi_{a_1, a_2, \dots, a_n}(R)$ where $a_1, a_2, \dots, a_n$ represent a set of attribute names. The result of a projection is the set that contains all rows with the values from those selected attributes only. Thus a projection is a result set with selected attributes. A projection is more than just an attribute set—it's an attribute set with all rows extracted from the relation with the attributes specified in the attribute set.

Fifth normal form is defined based on the concept of a join dependency (JD), which is a further generalization of MVDs. A join dependency is in turn defined based on the concept of a lossless-join decomposition of a relation R . Let's say we can decompose a relation R into a series of smaller relations $\{R_1, R_2, \dots, R_n\}$. Such a decomposition is said to be lossless if it does not result in the loss of information in the original relation, or in other words, the original relation can be recovered with a natural join of these decomposed relations. A JD is represented with a bowtie symbol in front of the decomposed relation set as $\bowtie\{R_1, R_2, \dots, R_n\}$. Also, to conclude our previous statement that a JD is a generalization of an MVD, the following MVD-JD conversion law is provided:

An MVD $X \twoheadrightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie\{XY, X(R - Y)\}$.

Now let's define 5NF. A relation R is in 5NF if and only if one of the following conditions is met for each of the relation R 's join dependencies $\bowtie\{R_1, R_2, \dots, R_n\}$:

- Every JD is a *trivial* JD, that is, $R_i = R$ for some i . This condition is equivalent to trivializing JDs.

Table 14.5 An Example Table for Demonstrating the Concept of a MVD

Relation R					
Customer		AccountType		Term	
Jon		Business		ATM	
Jon		Business		Fee	
Odie		Checking		CreditLimit	
Odie		Checking		ATM	
Odie		Checking		Fee	
Odie		Checking		MinBalance	
Garfield		Checking		ATM	
Garfield		Checking		MaxWithdraw	
Garfield		Business		ATM	
Garfield		Business		Overdraft	
Garfield		Savings		WireTransfer	
R1 = CT		R2 = CA		R3 = AT	
Customer	Term	Customer	AccountType	AccountType	Term
Jon	ATM	Jon	Business	Business	ATM
Jon	Fee	Odie	Checking	Business	Fee
Odie	CreditLimit	Garfield	Checking	Business	Overdraft
Odie	ATM	Garfield	Business	Checking	CreditLimit
Odie	Fee	Garfield	Savings	Checking	ATM
Odie	MinBalance			Checking	Fee
Garfield	MaxWithdraw			Checking	MinBalance
Garfield	ATM			Checking	MaxWithdraw
Garfield	Overdraft			Savings	WireTransfer
Garfield	WireTransfer				

- Every R_i in a $JD \bowtie \{R_1, R_2, \dots, R_n\}$ is a superkey of R (this condition is equivalent to super[keying] JDs). Note that some texts state this condition as “every JD is implied by the set of FDs, which are key dependencies (KDs).” A key dependency has its left-hand side as a candidate key with the implication that a candidate key is just one of the keys in a full key set of a relation (also be reminded that the preferred key is defined as the primary key). Although the two definitions are equivalent to each other, the first definition is more explicit and preferred.

Finally, there exists a procedure to verify if a relation is in 5NF. This procedure states that *if a relation is in 3NF and each of its candidate keys is a simple key, it is guaranteed in 5NF*. This is a sufficient but not a necessary condition, though. This is an extremely useful normalization law thanks to Date and Fagin. Because of its practical importance as we will elaborate further, let’s call it *Date-Fagin 5NF golden rule*.

To help illustrate the concept of 5NF, an example in the context of our sample online banking application is given in Table 14.5. It is seen that the relation R has three attributes: Customer, AccountType, and Term. A customer may have several types of account, for example, Business, Checking, and Savings.

Each account has a set of terms associated with it. For a specific account, what terms apply is determined by the bank. However, even for the same type of account, different terms may apply to different customers, depending on certain factors, for example, the credit score and minimal balance a customer is able to maintain, and so on.

Now the question is if the relation R is in 5NF. By quickly applying the Date-Fagin 5NF golden rule, we cannot determine it conclusively, because it's an all-key and no simple keys exist. Since it's an all-key relation, none of its projections could be a superkey of the relation. Therefore, we can conclude that the relation is not in 5NF.

However, $\bowtie\{R1 = CT, R2 = CA, R3 = AT\}$ is a JD for relation R . Note that in this JD notation, for convenience, the first letter of each attribute of the relation in upper case is used to denote the corresponding attribute. Therefore, we have C , A , and T for the attributes of Customer, AccountType, and Term, respectively.

Next, let's prove that the three decomposed relations, $R1$, $R2$, and $R3$, are in 5NF. Let's take the CT relation for example. In this case, the $JD \bowtie\{C, CT, T\}$ is a trivial JD for relation CT , and therefore, CT relation is in 5NF. The same proof applies to the other two relations of CA and AT as well, and thus, all decomposed relations are in 5NF.

The last question is which level of normalization a database design should settle down. The following section will try to answer this question.

14.5.10 Which Level of Normalization to Settle Down?

Contrary to the common belief that settling down to 3NF or BCNF would be sufficient, designers of real world databases should strive for 5NF—the “ultimate” level of normalization. By enforcing 5NF, the potential risks with inconsistent data are minimized, and fewer burdens are passed on to the application layer. However, it's a non-trivial effort to prove that a database is indeed in 5NF if it's not designed to be in 5NF consciously. The dilemma can be circumvented—thanks to the Date-Fagin golden rule, namely, if a relation is in 3NF and every key of the relation is a simple key, then the relation is guaranteed to be in 5NF. Accordingly, to be or not to be in 5NF is merely a choice of a designer. There is a simple way to remember the Date-Fagin 5NF golden rule: *in a 3NF relation, make every key a simple key, and then the relation would be a 5NF relation.*

Finally, be warned that 5NF doesn't guarantee redundancy-free. This can be seen clearly with the preceding example: both attributes of each of the three 5NF tables have redundant values. Whether redundancy should be a concern or not should be determined based on the consequences it may cause. Redundancy could lead to operational (INSERT, UPDATE, and DELETE) anomalies as well as redundant storage of data. It's really the anomalies caused by redundancy that we are mainly concerned with.

14.5.11 Denormalization?

You might have heard that denormalization can help improve performance. First, note that denormalization is different from having not performed normalization at

all in the first place. These are two separate concepts. Having not performed normalization in the first place means that potential anomalies exist and the designer may not be aware of them. Denormalization means the designer knows what is being traded for. So should denormalization be a recommended practice or not?

It depends on the data to be denormalized. If data is not expected to change, for example, purely historical data in OLAP or data warehouse scenarios, then denormalization might be a viable solution to a performance issue that cannot be resolved otherwise. However, if data is expected to change and change frequently, the denormalization is not recommended at all, because what's the point of having a *fast* database with data consistency compromised? If you run into such a situation, other solutions such as database/application optimization and tuning as well as using more advanced hardware should be sought after, or consider using trustworthy external consulting services.

From now on, we will focus on the physical design of a database, built on what we have learned about the conceptual and logical designs in the previous two sections. Also, we'll switch to the terms of *table* and *column* in place of *relation* and *attribute* after we transit from logical to physical design.

14.6 PHYSICAL DESIGN

While conceptual and logical designs are independent of specific database platforms, a physical design of a database gets into the gory details of a database management system. This is because the physical design phase is a process of mapping a logical design into corresponding tables that can be implemented on a specific database management system. In this section, we will illustrate how the logical design of our sample online banking application can be implemented with Oracle. The following steps of converting a logical design into a physical design will be discussed:

1. Deciding on naming conventions
2. Creating the designated user data tablespace and application temporary tablespace
3. Creating the schema user for the application
4. Creating application schema objects
5. Changing schema objects
6. Enforcing business rules and data integrity
7. Adding views
8. Creating sequences and synonyms
9. Adding indexes
10. Security

First, to follow along, you need to install Oracle server software on a platform of your choice and then create an Oracle instance. In my case, I installed Oracle 11g Release 2 on a system with the following specs:

- CPUs (4): AMD Phenom II X4 quad-core 830 @2.8 GHz (2 MB L2 + 4 MB Cache, 4 GHz System Bus)
- RAM size: 6 GB DDR3 SDRAM (3 × 2 GB)
- Disks (1 TB): SATA (7200 RPM, 8 MB Cache)
- OS (32-bit): Windows 7 Home Premium 64-bit

The instance and database configuration details are given below for your reference in case you are interested. This information was extracted from the HTML file saved prior to initiating database creation at the last step. You can save your configuration details in this manner for bookkeeping purpose as well.

Database Configuration Summary

Global Database Name:	ora11gr2
Database ConfigurationType:	Single Instance
SID:	ora11gr2
Management Option Type:	Database Control
Storage Type:	File System
Memory Configuration Type:	Automatic Memory Management

Database Configuration Details

Database Components

Component	Selected
Oracle JVM	true
Oracle Text	true
Oracle XML DB	true
Oracle Multimedia	true
Oracle OLAP	true
Oracle Spatial	true
Oracle Label Security	false
Sample Schemas	true
Enterprise Manager Repository	true
Oracle Application Express	true
Oracle Warehouse Builder	true
Oracle Database Vault	false
Oracle Database Extensions for .NET	false

Initialization Parameters

Name	Value
audit_trail	db
compatible	11.2.0.0.0
db_block_size	8KB
db_name	ora11gr2
db_recovery_file_dest_size	3852MB
dispatchers	(PROTOCOL=TCP) (SERVICE={SID}XDB)
memory_target	1228MB
open_cursors	300
processes	150
remote_login_passwordfile	EXCLUSIVE
undo_tablespace	UNDOTBS1

Character Sets

Name	Value
Database Character Set	AL32UTF8
National Character Set	AL16UTF16

From the database components shown above, you can see what Oracle features were included. From the initialization parameter table, the memory target was set to 1228 MB, which indicates that the auto memory management feature of Oracle 11g was used (refer to Chapter 7 on Oracle 11g memory management). In addition, the character sets table indicates that the UNICODE option was used.

Next, let's clarify naming conventions.

14.6.1 Naming Conventions

Different organizations may have different naming conventions for the various parts of a database. Although it may not matter much which set of naming conventions is used, it does help minimize miscommunications if a common set of naming conventions can be followed in an organization.

In this text, the following naming conventions are used:

- *Table Name.* A table name is all in capital letters with the singular form. Whenever a concatenation is needed, an underscore “_” is used. For example, ACCOUNT, CHECKING_ACCOUNT, and SAVINGS_ACCOUNT, and so on.
- *Column Name.* The same convention for naming a table name is followed with a column name as well. Besides, the name of a column is not prefixed with the table name, since the semantics of a column name is local to the table it belongs to,

except that when we refer to the ID column of a table. For example, we use CUSTOMER_ID to refer to the ID column of the CUSTOMER table, and NAME instead of CUSTOMER_NAME to refer to the name column of the same customer table. Note that the same upper case, singular form is used for the name of a column of a table.

- *Constraints.* A constraint is named in the format of <TABLE_NAME>_<CONSTRAINT_ABBREVIATION>_<COLUMN>, where each entity represented as <...> should be replaced with a proper item. The constraint abbreviations of PK, FK, and CK are used to denote a primary key (PK), a foreign key (FK), and a check constraint (CK), respectively. For example, CUSTOMER_PK_CUSTOMER_ID is the name of the primary key constraint of the CUSTOMER table, predicated on the CUSTOMER_ID column.

Other naming conventions will be made clear as we move along throughout the remainder of this chapter. The next section describes how to create the application data tablespace and temporary tablespace for SOBA. This step comes before creating a schema user since a user needs to be associated with such tablespaces when being created.

14.6.2 Creating Tablespaces

A database-centric application needs to have one or more tablespaces created to contain application data. In addition, it needs to have a temporary tablespace created as well. For SOBA, the tablespace is named *OnlineBanking*, and the temporary tablespace is named *OBTemp*. To create these tablespaces, log into SQL*Plus using the sys account as sysdba like the following:

```
sqlplus sys/<password>@<connect_string> as sysdba
```

Then type @create_soba_tablespaces.sql at the SQL> prompt. This script contains the following SQL statements (note that you need to change the file path to suit your needs):

```
CREATE TABLESPACE OnlineBanking DATAFILE 'C:\mspc\dev
\oradata\orallgr2\olbnk01.dbf'
SIZE 50M EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO;

CREATE TEMPORARY TABLESPACE OBTemp TEMPFILE
'C:\mspc\dev\oradata\orallgr2\obtemp01.dbf' SIZE 20M REUSE
EXTENT MANAGEMENT
LOCAL UNIFORM SIZE 2M;
/
```

The first DDL SQL above creates the tablespace *OnlineBanking* with the data file (olbnk01.dbf), size (50 MB), extent management (local), and segment space man-

agement (auto) as specified. The second DDL SQL above creates the temporary tablespace `OBTemp` with data file (`obtemp01.dbf`), size (20 MB), extent management (local), and uniform size 2M as specified. Note that the `REUSE` option is specified as well. The best performance and scalability practices of setting local extent management and auto segment space management are followed here.

This step of creating SOBA tablespaces is straightforward. Let's move to the next section, which describes how to create a schema user with proper privileges for the sample online banking application.

14.6.3 Creating a Schema User with Proper Privileges

To create the schema user, continue from the preceding `SQL>` prompt or log into `SQL*Plus` using the `sys` account as `sysdba` like `sqlplus sys/<password>@<connect_string>` as `sysdba`, and then type `@create_soba_schema_user.sql` at the `SQL>` prompt. This script contains the following SQL statements:

```
CREATE USER OBAdmin identified by OB#Admin DEFAULT TABLESPACE OnlineBanking
TEMPORARY TABLESPACE OBTemp;
GRANT CONNECT, RESOURCE to OBAdmin;
GRANT SELECT ANY DICTIONARY to OBAdmin;
GRANT CREATE VIEW TO OBADMIN;
GRANT CREATE SYNONYM TO OBADMIN;
/
```

From this point on, use the `OBADMIN` account to log into `SQL*Plus` by executing the command of `sqlplus OBAdmin/OB#Admin@<connect_string>`. Note that if you use a different account other than the `OBADMIN` account, the object will be stored in the other account.

The next section describes how to create other schema objects for this sample online banking application, such as tables, constraints, triggers, sequences, functions, views, indexes, and so on.

14.6.4 Creating Application Schema Objects

In this section, we describe how to create all necessary schema objects for SOBA. Let's create all objects first, and then explain in the next few sections about the objects created here. For easier reference, here is an outline of the SOBA schema objects to be created (note that all object names are mnemonic, so you probably already know what each object is about except a few such as `AUTHORITIES`, `ACL`, and so on, which will be explained later):

- `CUSTOMER` Table
- `ACCOUNT` Table
- `TRANSACTION` Table
- `LOGINUSER` Table

- TRANSFER Table
- BILLPAYMENT Table
- STATEMENT Table
- ACTIVITY View
- ACL Table
- AUTHORITIES Table
- USERS Synonym
- ACCOUNT BALANCE UPDATE Trigger
- USER AUTHORITIES Trigger
- COMPUTE BALANCE Function
- RANDOM STRING Function
- All Sequences
- TX_ACCOUNT_ID_TRANS_DATE Index

Let's start with creating the CUSTOMER table.

1. Create the CUSTOMER Table Execute @create_customer.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE TABLE CUSTOMER
(CUSTOMER_ID      VARCHAR2 (9) NOT NULL,
FIRST_NAME        VARCHAR2 (25)   NOT NULL,
LAST_NAME         VARCHAR2 (25)   NOT NULL,
PHONE             VARCHAR2 (12)    ,
ADDRESS           VARCHAR2 (50)    NOT NULL,
CITY              VARCHAR2 (25)    NOT NULL,
STATE            VARCHAR2 (2) NOT NULL,
ZIPCODE           VARCHAR2 (10)    ,
EMAIL             VARCHAR2 (50)    ,
STATUS            NUMBER (1)  NOT NULL,
CREATE_DATE       TIMESTAMP  NOT NULL) ;

ALTER TABLE CUSTOMER
  ADD CONSTRAINT CUSTOMER_PK_CUSTOMER_ID
    PRIMARY KEY (CUSTOMER_ID) ;

/
```

2. Create the ACCOUNT Table Execute @create_account.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE TABLE ACCOUNT
(ACCOUNT_ID      VARCHAR2 (9) NOT NULL,
```

```

NAME          VARCHAR2 (25)  NOT NULL,
TYPE          VARCHAR2 (10)  NOT NULL,
DESCRIPTION    VARCHAR2 (500)  NOT NULL,
STATUS        VARCHAR2 (2) NOT NULL,
BALANCE       NUMBER (10,2) NOT NULL,
OPEN_DATE     TIMESTAMP  NOT NULL,
CLOSE_DATE    TIMESTAMP,
CUSTOMER_ID   VARCHAR2 (9) NOT NULL);

ALTER TABLE ACCOUNT
  ADD CONSTRAINT ACCOUNT_PK_ACCOUNT_ID
    PRIMARY KEY (ACCOUNT_ID);

ALTER TABLE ACCOUNT
  ADD CONSTRAINT ACCOUNT_FK_CUSTOMER_ID
    FOREIGN KEY (CUSTOMER_ID)
      REFERENCES CUSTOMER (CUSTOMER_ID);

ALTER TABLE ACCOUNT
  ADD CONSTRAINT ACCOUNT_CK_BALANCE
    CHECK (BALANCE >= 0);

ALTER TABLE ACCOUNT
  ADD CONSTRAINT ACCOUNT_UK_CUSTOMER_ID_TYPE UNIQUE (CUSTOMER_ID,
TYPE);
/

```

3. Create the TRANSACTION Table Execute @create_transaction.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE TABLE TRANSACTION
(TRANSACTION_ID NUMBER(10)  NOT NULL,
TRANS_DATE      TIMESTAMP  NOT NULL,
TYPE           VARCHAR2 (10)  NOT NULL,
INITIATOR      VARCHAR2 (50)  NOT NULL,
DESCRIPTION    VARCHAR2 (500)  NOT NULL,
AMOUNT        NUMBER (10,2) NOT NULL,
BALANCE       NUMBER (10,2) NOT NULL,
ACCOUNT_ID    VARCHAR2 (9) NOT NULL,
STATUS        VARCHAR2 (9) NOT NULL);

ALTER TABLE TRANSACTION
  ADD CONSTRAINT TX_PK_TRANSACTION_ID
    PRIMARY KEY (TRANSACTION_ID);

ALTER TABLE TRANSACTION
  ADD CONSTRAINT TX_FK_ACCOUNT_ID

```

```

FOREIGN KEY (ACCOUNT_ID)
REFERENCES ACCOUNT (ACCOUNT_ID) ;
/

```

4. Create the LOGINUSER Table Execute @create_loginuser.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE TABLE LOGINUSER
(USERNAME      VARCHAR2 (9) NOT NULL,
PASSWORD      VARCHAR2 (8) NOT NULL,
ENABLED       NUMBER (1)  NOT NULL,
CREATE_DATE   TIMESTAMP   NOT NULL,
CLOSE_DATE    TIMESTAMP   ,
CUSTOMER_ID   VARCHAR2 (9) NOT NULL) ;

ALTER TABLE LOGINUSER
ADD CONSTRAINT LOGIN_PK_USERNAME
PRIMARY KEY (USERNAME) ;

ALTER TABLE LOGINUSER
ADD CONSTRAINT LOGIN_FK_CUSTOMER_ID
FOREIGN KEY (CUSTOMER_ID)
REFERENCES CUSTOMER (CUSTOMER_ID) ;
/

```

5. Create the TRANSFER Table Execute @create_transfer.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE TABLE TRANSFER
(TRANSFER_ID   NUMBER (10)  NOT NULL,
TRANSFER_DATE  TIMESTAMP   NOT NULL,
FROM_ACCOUNT_ID VARCHAR2 (10) NOT NULL,
TO_ACCOUNT_ID  VARCHAR2 (10) NOT NULL,
FROM_TX_ID     VARCHAR2 (10) NOT NULL,
TO_TX_ID       VARCHAR2 (10) NOT NULL,
INITIATOR      VARCHAR2 (10) NOT NULL,
DESCRIPTION    VARCHAR2 (500) NOT NULL,
AMOUNT         NUMBER (10,2) NOT NULL) ;

ALTER TABLE TRANSFER
ADD CONSTRAINT TSF_PK_TRANSFER_ID
PRIMARY KEY (TRANSFER_ID) ;

```

```

ALTER TABLE TRANSFER
  ADD CONSTRAINT TSFX_FK_ACCOUNT_ID
    FOREIGN KEY (FROM_ACCOUNT_ID)
      REFERENCES ACCOUNT (ACCOUNT_ID) ;
/

```

6. Create the *BILLPAYMENT* Table Execute @create_billpayment.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE TABLE BILL_PAYMENT
(ID          NUMBER (10) NOT NULL,
ACCOUNT_ID   VARCHAR2 (9) NOT NULL,
DESCRIPTION  VARCHAR2 (500) NOT NULL,
AMOUNT       NUMBER (10,2) NOT NULL,
FROM_ACCOUNT VARCHAR2 (25) NOT NULL,
BILLER       VARCHAR2 (25) NOT NULL,
ADDRESS      VARCHAR2 (50) NOT NULL,
CITY         VARCHAR2 (25) NOT NULL,
STATE        VARCHAR2 (2) NOT NULL,
ZIPCODE      VARCHAR2 (10) NOT NULL,
STATUS       VARCHAR2 (25) NOT NULL,
SCHEDULE_DATE TIMESTAMP,
SEND_DATE    TIMESTAMP) ;

ALTER TABLE BILL_PAYMENT
  ADD CONSTRAINT BLL_PYMNT_PK_ID
    PRIMARY KEY (ID) ;

CREATE SEQUENCE BLL_PYMNT_SEQ
START WITH 1
MAXVALUE 999999999
MINVALUE 1
NOCYCLE
CACHE 20
NOORDER;
/

```

7. Create the *STATEMENT* Table Execute @create_statement.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE TABLE STATEMENT
(STATEMENT_ID VARCHAR2 (8) NOT NULL,
ACCOUNT_ID   VARCHAR2 (8) NOT NULL,

```

```

START_DATE      DATE                NOT NULL,
END_DATE        DATE                NOT NULL,
SCAN_TIME       TIMESTAMP           NOT NULL,
TYPE            VARCHAR2 (10)       NOT NULL,
INITIATOR       VARCHAR2 (10)       NOT NULL,
DESCRIPTION     VARCHAR2 (500)      NOT NULL,
REPORT          CLOB                );

```

```

ALTER TABLE STATEMENT
  ADD CONSTRAINT STATEMENT_PK_STATEMENT_ID
    PRIMARY KEY (STATEMENT_ID);

```

```

ALTER TABLE STATEMENT
  ADD CONSTRAINT STATEMENT_FK__ACCOUNT_ID
    FOREIGN KEY (ACCOUNT_ID)
      REFERENCES ACCOUNT (ACCOUNT_ID);

```

```
/
```

8. Create the ACTIVITY View Execute @create_activity_view.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE VIEW ACTIVITY AS
  SELECT a.CUSTOMER_ID, a.ACCOUNT_ID, a.NAME, a.TYPE AS ACCOUNT_TYPE,
         t.TRANSACTION_ID, t.TRAN_DATE, t.TYPE AS TX_TYPE, t.INITIATOR,
         t.DESCRPTION, t.AMOUNT, t.BALANCE, t.STATUS
  FROM ACCOUNT a, TRANSACTION t
 WHERE a.ACCOUNT_ID = t.ACCOUNT_ID ORDER BY a.ACCOUNT_ID;
<...>
/

```

Note this Activity view has deviated from the ACTIVITY table we started with in the previous section in discussing normalizations. This kind of change is typical as a design process is evolved and refined.

9. Create the ACL Table Execute @create_acl.sql at the SQL> prompt. This script contains the following SQL statements:

```

CREATE SEQUENCE acl_sid_id_seq START WITH 2000;
CREATE SEQUENCE acl_class_id_seq START WITH 2000;
CREATE SEQUENCE aoi_id_seq START WITH 2000;
CREATE SEQUENCE ae_id_seq START WITH 2000;

CREATE TABLE ACL_SID (
  ID          NUMBER (20)          NOT NULL,
  SID         VARCHAR2 (120)       NOT NULL,
  PRINCIPAL   NUMBER (8)          NOT NULL
);

```

```

/
ALTER TABLE ACL_SID
  ADD CONSTRAINT AS_PK PRIMARY KEY (ID);
/
ALTER TABLE ACL_SID
  ADD CONSTRAINT AS_UK UNIQUE (SID, PRINCIPAL);
/
CREATE OR REPLACE TRIGGER acl_sid_pk_trg
BEFORE INSERT ON ACL_SID
FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL
  THEN
    SELECT acl_sid_id_seq.NEXTVAL INTO :NEW.id FROM DUAL;
  END IF;
END;
/
CREATE TABLE ACL_CLASS (
  ID NUMBER (20) NOT NULL,
  CLASS VARCHAR2 (120) NOT NULL,
  CONSTRAINT AC_PK PRIMARY KEY (ID),
  CONSTRAINT AC_UK UNIQUE (CLASS)
);
/

CREATE OR REPLACE TRIGGER acl_class_pk_trg
BEFORE INSERT ON ACL_CLASS
FOR EACH ROW
BEGIN
  IF :NEW.id IS NULL
  THEN
    SELECT acl_class_id_seq.NEXTVAL INTO :NEW.id FROM DUAL;
  END IF;
END;
/
CREATE TABLE ACL_OBJECT_IDENTITY (
  ID NUMBER (20) NOT NULL,
  OBJECT_ID_CLASS NUMBER (20) NOT NULL,
  OBJECT_ID_IDENTITY NUMBER (20) NOT NULL,
  PARENT_OBJECT NUMBER (20),
  OWNER_SID NUMBER (20),
  ENTRIES_INHERITING NUMBER (10) NOT NULL,
  CONSTRAINT AOI_PK PRIMARY KEY (ID),
  CONSTRAINT AOI_UK UNIQUE (OBJECT_ID_CLASS, OBJECT_ID_IDENTITY)
);
/
ALTER TABLE ACL_OBJECT_IDENTITY
  ADD CONSTRAINT AOI_FK_AOI_PO
    FOREIGN KEY (PARENT_OBJECT)
    REFERENCES ACL_OBJECT_IDENTITY (ID);
/
ALTER TABLE ACL_OBJECT_IDENTITY
  ADD CONSTRAINT AOI_FK_AC_PO

```



```

        FOREIGN KEY (OBJECT_ID_CLASS)
        REFERENCES ACL_CLASS (ID);
    /
ALTER TABLE ACL_OBJECT_IDENTITY
    ADD CONSTRAINT AOI_FK_AS_SID
        FOREIGN KEY (OWNER_SID)
        REFERENCES ACL_SID (ID);
    /
CREATE OR REPLACE TRIGGER aoi_pk_trg
BEFORE INSERT ON ACL_OBJECT_IDENTITY
FOR EACH ROW
BEGIN
    IF :NEW.id IS NULL
    THEN
        SELECT aoi_id_seq.NEXTVAL INTO :NEW.id FROM DUAL;
    END IF;
END;
    /
CREATE TABLE ACL_ENTRY (
    ID                NUMBER (20) NOT NULL,
    ACL_OBJECT_IDENTITY NUMBER (20) NOT NULL,
    ACE_ORDER         NUMBER (10) NOT NULL,
    SID               NUMBER (20) NOT NULL,
    MASK              NUMBER (10) NOT NULL,
    GRANTING           NUMBER (1)  NOT NULL,
    AUDIT_SUCCESS      NUMBER (1)  NOT NULL,
    AUDIT_FAILURE      NUMBER (1)  NOT NULL,
    CONSTRAINT AE_PK PRIMARY KEY (ID),
    CONSTRAINT AE_UK UNIQUE ( ACL_OBJECT_IDENTITY, ACE_ORDER)
);
    /

ALTER TABLE ACL_ENTRY
    ADD CONSTRAINT AE_FK_AOI_OII
        FOREIGN KEY (ACL_OBJECT_IDENTITY)
        REFERENCES ACL_OBJECT_IDENTITY (ID);
    /
ALTER TABLE ACL_ENTRY
    ADD CONSTRAINT AE_FK_AS_SID
        FOREIGN KEY (SID)
        REFERENCES ACL_SID (ID);
    /
CREATE OR REPLACE TRIGGER ae_pk_trg
BEFORE INSERT ON ACL_ENTRY
FOR EACH ROW
BEGIN
    IF :NEW.id IS NULL
    THEN
        SELECT ae_id_seq.NEXTVAL INTO :NEW.id FROM DUAL;
    END IF;
END;
    /

```

10. Create the AUTHORITIES Table Execute @create_authorities.sql at the SQL> prompt. This script contains the following SQL statement:

```
CREATE TABLE AUTHORITIES (
    USERNAME VARCHAR2 (10)    NOT NULL,
    AUTHORITY VARCHAR2 (10)    NOT NULL,
    CONSTRAINT AUTH_FK_USERNAME FOREIGN KEY (USERNAME) REFERENCES
    USERS (USERNAME)
);
/
```

11. Create the USERS Synonym Execute @create_synonym_users.sql at the SQL> prompt. This script contains the following SQL statement:

```
CREATE SYNONYM USERS FOR LOGINUSER;
/
```

12. Create the ACCOUNT_BALANCE_UPDATE Trigger Execute @create_trigger_account_balance_update.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE OR REPLACE TRIGGER account_balance_update
    BEFORE INSERT on TRANSACTION

    FOR EACH ROW
    DECLARE account_new_balance NUMBER (10,2);
    BEGIN
        SELECT BALANCE INTO account_new_balance FROM ACCOUNT WHERE
        ACCOUNT.ACCOUNT_ID = :new.ACCOUNT_ID;
        account_new_balance := account_new_balance + :new.AMOUNT;
        UPDATE ACCOUNT SET (BALANCE) = account_new_balance
        WHERE ACCOUNT.ACCOUNT_ID = :new.ACCOUNT_ID;
        :new.balance := account_new_balance;
    END;
/
```

13. Create the USER_AUTHORITIES Trigger Execute @create_trigger_user_auth.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE OR REPLACE TRIGGER USER_AUTH
    AFTER INSERT on USERS
    REFERENCING NEW AS n OLD AS o
    FOR EACH ROW

    BEGIN

        INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES (:n.username,
        'ROLE_CUST');
    END;
/
```

14. Create the *COMPUTE_BALANCE* Function Execute @create_compute_balance.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE OR REPLACE FUNCTION compute_balance (acct_id NUMBER, amount
NUMBER)
RETURN NUMBER IS
curr_balance NUMBER;
BEGIN
SELECT balance into curr_balance FROM ACCOUNT
WHERE account_id = acct_id;
RETURN (amount + curr_balance);
END;
/
```

15. Create the *RANDOM_STRING* Function Execute @create_random_string.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE OR REPLACE FUNCTION random_string (type IN VARCHAR2, length IN
NUMBER)
RETURN VARCHAR2 IS
rdm_string VARCHAR2 (500);
BEGIN
SELECT DBMS_RANDOM.STRING (type, length) INTO rdm_string FROM DUAL;
RETURN rdm_string;
END;
/
```

16. Create all Sequences Execute @create_seq.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE SEQUENCE customer_id_seq MINVALUE 1 MAXVALUE 99999999;
CREATE SEQUENCE account_id_seq MINVALUE 1 MAXVALUE 99999999;
CREATE SEQUENCE transaction_id_seq MINVALUE 1 MAXVALUE 99999999;
/
```

17. Create the *TX_ACCOUNT_ID_TRANS_DATE* Index Execute @create_index_tx_account_id_trans_date_ix.sql at the SQL> prompt. This script contains the following SQL statements:

```
CREATE INDEX TX_ACCOUNT_ID_TX_DATE_IX on TRANSACTION (ACCOUNT_ID,
TRANS_DATE);
/
```

14.6.5 Changing Schema Objects

After creating a schema object from scratch, you may want to make some changes to the object created. Some most common changes are:

- *Renaming a Table.* One can use the following SQL to rename a table:

```
RENAME <old_table_name> TO <new_table_name>;
```

- *Adding Columns.* Additional columns can be added to an existing table as follows:

```
ALTER TABLE <table_name> ADD <new_column_name><datatype> [NOT NULL];
```

- *Changing Column Data Types.* The data type of a column can be changed as follows:

```
ALTER TABLE <table_name> MODIFY <column_name> <new_datatype>;
```

- *Changing NULL Options.* The NULL option of a column can be changed from NULL to NOT NULL (or vice versa) as follows:

```
ALTER TABLE <table_name> MODIFY <column_name> NOT NULL;
```

However, when you make such changes in a production environment, it's necessary to test it out first in a staging environment. Also, carefully consider the consequences of such changes on other parts of the application, for example, changing the name of a table may require corresponding application coding changes as well.

Figure 14.9 shows the schema diagram created with Visio for this sample online banking application. The next section discusses how business rules and data integrity are enforced with various types of constraints as created in this section.

14.6.6 Enforcing Business Rules and Data Integrity

Business rules must be enforced rigorously with an enterprise application or a customer-oriented application such as an online banking application. However, business rules can be enforced either at the database layer or application layer or even at both layers. How business rules should be enforced requires application developers and database developers to work together so that they will neither overlap nor miss at both levels. However, if a product is required to support multiple database system platforms, then it's better to enforce the relevant business rules at the application layer rather than at the database layer. Even for a product that runs exclusively on a chosen database system platform, it might be beneficial from performance and scalability perspectives to implement business rules at the application layer rather than at the database layer so that garbage data will not be flushed down to the database layer to waste resources. Although there are no hard rules, it's important to make sure that the same business rule would not be implemented twice unless absolutely necessary.

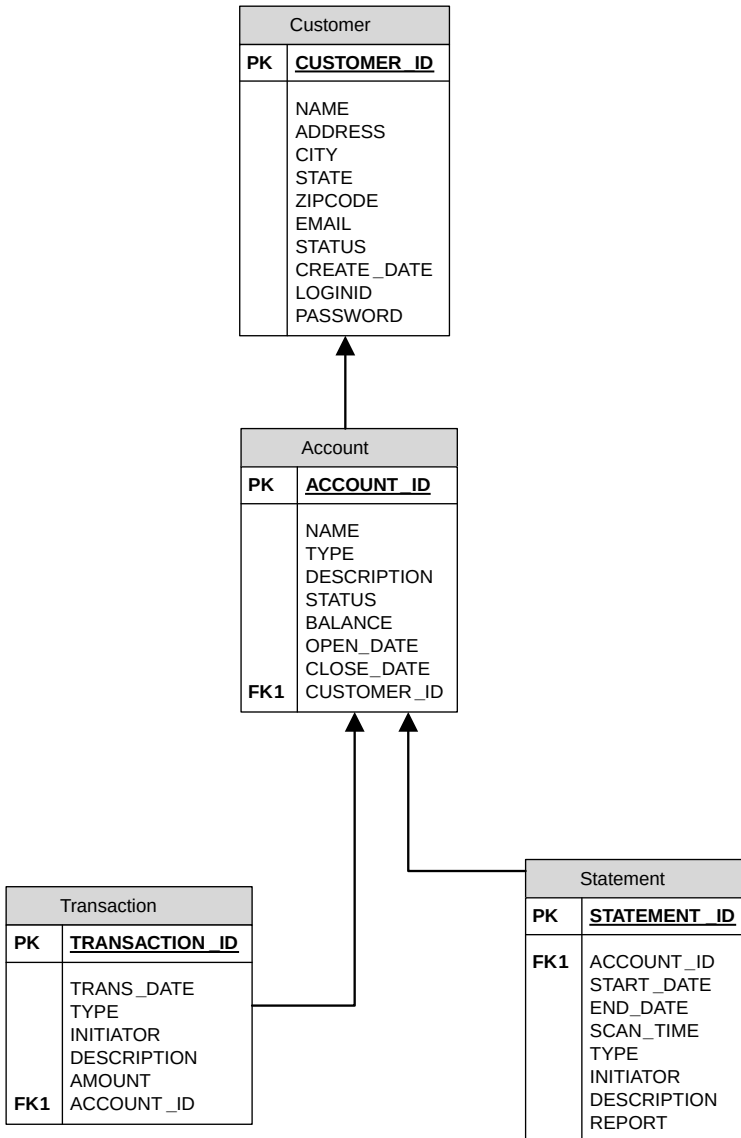


Figure 14.9 Schema diagram for the sample online banking application.

In this section, we'll focus on the database layer only—mainly on how one can use Oracle built-in constraints to enforce business rules and data integrity. The following types of constraints are discussed:

- **NOT NULL Constraints.** Refer to the CREATE/ALTER TABLE SQLs in Section 14.6.4 and you would find many such constraints. Some examples include: a customer cannot be created unless he has a valid physical address; an account

cannot be created unless it has a valid account type, and so on. Essentially, every required column must be created with the NOT NULL constraint.

- *Primary Key Constraints.* Primary key constraints guarantee that every row in a table can be uniquely identified with a primary key value. You can update the non-PK columns but you cannot update or modify the PK-columns, and you cannot insert a new row with a PK value that already exists in the table.
- *Foreign Key (Referential) Constraints.* Foreign key constraints are typical with one-to-many relationships, which can also be described as parent-child relationships. By creating FK constraints in the child table, it's ensured that whenever a new entry is inserted in the child table, it has a valid foreign key value or a valid primary key value in the parent table. On the other hand, it's also ensured that a table cannot be deleted if it's referenced by one or more child tables. Take the CUSTOMER-ACCOUNT relationship for example. A customer can have multiple accounts, so it's necessary to put a FK constraint on the ACCOUNT table with the PK CUSTOMER_ID from the CUSTOMER table. This would guarantee that transactions for an account would always been associated with a valid customer, or a customer could not be deleted if it has an account that is active.
- *Check Constraints.* Check constraints are used to limit the values that a column can take. For example, with the ACCOUNT table, a check constraint is added to enforce that the balance of the account cannot be negative. This would prevent some bandits from withdrawing millions of dollars and then run away and disappear forever.
- *Unique Constraints.* Since a table can have only one PK, unique constraints help identify additional column sets that must be unique. For example, we have a unique constraint on the STATEMENT table that a combination of CUSTOMER_ID, ACCOUNT_ID, START_DATE, and END_DATE must be unique so that the statement for the same period would not be created more than once. Note that ORACLE creates indexes both on PK and on unique constraints automatically. Besides, like the PK constraint, the unique constraint does not allow two rows with the same unique constraint value to be created. However, a unique constraint differs from a PK constraint that some of its columns can take NULL values.
- *Triggers.* A trigger is a piece of code written in proprietary language of a database system that will be fired up or executed when certain event occurs or certain conditions are satisfied. With Oracle, triggers can be written in PL/SQL or Java. Refer to Section 14.6.4 for the two triggers created: One is an account balance update trigger, while the other is a user authorities trigger. What these two triggers do exactly is left as an exercise as listed at the end of this chapter.
- *Functions.* An Oracle function is similar to a trigger except that it can be called in other SQL statements. Refer to Section 14.6.4 for the two functions created for SOBA.

In the next section, we discuss how adding views is a common practice in the physical design phase of a database.

14.6.7 Adding Views

A view is not a physical table, but it gives a user an illusion that it acted like a table and the data actually came from a table. That's the essence of a view: it allows the data within the same table to be viewed multiple ways without having to create multiple tables with the same data in the same table. Thus, the concept of a view is simple: Define a query that is known to be used frequently by users, store it in the database and users can simply invoke it by name just as they would with a table. In a word, a view can help hide the complexities of a physical table and give a user only what he needs. From the performance and scalability perspectives, views are a much more efficient way than letting a developer select all the columns of a table and then pick what he needs.

The syntax for creating a view is as simple as follows (note that the schema user requires the system privilege of `CREATE ANY VIEW`):

```
CREATE OR REPLACE VIEW view_name AS
SELECT statement
;
```

The `ACTIVITY` view described in Section 14.6.4 illustrates how an account activity view can be created with the `ACCOUNT` and `TRANSACTION` tables. The `SELECT SQL` shows how a user can query the view with a given customer. Now the question is how this query will be executed. It would be too inefficient from performance and scalability perspectives if first all data for all customers are retrieved and then the data for the specified customer is selected. Oracle optimizer handles it efficiently by rewriting the SQL with the underlying view taken into account.

In the next section, we'll discuss other two types of Oracle objects, sequences and synonyms, which are widely used in the physical design of an Oracle database.

14.6.8 Creating Sequences and Synonyms

An Oracle sequence is basically a counter that increments each time it is used. This is a very useful and necessary feature as many ID columns require unique numbers to identify the rows of a table. For example, the table `TRANSACTION` has a `TRANSACTION_ID` column whose value can be assigned with a sequence object. However, not all ID columns can be populated with sequence objects. For example, account and customer IDs are not necessarily random unique numbers. They might have some implied meanings embedded with them, for example, partitioning information.

The full syntax for creating a sequence object is shown below. All options are self-explanatory except that the `CYCLE` option specifies that the sequence numbers should loop repeatedly between min and max values specified.

```
CREATE SEQUENCE sequence_name [INCREMENT BY increment_quantity]
[START WITH starting_value] [MAXVALUE max_value] [MINVALUE min_value]
[CYCLE];
```

A sequence object has two “pseudocolumns” named CURRVAL and NEXTVAL. For example, after creating a sequence object with the following SQL:

```
CREATE SEQUENCE transaction_id_seq;
```

the next transaction ID can be populated with `transaction_id_seq.nextval` in an INSERT SQL statement. This can be used not only in production but also in a test script.

Although a sequence object can be used across multiple tables, usually it’s recommended to use a sequence object on a per-table basis. In addition, from performance and scalability perspectives, it might be necessary to grab a block of sequence numbers a time rather than one by one, to help mitigate contention on the underlying sequence object. Refer to Section 14.6.4 for the sequences created for SOBA.

The other type of Oracle object to be covered in this section is a *synonym*. A synonym is just an alias for a table or a view. You can create a synonym with the following command:

```
CREATE SYNONYM synonym_name FOR object_name;
```

Then you can query the synonym just as if it were the table or view it was created for. A synonym doesn’t have anything to do with performance or scalability. It’s just for convenience, for example, when the name of a table or view is really long, and you want to have a more mnemonic one. Section 14.6.4 provides an example of a synonym named USERS, which is a synonym to the LOGINUSER table. As we will see later, the application tier expects a table named USERS for authenticating users, so a synonym is a perfect solution in this case.

14.6.9 Adding Indexes

Adding proper indexes helps speed up SELECT queries. However, it has undesirable side effects on INSERT, UPDATE, and DELETE performance, because indexes must be created, updated, or deleted as part of these operations. But typically the benefits of having indexes far outweigh the adverse effects on INSERT, UPDATE and DELETE SQLs, so adding proper indexes has always been a necessary part of the physical design of a database.

Here are some guidelines about creating indexes for an Oracle database:

- Oracle uses the naming convention of `TNAME_CNAME_Index_Type` where the first two parts represent the names of the table and column and the last part represents the type of the index with PK, UK, IX for Primary Key index, Unique constraint index and non-unique index, respectively. Since it has too many parts, abbreviations can be used for the names of the table and column(s).
- Oracle creates an index on a primary key and a unique constraint automatically, but not on a foreign key. In general, indexes (non-unique) on foreign keys are necessary for speeding up joins.

- A general rule of thumb is that if a query is anticipated to be used frequently, then an index should be created on the columns that appear in the query's WHERE clause.
- Larger tables need indexes more than smaller tables.
- Try to limit the number of indexes created on a table. It's possible to consolidate some indexes into fewer ones without compromising the performance of the queries that are dependent on those indexes.

As was introduced in Section 14.6.4, a non-unique index named `IX_TRANS_ACCOUNT_ID_TRANS_DATE` was created on the `ACCOUNT_ID` and `TRANS_DATE` columns of the table `TRANSACTION` using the following SQL:

```
CREATE INDEX IX_TRANS_ACCOUNT_ID_TRANS_DATE
on TRANSACTION (ACCOUNT_ID, TRANS_DATE) ;
```

This index was intended to speed up queries based on transaction ID and transaction date. However, according to my experience, most of the indexes would be created during internal performance and scalability tests with adequate data volume, representative use scenarios, and properly sized hardware. Some additional indexes would come up from the observations in customer's production environment. Refer to some of the case studies presented in the last part of this text to learn how indexing could help improve the performance and scalability of Oracle-based enterprise applications significantly.

14.6.10 Security

As part of the SOBA security, the following four schema objects were created in Section 14.6.4:

- `LOGINUSER` table—Stores username and credential, which are required for accessing SOBA online
- `AUTHORITIES` table—Stores user roles or authorities, which are used to control access to SOBA resources
- `ACL` table—Stores domain object permissions based on the concept of an access control list (ACL)
- `USER_AUTHORITIES` trigger—Inserts the role of `ROLE_CUST` into the `AUTHORITIES` table whenever a customer is created

These security schema objects are created either for authentication or for authorization as required for securing SOBA. The next chapter provides more detailed coverage on security enforcement using these security schema objects, especially how domain objects can be secured by enabling access control list (ACL) on them.

14.7 IMPLEMENTATION

Implementing a large-scale, Oracle-based enterprise application involves not only a good design to begin with but also a well-orchestrated implementation process in order to succeed. It cannot be simply a trial-by-error exercise. Given the various practical constraints such as limited budget and time frame, the experiences and skill sets of the existing development team, it certainly matters how an end-to-end implementation from starting coding to releasing to market is actually carried out toward the ultimate goal of delivering a product that helps enhance the competitive edge of your organization.

In general, there is no panacea about how to carry out a complex software product implementation process to a successful end, as there are too many factors that can easily derail it. However, on the optimistic side, there are proven methodologies that are helpful for mitigating the potential risks of failures of a software project. In this section, we explore some of such proven methodologies in the context of implementing the secure online banking application we started in this chapter, including:

- Choosing an effective and efficient coding path
- Leveraging proven Oracle database design principles
- Leveraging proven application design patterns
- Enforcing with an effective and efficient testing process

Let's begin with discussing choosing an effective and efficient coding path next.

14.7.1 Choosing an Effective and Efficient Coding Path

Coding is a major part of implementing a software project of any size. Coding is not just about programming. A more important, deterministic decision to make is what coding path to follow. Specific to this secure online banking application in question, the following considerations need to be taken into account when determining the coding path:

- For a given feature, whether it should be coded on the database side or application side. Although there is no general, hard rule to favor one option against the other, it makes sense simply to let a database system do what it can do best. Don't be afraid of using such common, proven database features as triggers and/or stored procedures, and so on. Such features are safe to use, highly performing, and scalable intrinsically.
- On the application side, a decision needs to be made on which development platform should be chosen. Most likely, the choice is between Java platform and Microsoft .NET platform. Both are popular and robust software development platforms, with a major difference that .NET runs on Windows systems only while Java runs not only on Windows but also on various flavors of UNIX and Linux systems.

- Java—commercial or open source frameworks. Keep in mind that even with Java development platform, many options are available. For example, one can lean on some proven, commercial Java application server providers like IBM Websphere or Oracle WebLogic, or one can comb through many open source Java development frameworks available for free and pick one. There are pros and cons with each option, and sometimes, it might even be more of company policy based rather than technology based. But if you choose open source frameworks, exercise caution that some might be more reliable, highly performing, and scalable than others. I have seen with real product development that one had to switch midway from one open source framework to another because of the implementation issues that needed to be worked around. Such a major setback would be translated into the loss of time, and eventually the loss of competitive advantages in this extremely relentless market. Imagine that a real enterprise software product development could rarely be a single individual's effort, so if a fundamentally ineffective coding path were chosen, the efforts of the entire crew (development, QA, and performance and scalability tests) would be wasted.
- Effectiveness versus efficiency. A coding path might be proven an effective one based on internal and/or external experiences. However, a chosen effective coding path does not necessarily guarantee a smooth, efficient implementation with the end result of a successful delivery of a software product. It's important to make sure that an effective coding path could be followed and carried out efficiently, which implies completing the implementation on time with all major objectives accomplished.

Once again, there is no panacea about how to carry out a software product coding path efficiently, but the potential risks could be minimized if one can leverage some of the proven best design principles and design patterns both from database and application perspectives as well as enforce the likelihood of success with an effective and efficient testing process. These are the subjects of the next few sections.

14.7.2 Leveraging Proven Oracle Database Design Principles

Proven database design principles are helpful because they were distilled from practices of building database-centric software in the past several decades. One will have a better chance to succeed in building a complex enterprise software product if a disciplined approach is taken, for example, using proven design principles whenever possible. This is especially true with building Oracle-based large-scale enterprise applications, because of the weight and scale of Oracle in running today's business operations in many organizations around the globe.

In this section, we conceptually cover the following three most fundamental Oracle database design principles:

- Beginning with a sound data model.
- Minding your data quality.
- Using SQL design patterns whenever possible.

Note that this section is about Oracle database *design* principles, or in other words, it's not a full coverage of all Oracle-related best practices to get your Oracle database up and running optimally in terms of performance and scalability. One has a lot to do to optimize and tune an Oracle database for the best possible performance and scalability, which is covered in the other chapters of this text. The purpose of this section is to help you understand how to come up with a smart design from the beginning so that opportunities for optimizing and tuning the performance and scalability of your Oracle database will exist and can be exploited in the later stages of implementation and deployment.

Not surprisingly, beginning with a sound data model is the number one most important Oracle database design principle. This is because the data model conceptually stays at the bottom, and if the data model is not stable that frequent changes are required, then all database logic as well as application logic built on it might be disturbed, resulting in costly recoding and retesting efforts from the entire development crew. Moreover, an unstable data model would cause enormous difficulties for upgrading the application from older to newer versions. These are non-trivial issues as the tangible consequences would eventually have to come out of the bottom line of an organization, namely, the development organization would have to pay for such significant setbacks.

The quantitative aspect of a sound data model is that it needs to be normalized at least to the 3NF. You might have heard that in order to trade for performance it might be necessary to denormalize a schema design to below 3NF. Note that denormalization is rarely necessary as significant breakthroughs with all types of hardware resources (CPU, memory, network, and storage) have been made steadily from year to year. However, if you think that denormalization is the only option to overcome some performance and scalability barriers with your Oracle-based application, it's necessary to have reliable test data to back it up.

The qualitative aspect of a sound data model is a measure of whether your data model can meet your near-term, mid-term, and long-term needs without having to be remodeled down the road. We already stated the undesirable consequences with frequently changing a data model in the previous paragraph. However, a balance needs to be made between the required stability of a data model and required flexibility to accommodate the unforeseeable future needs for changes with the application. There is no hard rule on such a delicate issue. The most proper decision depends on the priority of an organization between releasing to market sooner and minimizing long-term development cost.

Regarding data quality principle, it's about whether there are too many tables that are stuffed with too many attributes that actually may not be needed. Certainly, one needs to make sure that all necessary attributes are in place. However, I have seen with real products that tables were filled up with hundreds of attributes, which turned out to be a performance and scalability killer for the application. Eventually, those tables had to be cleansed with fewer attributes to meet the performance and scalability requirements. Retrospectively, it's not clear why those extra attributes were put into those tables, but if you are designing a new database, keep it in mind that having too many not very useful attributes for a table may hurt performance and scalability badly.

The principle of using SQL design patterns is an interesting one. First, note that you may not need to code SQLs directly on the database side. SQLs are typically issued from the application side. Nowadays it has become very popular to use a separate object-to-relation mapping (ORM) technology on the application side to take care of generating various types of SQLs automatically. A sound ORM implementation uses proven SQL design patterns to the largest extent, so there is less burden on the database developer or administrator in this regard. However, if you find poorly composed SQLs from an ORM implementation through your performance and scalability tests, options are available to apply some influence on the SQLs originated from the ORM. On the other hand, if instead of using an ORM, you are using some other mechanism to issue your own SQLs, such as using standard JDBC, make sure that SQLs are composed optimally and tunable. The worst thing one should always try to avoid is to retrieve all the columns of a table from the database and then pick only a few as needed on the application side. Quite surprisingly, many people actually believe retrieving all columns of a table should cost the same as retrieving only a few columns of a table. I once had a real experience with a real product that the application was consuming a lot of system resources by repeatedly executing one “SELECT * FROM <table> . . .” statement, which was *a retrieving-all* SQL query. I approached the manager of the development team to see if it’s possible to limit the columns of the table to be selected on to those only actually needed but got a blunt reply that “No. We need to get prepared for meeting not only the needs of today but also the needs of future.” It’s so ironic that the company was acquired by another company and the project was cancelled and never saw a future. The moral of the story is that if we can’t deal with it today, it may not have a future to worry about!

In the next section, we explore leveraging proven application design patterns for building highly performing, scalable Oracle-based enterprise applications.

14.7.3 Leveraging Proven Application Design Patterns

Despite of various definitions on design patterns, a design pattern simply is a proven solution to a common problem in the realm of software development. Using design patterns as much as possible is recommended, as it can help develop more reliable software more predictably by preventing developers from reinventing the wheels or falling into the quagmire of repeatedly hitting-and-missing in coding an application.

Before examining some of the common design patterns, I’d like to offer a general description about the design patterns from the following multiple perspectives:

- First, a design pattern is language independent. A design pattern describes the same methodology of solving a common underlying design problem. A same pattern can be implemented in any languages. Therefore, the first step toward applying a design pattern is to understand what problem it solves; or for a given problem, look up the common design patterns and find out which pattern matches your problem as a solution.

- Although design patterns are language independent, eventually one has to choose a specific language to realize them, ranging from C++ , to C# or Java, or even to many popular scripting languages. For your interest in a specific language domain, you can search the Web and find out the most relevant resources easily. For Java developers, a good text is James W. Cooper's book titled *Java Design Patterns* as listed at the end of this chapter.
- As a developer, most of the time, you may not need to implement a design pattern from scratch by yourself. For example, if you are using a solid open source Java development framework like Spring Source, many design patterns have already been built into the framework, and the only thing left for you is to actually use them. In such a case, your task would be to understand how they work and how to use them.
- Design patterns are categorized according to the problems they solve, such as:
 - Creational patterns that are responsible for taking care of the logistic details of creating objects. As is evident by its name, a *factory* design pattern belongs to this category.
 - Structural patterns that act as fundamental elements for building larger and more complex software systems from a structural point of view. If you have already had some exposures to software development, you might be familiar with such design patterns as *adapter*, *facade*, *proxy*, and so on, which belong to this category.
 - Behavioral patterns that facilitate the communication and flow logic between objects. A typical pattern in this category is the *publish/subscribe* pattern or *observer* pattern that serves as the infrastructural mechanism of almost every messaging application.

Although this text is not entirely about design patterns, we'll explore later in detail a very common design pattern used in developing Web applications, the Model-View-Controller (MVC) design pattern. The MVC design pattern will be introduced along with implementing the secure online banking application with an Oracle database as the data store. You will see how convenient it is to develop a Web application using a robust Java open source framework like Spring Source that has the MVC design pattern built in.

Next, I'll help you understand that coding is only one part of a software development process. The other major part is testing, which is orchestrated from multiple levels with one level built upon another.

14.7.4 Enforcing with an Effective and Efficient Testing Process

There was a notion that if developers were truly capable, then basically certain tests at a certain level would not be needed. It's certainly true that more capable developers create better quality software with fewer bugs, but it's impractical for any developers to foresee all defects in various forms when the software under development gets sufficiently complex. Therefore, adequate tests at various levels are always an indispensable part of a successful software project.

Software quality assurance (QA) and usability tests should be performed at the following various levels:

- *Unit Testing.* This is the most basic type of testing to help make sure that a product works at each single programming unit, which usually is a method of a class in object-oriented programming languages (not surprisingly, if a software system has to work, then it has to work at the level of each of its smallest constituent unit). Note that:
 - Unit testing programs go side by side with the main source programs. They test not only all normal operations but also all potential exceptions of each programming unit.
 - It's very rare that unit testing programs are developed from scratch. Many unit testing frameworks exist to help facilitate the tasks of unit testing. On the Java platform, two unit testing frameworks, JUnit and TestNG, are widely used. If you are using Spring Source, further built-in unit testing supports are available.
 - Unit testing should be part of coding the main functions of a product. Developers should run unit testing before checking in their implementations and changes to the main source code. That way, bugs can be caught at the unit testing level so that they will not be propagated into a formal build at the system level, thus preventing wasting the entire crew's time.
- *Integration Testing.* This is the next level of testing above unit testing. It tests the interactions of various objects to make sure that each component works as expected. To help reduce dependency and complexity, stubs and mock objects might need to be introduced. Both stubs and mocks are designed as substitutes for real objects. However, there are certain subtle differences between a stub and a mock object:
 - A stub responds to the method calls of its caller in a predetermined way, for example, with hard-coded data. Then the caller can assert if the returned result is what was expected. With a stub, the caller would not know what methods of the callee or stub were actually called other than the result returned from the callee.
 - In contrast to a stub, a mock object checks and verifies the methods of the dependent object expected to be called so that it gives a finer granularity into what methods actually called inside a dependent object. Therefore, usually, a stub is used for state verification, while a mock is used for behavior verification. Once again, you should leverage some of the available libraries to create your mock objects, for example, using EasyMock and jMock if you are on Java.
- *Functional Testing.* This type of testing is also more commonly called QA testing. It goes beyond the unit and component levels by testing a product at the system level to help verify that the system would work correctly as if it were used in a real environment except that it would not simulate the load intensity typical

in a real environment. Higher load intensity testing or volume testing belongs to the next level of testing—performance and scalability testing—as described next.

- *Performance and Scalability Testing.* The necessity for this level of testing is obvious. With QA testing, its objective is to verify if the system would work under very ideal conditions, for example, when only one or a few users are using it if it's a Web application. But how the system would respond in a normal production environment if hundreds or thousands of or even more users access it concurrently? For any commercial software product, it must be usable under normal usage at a customer's site. The purposes of performance and scalability testing are multifold:
 - If a product has never been tested at the performance and scalability testing level, then it's necessary to establish an initial performance baseline with projected workloads on the hardware systems comparable to what would be used in a production environment. This is also a good opportunity for optimizing and tuning the performance and scalability of the product from both software and hardware perspectives, with plenty of low-hanging fruits to harvest.
 - After completing the most basic performance and scalability tests as described above, the next step is sizing testing, which answers questions like: (1) For given workloads and hardware, what would be the best possible performance of the typical use cases tested? (2) How would the system performance degrade or improve if the workload intensity and/or hardware are scaled up or down? Such information is valuable for guiding customer deployment of the product, as is emphasized in the next section.

Note that performance and scalability testing differs from QA testing because it requires a *testable* build to start with. However, one should not wait until such a testable build is available. One can start as early as when an *installable* build is available. That's because even with an installable but not necessarily fully testable build, one can get an early peek at the architecture and implementation of the various functions of the product and get started with the preparatory work for more formal performance and scalability testing to be conducted later. It's also possible to start developing some test scripts or even conduct some preliminary tests with less data or lower workload intensity using an installable but not necessarily fully testable build. By starting early with a build that is at least installable, some intrinsic performance and scalability problems might be discovered early in the product development life cycle so that it would be less costly to fix them. Proactive rather than reactive or passive thinking can help improve the efficiency of performance and scalability testing significantly.

Assuming that a product has been tested thoroughly at all levels, the next stage is to release to market, which is commonly abbreviated as RTM or GA (general availability). This is a significant milestone for a product. Next, let's see what need to be shipped in addition to the software itself during the RTM phase.

14.8 RELEASE TO MARKET (RTM)

The quality of a software product alone may not guarantee its immediate success. In addition to hefty marketing efforts, a series of high-quality product documentations are required to accompany the release of the software. These documentations should cover the following subjects:

- Overview of the product architecture. This will help customers understand what technologies are used to build the product and how the product works.
- A getting-started guide to help walk a user through the steps of how to set it up and perform some simple, typical tasks.
- A hardware sizing guide to help customers determine what hardware systems will be needed with projected usage and workloads. Such a step is extremely important, as it would be costly to resolve a customer performance escalation. I personally have participated in resolving many customer performance escalations, most of which could have been avoided if the hardware was sized properly from the beginning at the customer's sites.
- Best practices. It's not very uncommon that the performance of an enterprise application could be improved by multiple times or even orders of magnitude with the optimal settings obtained with adequate performance and scalability tests. Both customers and ultimately the software vendor can benefit enormously from such best practices. This is a necessary measure to take in order to secure the highest-possible customer satisfaction.

After a product is RTMed or GAed, continuous improvements on the quality of the product should be sought after, as discussed next.

14.9 CONTINUOUS IMPROVEMENTS

Post-RTM activities typically include resolving customer escalations, adding patches, and preparing for the next version, and so on. Here, we won't dive into each of these areas. Instead, I'll offer a few quick tips on how one can continue to improve the performance and scalability of a product that has been RTMed or GAed. These tips include:

- *Actively Engaging Customer Feedback.* It's important to realize that the performance and scalability tests conducted internally might be skewed in the following aspects:
 - The use cases tested internally might not cover all use cases from real customers.
 - The workload profiles used with internal tests might differ from those in real customer environments.
 - Internal tests might have heavily weighted on synthetic data, which might not be sufficiently representative of customer data.

Such differences between internal test environment and customer environment need to be reconciled, which calls for actively engaging customer feedback. The improved use cases, workload profiles, and quality of data can help improve the performance and scalability of the future versions of the same product significantly.

- *Leveraging Customer Escalations.* It's very rare that a large-scale enterprise application would incur no customer performance escalations. Although such escalations are costly and should be prevented as much as possible through pre-release performance and scalability testing, they should be taken as valuable lessons to help build a more rigorous and comprehensive internal performance and scalability testing methodology so that chances for escalations on future versions of the product can be minimized as much as possible.
- *Expanding the Scope of Performance and Scalability Testing Post-RTM.* Because of the time constraint, performance and scalability testing prior to RTM might be limited to certain platforms with certain specific configurations only. Post-RTM provides an opportunity for expanding the scope of performance and scalability testing to cover more platforms if the product runs on multiple platforms. In addition, stressing testing for finding the scalability limit of the product can be arranged and carried out post-RTM. All in all, post-RTM is just another opportunity too valuable to pass by.

All in all, an RTM is only a new start for meeting new and more challenges with a large-scale enterprise application. Oracle releases new versions continuously, and so do all Oracle-based enterprise applications.

14.10 SUMMARY

In this chapter, we explored how to design and build performance and scalability into an Oracle-based application. We started with looking at various development methodologies that set the stage for developing a software application. We then analyzed the life cycle of a software project, from planning, through all the subsequent stages of requirements gathering, conceptual design via data modeling, logical design via normalization, physical design, and implementation, all the way up to RTM and post-RTM continuous improvements. We spent quite some time on the theory of normalization, which I hope would help establish a solid mathematical framework for you to truly understand the complexity of normalization.

In the next chapter, we'll implement this sample application of SOBA mainly using the Spring Framework, based on the SOBA database we designed in this chapter. Be advised that it's going to be rough if you do not have sufficient knowledge about how computers work and how software is built, especially in Java. However, if you take a relaxed approach by doing a little bit a time, I am sure you will be able to contain it.

RECOMMENDED READING

For an overview of UML, refer to the following text:

H. Eriksson and M. Penker, *UML Toolkit*, Wiley, New York, 1998.

If you want to know from where the concept of relational data model originated, read the following epic-making paper by the creator (also, it's worthwhile to spend a few moments to know more about Dr. Codd at http://en.wikipedia.org/wiki/Edgar_F._Codd):

E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Comm. of ACM* 13:6 (1970).

For those who are serious about 4NF and 5NF, studying the following classic papers is strongly recommended (these were the authors who defined 4NF and 5NF):

R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases," *ACM Trans. Database Syst.* 2(3) (Sept. 1977), 26–278.

R. Fagin, "Normal Forms and Relational Database Operators," P. A. Bernstein, ed., *Proc. of the 1979 ACM SIGMOD conference*, pp. 153–160.

C. J. Date and R. Fagin, "Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases," *ACM Trans. Database Syst.* 17(3) (Sept 1992), 465–476.

The following two texts are two typical database textbooks with good coverage on basic database concepts and theories:

R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed., McGraw-Hill Science/Engineering/Math, New York, 2002.

P. O'Neil and E. O'Neil, *Database: Principles, Programming and Performance*, Morgan Kaufman, San Francisco, 2002.

The following two texts are good reference books for practical database design:

T. Kyte, *Effective Oracle by Design* (Osborne ORACLE Press Series), McGraw-Hill Osborne Media, Emeryville, 2003.

J. Date, *Database in Depth: Relational Theory for Practitioners*, O'Reilly Media, Upper Saddle River, 2005.

Texts about design patterns:

E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st edn. Addison-Wesley Professional, Reading, 1994.

J. W. Cooper, *Java Design Patterns: A Tutorial*, Addison Wesley, Sebastopol, 2000.

Texts recommended for learning more about Spring Framework and RESTful Web services:

Spring Documentations:

<http://www.springsource.org/documentation>.

<http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf>

RESTful Web services:

Bill Burke, *RESTful Java with JAX-RS*, O'Reilly, 2010.

EXERCISES

- 14.1** What are the factors for determining which development methodology to be used with a software product? If you are part of a software project, which methodology is in use with your project?
- 14.2** At what stage a feasibility study is called for? How would you go about it and what would be the expected deliverables out of a feasibility study?
- 14.3** What are the differences between use cases and user views? Give some additional use cases and user views in addition to those mentioned in this chapter for the sample application SOBA.
- 14.4** If you are charged with designing a data model for a database-centric application, which modeling tool would you choose and what would be your justifications?
- 14.5** What are the objectives of normalization?
- 14.6** Explain the differences among a simple key, a compound key, a composite key, a superkey, a candidate key, a primary key, a foreign key, a trivial super key, and an all key.
- 14.7** What keys are indexed automatically in Oracle?
- 14.8** Name a few central concepts that govern various normalization levels.
- 14.9** Summarize what problem each level of normalization solves from 3NF to 5NF.
- 14.10** What are the major differences between the logical design and physical design of a database?
- 14.11** Refer to Section 14.6.4 on the two triggers created. Explain what those two triggers are meant for exactly.
- 14.12** What factors should be considered when choosing a development platform? If you have some experience in programming, which development platform is your favorite and why?
- 14.13** How would you leverage proven database design principles and application design patterns? How would you strike a balance between the two?
- 14.14** List the characteristics of testing at various levels. How would you orchestrate those testing efforts at various levels so that maximum effectiveness and efficiency could be achieved as much as possible?

15

Project: Soba—A Secure Online Banking Application On Oracle

*Anything that we can do to raise personal savings is very much in the interest of
this country.*

—Alan Greenspan

In this chapter, we take an end-to-end approach to building a sample secure online banking application named SOBA that runs on Oracle. I decided to take it this far based on my observation that almost every Oracle database plays the role of a data-tier or backend-tier while having the application logic coded on the application tier, which communicates with Oracle to store and retrieve data. In front of an application tier typically is a Web tier that receives requests from and sends responses back to clients. The client tier consists of Web browsers and human users. This is a typical *n-tier* architecture that most of today's enterprise applications are built upon. See Figure 15.1 for a *logical* illustration of an *n-tier* software application architecture. Note that it's a *logical* illustration because all tiers can simply be deployed on one system, as is the case with the development of SOBA demonstrated here.

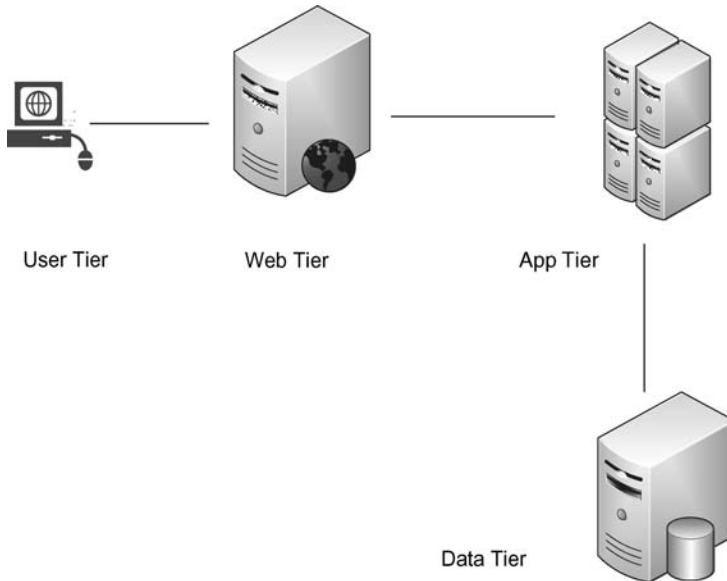


Figure 15.1 A logical illustration of an *n*-tier software application architecture.

With SOBA, Oracle has been chosen to serve as the backend tier for obvious reasons. However, there are many options to build the application tier, out of which the Spring Source Framework has been chosen as the top choice for developing SOBA. This choice of Spring is majorly based on the fact that Spring Source Framework has a huge developer base in all Java communities across several continents, and also because I happen to know more about Java and Spring Source than any other development platforms.

This chapter mainly consists of the following sections:

- Getting SOBA Up and Running
- Overview of Spring Framework
- MVC Architecture
- Spring MVC Framework Applied to SOBA
- Hibernate Object-Relational Mapping (ORM) Applied to SOBA
- RESTful Web Services Applied to SOBA
- Spring Security Applied to SOBA
- Spring ACL Applied to SOBA

Let's first get SOBA up and running before we examine how it was built with a typical development stack of Oracle, Spring, Hibernate, and RESTful Web services.

15.1 GETTING SOBA UP AND RUNNING

To get SOBA up and running, follow the procedure given next to complete the setup.

15.1.1 Prerequisite Software

Driven by the dependencies, let's start with the bottom of the entire SOBA stack, which is Oracle. We then list the stack components upwards. All software products that are required to develop SOBA include (Note: this is more of a list of all products that I used and that I know would work. If you deviate from the version numbers listed below, there is a fair chance that it may not work or unanticipated issues may occur. In general, based on my experience, it is extremely crucial to pay attention to the version of a product, as it's more likely than not that an untested version may not work compatibly.):

- Oracle 11g R2
- 32-bit JDK 1.6.0 update 23
- Oracle JDBC driver `ojdbc6.jar` that supports JDBC 4
- 32-bit Tomcat Web Server 6.0.14 (Windows version)
- Apache Ant 1.8.2
- Spring Framework 3.0.x
- Eclipse Helios Release
- Windows 7 64-bit Premium Home Edition. The system I used was a desktop with an AMD Phenom II quad-core 830 @ 2.8 GHz (2 MB L2 + 4 MB L3 Cache and 4 GHz system bus), 6 GB RAM, and 1 TB SATA disk @ 7200 RPM and 64 MB Cache (for under \$600).

I'd like to emphasize that this is the entire software stack including the hardware that I used and tested, or more specifically that I know it would work if configured similarly to what I have gone through. Although I'll try my best to cover all particular settings and configurations as we proceed, it's impossible to match everyone's knowledge and experience background, and therefore it's not guaranteed that nobody would encounter no issues at all. So, to set the expectations properly, I have to assume that the reader has some minimum exposure to how computers work, how software is built in general, how Java works, and so on. And most importantly, I assume the reader would have a great issue-hunting capability through diligent self-research. In particular, things like how to set an environment variable, how to install a JDK or Eclipse, and so on, will not be covered, because almost everyone is able to learn such things instantly. Or, I should say that the only area that I don't assume the user knows a lot about is Spring Framework. I hope you would agree that this is a fair assumption.

Assuming that you have acquired these software products, next a brief covering of the initial setup is given.

15.1.2 Initial Software Stack Setup

If you have your desktop or laptop ready, install and configure the following software in the same order as given below:

1. Install Oracle 11GR2 database. Make sure you can connect to your database via SQL*Plus.
2. Install JDK 1.6.0 or above. Set the JAVA_HOME environment variable on your system.
3. Install Eclipse Helios.
4. Install Apache ANT 1.8.2 or above. Set ANT_HOME environment variable and add Ant to your Windows PATH environment variable.
5. Install Tomcat 6.0.14. Execute the following command to create an SSL certificate for your Tomcat to use with the HTTPS protocol. Then edit your Tomcat `server.xml` file as described below to enable HTTPS to be used with SOBA:

(a) `%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA`

A few things to pay attention to:

- Use “changeit” for the password (without quotation marks)
- When asked “What is your first and last name?,” enter “localhost” or your machine name. If you enter you real first and last names like “henry liu” in my case, then when you try to test a RESTful Web API with HTTPS using a URL like `https://localhost:8443/soba/...` as will be illustrated in Section 15.4, an error will occur complaining that “hostname didn’t match: localhost != henry liu.” All other entries for creating the certificate don’t seem to matter much.

- (b) Find and edit your `server.xml` file in the directory of `<tomcat_install>\conf` by un-commenting the following section by removing `<!--` and `-->` around it:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"/>
```

- (c) Add the following line in your `tomcat-users.xml` file in the same `conf` directory:

```
<user username="tomcat" password="s3cret" roles="manager"/>
```

6. Copy Oracle JDBC driver `ojdbc6.jar` to the `lib` directory of your Tomcat install.

Next, instructions are given on how to setup and configure SOBA on Oracle and Eclipse IDE.

15.1.3 Creating SOBA Database on Oracle

If you already created the SOBA database on Oracle by following the procedure given in Section 14.6, you may proceed to the next section of *Installing SOBA on Eclipse IDE*. Otherwise, follow the procedure given in Section 14.6 to create the SOBA database on Oracle.

15.1.4 Installing SOBA on Eclipse IDE

To install SOBA on your Eclipse IDE, follow the below procedure:

1. Install Eclipse IDE Helios. Note that if you have never used Eclipse, you might need to take a quick self-training on how to get around in an Eclipse Java IDE in general.
2. Download the SOBA project zip file from this book's Web site and import it into your Eclipse workspace as follows:

```
Click File -> Import -> Existing Projects into Workspace ->
Next -> Select archive file -> Browse <select the downloaded
zip file and proceed to complete it>
```

After the SOBA project is imported successfully, you should see a similar structure of the SOBA project as shown in Figure 15.2 in Eclipse. Note the Java packages of `model.vo`, `model.dao`, `restfulweb`, `service`, `test`, `utility`, and `web`. Note also the `war` folder for deployment with a standard structure of a `WEB-INF` folder. This folder contains a `classes` folder for all compiled Java classes as well as a `jsp` folder for all jsp files except the `login.jsp` file, which is placed under the `WEB-INF` folder directly. The Ant build files are placed under the project folder directly.

3. Make sure you have all jar files required for SOBA added to the SOBA project in your Eclipse Java IDE. Figure 15.3 shows all jars added to the SOBA project, while Figure 15.4 shows how these jars were added to the SOBA project. At this point, you should not see any SOBA source files marked with a red "x" indicating syntax errors. You need to fix them if any.
4. Locate the `build.properties` file available in SOBA's project folder. Make changes to the entries underlined to match your settings:

```
user.home=c:
appserver.home=c:/tm6014
appserver.lib=${appserver.home}/lib

deploy.path=${appserver.home}/webapps

tomcat.manager.url=http://localhost:8080/manager
tomcat.manager.username=tomcat
tomcat.manager.password=s3cret
```



Figure 15.2 The structure of SOBA in Eclipse.

```
db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@p6620f:1521:orallgr2
db.user=OBAdmin
db.pw=OB#Admin
```

The next section covers how to configure SOBA to work with Oracle.

15.1.5 Configuring SOBA to Work with Oracle

To configure SOBA to work with your SOBA Oracle database created as described in Section 15.1.3, follow the below procedure:

1. Make sure the `ojdbc6.jar` file is added to your SOBA project.
2. Check your Oracle settings in the `jdbc.properties` file in the directory of `<project_dir>\war\WEB-INF\classes`.
3. Double check your Oracle settings in your `build.properties` file per step 4 of the preceding Section.

The next section covers how to configure SOBA to work with Hibernate.

antlr-2.7.6.jar	jta-1.1.jar
aopalliance-1.0.jar	junit-4.8.2.jar
asm-all-3.3.jar	log4j-1.2.14.jar
aspectjrt-1.6.8.jar	org.springframework.orm-3.0.4.RELEASE.jar
aspectjweaver-1.6.8.jar	org.springframework.oxm-3.0.4.RELEASE.jar
backport-util-concurrent-3.0.jar	restclient-cli-2[1].3.2-jar-with-dependencies.jar
cglib-2.2.jar	restclient-ui-2[1].3.2-jar-with-dependencies.jar
commons-codec.jar	servlet-api.jar
commons-collections-3.1.jar	slf4j-api-1.6.0.jar
commons-dbcp.jar	slf4j-log4j12-1.6.0.jar
commons-logging.jar	spring-aop-3.0.3.RELEASE.jar
commons-pool.jar	spring-asm-3.0.3.RELEASE.jar
dom4j-1.6.1.jar	spring-beans-3.0.3.RELEASE.jar
ehcache-1.6.2.jar	spring-context-3.0.3.RELEASE.jar
hessian-3.0.1.jar	spring-context-support-3.0.3.RELEASE.jar
hibernate3.jar	spring-core-3.0.3.RELEASE.jar
hsqldb-1.8.0.10.jar	spring-expression-3.0.3.RELEASE.jar
httpclient-4.0.3.jar	spring-jdbc-3.0.3.RELEASE.jar
httpcore-4.0.1.jar	spring-security-acl-3.0.3.RELEASE.jar
httpcore-nio-4.0.1.jar	spring-security-config-3.0.3.RELEASE.jar
httpmime-4.0.3.jar	spring-security-core-3.0.3.RELEASE.jar
jackson-core-asl-1.6.2.jar	spring-security-core-tiger-2[1].0.3.jar
jackson-mapper-asl-1.6.2.jar	spring-security-taglibs-3.0.3.RELEASE.jar
javassist-3.12.0.GA.jar	spring-security-web-3.0.3.RELEASE.jar
javax.persistence.jar	spring-tx-3.0.3.RELEASE.jar
jaxb-api-2.1.jar	spring-web-3.0.3.RELEASE.jar
jaxb-impl-2.1.12.jar	spring-webmvc-3.0.3.RELEASE.jar
jcl-over-slf4j-1.6.0.jar	standard-1.1.2.jar
jstl-1.1.2.jar	

Figure 15.3 All jars required for SOBA in Eclipse.

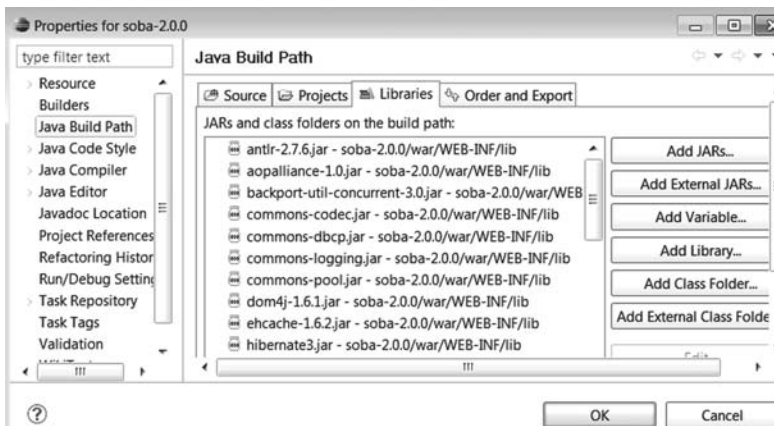


Figure 15.4 How required jars added to the build path of SOBA in Eclipse.

15.1.6 Configuring SOBA to Work with Hibernate

To configure SOBA to work with Hibernate, follow the below procedure:

1. Locate the file `hibernate.cfg.xml` in the directory of `<project_dir>\war\WEB-INF\classes`. Make sure all Oracle settings match your environment.
2. Locate the file `BillPayment.hbm.xml` in the directory of `<project_dir>\war\WEB-INF\classes\com\perfmath\odps\soba\model\vo`.

Make sure the above two files exist in their respective Tomcat subdirectories under `webapps\soba\WEB-INF`. The next section covers how to build SOBA with Ant and how to deploy SOBA to run on Tomcat.

15.1.7 Building SOBA and Deploying SOBA with Ant to Run on Tomcat

I typically open up two MS-DOS command prompts: At one of them, I start up Tomcat by executing `tomcat6.exe`, and at the other, I issue the command of `%ANT_HOME%\bin\ant deploy reload` to reload SOBA into Tomcat. This Ant command would build, redeploy and reload SOBA onto Tomcat. However, I found that I could do that for only a few times, and then a Tomcat JVM PermGen error would occur. When the error occurs, I had to quit the Tomcat process using Windows Task Manager and restart it manually (Note: pressing Ctrl/C somehow did not stop Tomcat so I had to quit it using Windows Task Manager and then start up a new MS-DOS command prompt to restart it.).

If you have done all of the above properly, hopefully you could start up the login page of SOBA at `https://localhost:8443/soba`. Congratulations if you are greeted with a login form.

Next, we'll get to the details of how SOBA was developed with the Spring MVC Framework.

15.2 OVERVIEW OF SPRING FRAMEWORK

Before we start, it might be helpful to provide a little bit information on the background of Spring.

15.2.1 Background

If you have a minimum of programming experience in trying out a simple "Hello World!" Java program, you already know how Java works. On the other hand, if you are familiar with the concept of an Enterprise Java Bean (EJB), you know how powerful it is in terms of facilitating your Java-based enterprise application development, but in the meanwhile you might have realized how complex it is in getting

them to work. The EJB methodology advocates developing an enterprise application using components or modules. The life cycles of EJBs are managed with containers commercialized by a few vendors such as IBM with WebSphere and BEA with WebLogic, which is now part of Oracle.

An enterprise application has two parts: backend code dealing with business logic and front-end interface code presenting the application to users. The EJB camp focuses on streamlining backend code development using various enterprise application design patterns so that developers do not have to code everything from scratch, for example, transaction processing, concurrency control, remote procedure calls, security, and so on. Such services are provided by EJB containers and all a developer has to do is just to use them rather than code them. In this sense, an EJB is a great idea and also a great approach to developing large scale enterprise applications.

However, developers found out soon that EJBs were a lot more complex than they could get used to. The major high adoption barriers for EJBs include: excessive checked exceptions, required interfaces, abstract bean classes, and so on. In addition to this complexity issue, the other issue with EJBs was the performance penalty caused by remote method invocations (RMIs), which existed in EJB 1.0 but significantly improved in EJB 2.0 by introducing the concept of local interfaces. However, the complexities with EJBs remained there.

So between those two extremes of hello-world-like Java programs and fully fledged EJBs, a middle ground was found, which is *Spring Framework* plus *Hibernate* (we will demonstrate how SOBA was built with Spring Framework and Hibernate soon). As we already explained briefly, Hibernate is an ORM (Object-Relational Mapping) framework that focuses on Java object persistence into relational databases, while Spring Framework is a Java-based development framework for coding business logic using Plain Old Java Objects (POJOs). The term *POJO* was coined by Martin Fowler and his associates. A POJO is just a regular Java object without any dependencies on prescribed framework APIs, or to some extent, a POJO is just the opposite of an EJB as you can imagine. When Spring and Hibernate gained more and more acceptance, EJB 3.0 introduced JPA (Java Persistence API) to match what Hibernate does and accommodated POJOs like Spring as well, but once-disillusioned developers continued with Spring and Hibernate. Another reason that Spring has been gaining more and more acceptance is that it's an open source framework, and it continued to be open source even after being acquired by EMC/VMWare a few years ago. Because of its relative simplicity, high popularity, and open source nature, I decided to build the SOBA demo with Spring.

15.2.2 Spring for Building Flexible Applications Faster

Although it's impractical to cover all the subjects of Spring here (Spring 3.0 Reference Documentation itself is 791 pages long), a high-level overview of Spring would be helpful. Essentially, Spring is a lightweight solution for building *flexible* enterprise applications *faster*. Spring is module-based just like EJBs, making it

possible to build *flexible* applications. On the other hand, in addition to its own solid implementations such as Inversion of Control (IoC) containers and a fully-featured MVC framework, Spring also allows easy integration with other frameworks such as Struts (a UI API), Hibernate, JDBC abstraction layer, AOP (Aspect-Oriented Programming), and so on, making it possible to build flexible applications *faster*. AOP works along with OOP (Object-Oriented Programming) to facilitate modularizing software by providing solutions to such common *aspects* (or *crosscutting* issues in AOP literature) as logging, transaction management, and so on. We won't delve deep into each of these important features, but knowing what Spring can do without knowing how it's done (which takes too much time and is not an urgent issue unless you really need to use one or some of them) is equally beneficial.

Next, let's explore the core concepts of the IoC and dependency injection in the unique context of Spring.

15.2.3 Spring Inversion of Control (IoC) and Dependency Injection

Spring as a Java-based development platform differentiates itself from others by: (1) enabling building applications using POJOs, and (2) enabling applying services non-invasively to POJOs. The key concepts behind these capabilities are *Inversion of Control (IoC)* and *Dependency Injection*. From computer science and practical software development points of view, it's important to understand these two concepts.

It's true that software complexity comes from the dependencies among the various components and modules of an application. Developing a standalone component or a module might not be too hard, but the challenges associated with how to wire them together cannot be underestimated. The complexities with developing enterprise components are simplified in Spring with the use of a generic design principle of the *inversion of control* (IoC). To explain what the IoC is about, let's consider a Java component A that references another two Java components B and C. When an instance of component A is invoked, it needs to know how to reference the dependent components of B and C. That is typically made possible with a *lookup* service. In a traditional lookup, component A would make a request to its container, and the container would look up components B and C and provide component A with the references to components B and C. This approach is termed a *passive* lookup. With the IoC, component A would not request the references to components B and C. Instead, the lookup process is reversed that the references of component A to components B and C would be delivered by the IoC. So the traditional two-way lookup (one way request and one way delivery) is turned into a one-way service of delivery only with the concept of the IoC.

How is an IoC process accomplished? That's what the concept of a dependency injection (DI) is about. With the above example of components A, B and C, the references of A to B and C are specified (or *injected*) in an external configuration file. We'll see such examples with SOBA later.

Next, let's briefly introduce the features of Spring 3.0 before we get to the specifics of Spring Model-View-Controller framework in the next section.

15.2.4 Features of Spring 3.0

While facilitating developing modularized applications, the Spring Framework itself is modularized with about 20 modules that implement all features it supports. These modules are classified into the following categories:

- *Core Container.* The Core Container category includes the following modules:
 - *Core and Beans Modules.* These modules are the foundation of Spring Framework, which includes the Dependency Injection and IoC features. The standard factory design pattern is used for the implementation of the `BeanFactory` class, which has made it possible to remove the need for programmatic singletons and to decouple the specification and configuration of dependencies by internalizing specification in the code while externalizing configuration in an external file such as an XML file. We'll understand this better later when we see the code of implementing SOBA in Spring.
 - *Context.* This module basically provides a means for looking up objects, similar to a JNDI registry if you are familiar with Java. It also supports event-propagation, resource-loading, and the transparent creation of contexts using, for example, a servlet container. All these functions are achieved mostly through an `ApplicationContext` interface.
 - *The Expression Language Module.* This module extends the unified expression language (UEL) as specified in the JSP 2.1 specification. It is used for querying and manipulating object graphs at runtime. In addition, it supports retrieval of objects by name from Spring's IoC container. It also supports getting/setting property values, property assignment, method invocation, accessing the context of arrays, collections and indexes, logical and arithmetic operators, named variables, and so on.
- *Data Access/Integration.* This category consists of the modules of JDBC, ORM, JMS (Java Messaging Service), OXM. Each of these modules is described as follows:
 - *JDBC.* This module provides a JDBC abstraction interface and implementation. It removes the need to do tedious coding in order to enable data retrieval and storing between an application and a database. We have seen how JDBC is used with SOBA to enable interaction with Oracle in Chapter 10. We'll see this again later in this chapter.
 - *ORM.* This module provides an object-relational mapping interface and implementation to help bridge the gap between the OOP on the application side and the relational model on the database side. We'll see how ORM is used with SOBA as an alternative to JDBC to enable interaction with Oracle later.
 - *JMS.* This module is a JMS integration framework that simplifies the use of the JMS API to enable applications to exchange messages using standard Java Messaging Services protocols. It's similar to JDBC except that the context here is about messages rather than data as in the case of JDBC.

- *OXM*. The term OXM stands for Object/XML Mapping. The ORM APIs it supports include those data persistence API frameworks such as JPA (Java Persistence API), JDO (Java Data Objects provided by Oracle), Hibernate and iBatis, and so on. We'll see how Hibernate is used with SOBA later to map Java objects with tables in Oracle.
- *Web*. This category includes the modules such as the Web, Web-Servlet, Web-Struts, and Web-Portlet, and so on. The Web-Servlet module is especially popular as it provides the infrastructure needed for developing Web applications. Its major part is the MVC framework that is a standard architecture for building Web applications. This is also the architecture that SOBA is based on, as we'll see later.
- *AOP*. This model provides support for using AOP Alliance-compliant aspect-oriented programming model to solve crosscutting issues such as logging, transaction management, and so on, as we explained previously.
- *Instrumentation*. This category includes modules that provide support for instrumenting Java classes. It enables automatically discovering applications as managed resources and automatically exporting application beans to JMX for management. It exposes performance metrics of the resources and the Spring container at runtime so that other frameworks and products can use to build management and monitoring solutions.
- *Test*. This category provides modules to enable testing Spring components with standard testing frameworks such as JUnit and TestNG, and so on. It can be used to load and cache Spring Application Contexts, and also to create mock objects so that you can test your code standalone.

I hope the above high-level overview gives you a clear picture of what Spring is capable of doing. If you are a computer science student or an actual player in the field and interested in a career in developing large-scale enterprise applications, then Java/Spring/Oracle is a very good stack to focus on to build your marketable skill sets. That's one of my intentions to offer this SOBA demo in this text to help you get better preconditioned.

Next, we delve deep into how Spring's MVC implementation works.

15.3 MVC ARCHITECTURE

To best present the Spring MVC framework and how SOBA is built with it, this section is organized in two parts: (1) How a generic MVC architecture works in general, and (2) how the Spring MVC framework works behind SOBA, illustrated with a series of user views taken as screenshots of some Web pages of SOBA. Although this is not a text about teaching software design patterns, I do hope you can walk away with a solid understanding of how MVC works and more importantly, how you can apply MVC to build a Web application quickly either using Spring or the development platform of your choice.

15.3.1 MVC Architecture in General

First, note that the MVC architecture is a generic software architecture, which means that it can be implemented in any language. In this section, let's explore what's behind the concept of the MVC architecture without tying it to any specific language.

I found that it's easier to explain what the MVC architecture is about from a *teleological* point of view based on what get involved in a typical enterprise application. I'll use the notation of $X \Rightarrow Y$, which can be interpreted with a proposition like "Y exists or is called for as the result of X."

- *Application Data and State $\Rightarrow$ Model.* On the one hand, an application inevitably operates on and manages data. On the other hand, the collective value sets of the data items at different points of time constitute the state of the application at the corresponding points of time. It's the application data and state that the *model* part is most concerned with (note that some literatures use *behavior* rather than *state* to characterize the concept of a model. The terms of *state* and *behavior* should be interchangeable because a behavior is a manifest of a state in general). This explains why there is a *model* part in MVC. Another way describing the model part of an MVC is that it corresponds to the business entity (or data) and business logic of an application. In terms of a front-/mid-/data-layered architecture, the concept of a *model* logically corresponds to the data layer.
- *Application Data and State Presented to a User $\Rightarrow$ View.* There would be no need to have an application operate on and manage data and state if the data and state never need to be viewed by a user. The visual interface part is the *view* part of MVC. Whether that view is made possible through HTML or XHTML or ASP or JSP is irrelevant. In terms of a front-/mid-/data-layered architecture, the concept of a *view* logically corresponds to the front layer.
- *User-Application Interaction $\Rightarrow$ Controller.* With a user standing on one side and application data on the other side, which typically is called a backend, an application would not function if there were no paths for a user to query or reach the data of his interests stored at the backend. This role is fulfilled by a *controller*, which explains the existence of a *controller* in the MVC architecture. A controller takes input requests from a user, queries and/or operates on the application data and state, and then provides users with the information originated from the application data and state at the backend. In terms of a front-/mid-/data-layered architecture, the concept of a *controller* logically corresponds to the mid-layer.

Figure 15.5 explains further the inter-relationships among those parts of a model, a view and a controller of the MVC architecture. You can trace the logical flows among the MVC components as follows:

- The `user` interacts with the `view` directly with any of the following gestures:
 - Entering data that will be added to the `model`
 - Initiating queries about the data contained in the `model`
 - Initiating intended changes to the `model`

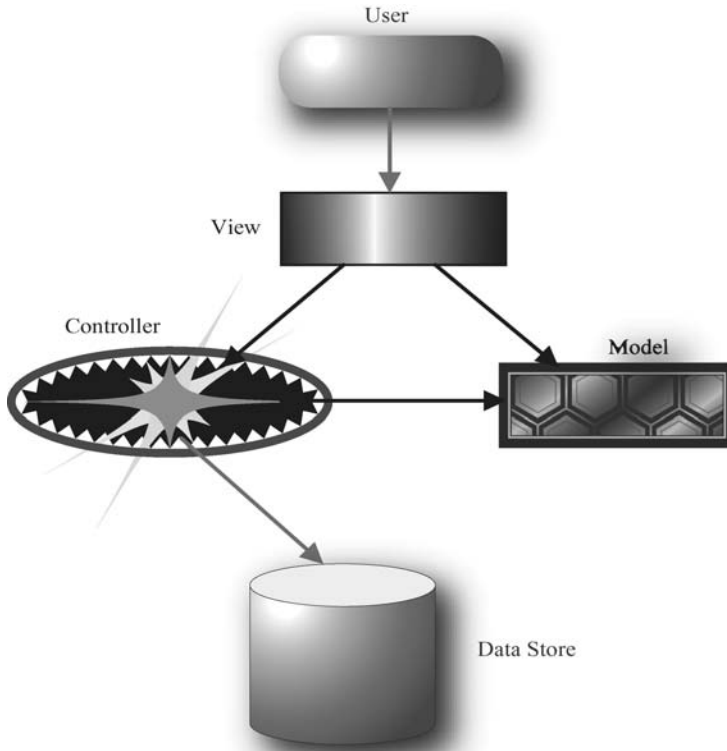


Figure 15.5 The MVC architecture.

- The view interacts with the model either indirectly through the controller or directly. Typically it's preferred to segregate the view and the model so that they will not be coupled with each other with *direct* interactions. This means that the view should send the request to the controller and let the controller handle it on its behalf.
- The controller may interact directly with the model on the user's behalf or update the model, change the view, and persist the change to the model at the database backend. It's clear that model, view, and controller are logical concepts and that the model is not equivalent to the physical backend.

A generic architecture is materialized into a framework if it is implemented by a vendor. Next, let's see how a generic MVC architecture is implemented by Spring and thus the Spring MVC framework.

At this point, if you have SOBA up and running on your system per the setup procedure described previously, you are ready to see how MVC plays out in SOBA. If not, then the series of screenshots to be presented in the next section should be

sufficient for you to understand the same. Either way, the SOBA demo will show you that the MVC architecture is not just a concept and that it can be put into practical use to build real applications.

15.3.2 Spring MVC in Action with SOBA

Like every secured Web application, the entry point to SOBA is the login page, as shown in Figure 15.6. We're already seeing MVC in action on this SOBA login page. First of all, by accessing this login page, the user expressed the gesture of becoming a new customer or accessing his account if he is already a customer. Secondly, this login page itself is a *view*, which displays partially the model behind SOBA. The login username and password are data items of the model. Besides, the two links, `Open Now` and `Register` above the login form, provide a user with intended changes to the model as we will see next. Assuming at this point there are no other users using SOBA, the system is in a fixed *state* with all customer data stored in the Oracle database. When the current user either opens an account or registers, changes would occur to the state of the application and thus to the model. The gesture of a user and the changes to the model are all coordinated by the controller, which will be initiated as soon as the user clicks one of the clickables on this login form.

Next, let's open an account to further illustrate MVC in action with SOBA. If you click the `Open Now` link, the view as shown in Figure 15.7 would appear. This is a form for creating a new customer. All the data entries will be added to the model after the `Submit` button is clicked. Then the updated view as reflected in Figure 15.8 will be presented to the user. This represents a change made to the state of the application or the model, as one new customer has been created.

Figure 15.6 The login page of SOBA.

SOBA :: create a new customer

First Name:	user1001
Last Name:	soba
Phone:	999-999-9999
Address:	One Way
City:	Any City
State:	CA
Zipcode:	95999
Email:	user1001@soba.com

Figure 15.7 SOBA: The form for creating a new customer.

Your customer ID 173666039 has been created successfully.

Use your customer ID to [create](#) your login ID for banking online.

Figure 15.8 SOBA: The view of displaying a customer is created successfully.

The next step is to create a login ID for the current user to gain access to his bank account online, as SOBA is an online banking application after all. After the “create your login ID” link is clicked, a new form of create a new login user is presented to the user, as shown in Figure 15.9. The resultant view after a successful creation of a login ID is shown in Figure 15.10. And a similar procedure ensues for creating an account for the current user, with the “create account” form and resultant view shown in Figures 15.11 and 15.12, respectively.

SOBA :: create a new login user

Customer Id:	173666039
Login Id (minimum length: 5):	user1001
Password:	••••••••

Figure 15.9 SOBA: The form for creating a new login user.

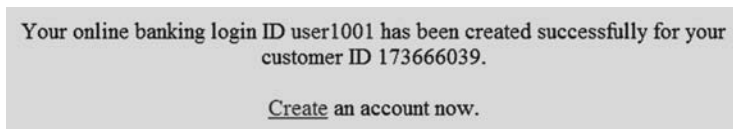


Figure 15.10 SOBA: The view of displaying a login user is created successfully.

SOBA :: create a new account

 A screenshot of a web form titled "SOBA :: create a new account". The form contains four input fields: "CustomerId:" with the value "173666039", "Name:" with the value "my checking", "Type:" with a dropdown menu showing "Checking", and "Description:" with the value "my checking online". Below the form is a "Submit" button.

Figure 15.11 SOBA: The form for creating a new account.

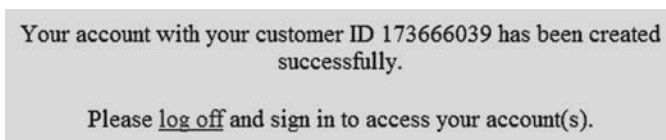


Figure 15.12 SOBA: The view of displaying an account is created successfully.

Next, let's expand further into all the wirings behind the Spring MVC framework that govern the logic flow of a typical Web application like SOBA. Since the processes for creating a customer, a login ID, and an account are similar, we'll focus on the `create a customer` process to illustrate how the Spring MVC framework is applied to SOBA with actual configurations and code snippets. If you have some or a lot of programming experience, you might prefer to see the code to understand how all these things work together to accomplish the relevant application functionality. That's exactly what we would do next.

15.4 SPRING MVC FRAMEWORK APPLIED TO SOBA

In Spring, each Java object or POJO constitutes a Java bean. An application is composed of various types of beans, which are managed by an IoC container. The configuration of a bean including its dependencies on other beans can be specified in an XML configuration file. At the startup of an application, these beans are loaded into an application context of an IoC container as specified in the XML configuration file.

The application context mentioned above is defined in an `ApplicationContext` interface, which is defined in the `org.springframework.context` package. The `ApplicationContext` interface extends the `BeanFactory` interface, which is defined in the `org.springframework.beans.factory` package. Unlike a `BeanFactory` interface, an `ApplicationContext` interface can be used in a completely declarative manner so that no programming in Java is needed on the developer's side. With a given XML configuration file that defines all beans one way or the other, a support class of `ContextLoader` makes it possible to automatically instantiate an `ApplicationContext` at the startup time of an application.

Since an `ApplicationContext` is an interface, it must be implemented in order to be usable. The `WebApplicationContext` is one of the implementations of the `ApplicationContext` interface. This `WebApplicationContext` interface along with a `DispatcherServlet` plays a critical role in the Spring MVC framework, as discussed next.

15.4.1 Spring DispatcherServlet and WebApplicationContext

As a Web application development framework, the Spring MVC framework has been designed around a `DispatcherServlet`, which is an expression of the "Front Controller" design pattern. A `DispatcherServlet` is defined in the package of `org.springframework.web.servlet`. A `DispatcherServlet` dispatches requests to handlers, which are managed in a `WebApplicationContext`. Figure 15.13 illustrates the interaction between a `DispatcherServlet` and a

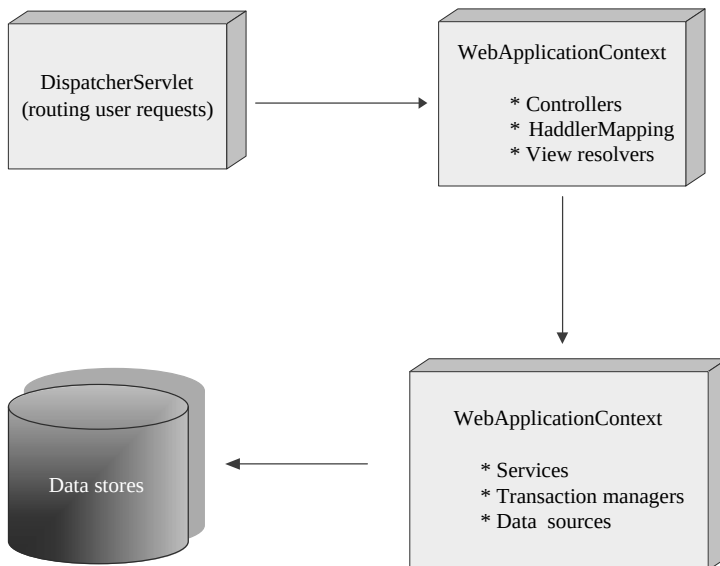


Figure 15.13 Spring MVC Framework: Interaction between a `DispatcherServlet` and a `WebApplicationContext`.

WebApplicationContext. We'll elaborate the concepts of controllers, handler mapping, view resolvers, and so on, soon.

To illustrate further the concrete use of the concepts of a DispatcherServlet and a WebApplicationContext, Listing 15.1 shows the `web.xml` file of SOBA. Because a Web application must run on a Web server or a servlet engine like Tomcat, there must be a `web.xml` file to help the Web server or a servlet engine to determine how to load up the deployment configurations of the application at startup, and thus the need for a `web.xml` file.

Listing 15.1 `web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <!-- Log4j configuration loading -->
  <listener>
    <listener-class>org.springframework.web.util.
      Log4jConfigListener</listener-class>
  </listener>
  <context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>/WEB-INF/classes/log4j.xml
    </param-value>
  </context-param>
  <!-- Bootstrapping context loading -->
  <listener>
    <listener-class>org.springframework.web.
      context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/soba-servlet.xml
      /WEB-INF/soba-services.xml
      /WEB-INF/soba-security.xml
    </param-value>
  </context-param>
  <context-param>
    <param-name>webAppRootKey</param-name>
    <param-value>soba.root</param-value>
  </context-param>

  <!-- session management listener -->
  <listener>
```

```

        <listener-class>org.springframework.security.web.
        session.
            HttpSessionEventPublisher</listener-class>
</listener>
    <session-config>
        <!-- session times out if no activities for 30 minutes -->
        <session-timeout>30</session-timeout>
</session-config>
<!-- Security entry point -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.
        DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- defining the DispatcherServlet -->
<servlet>
    <servlet-name>soba</servlet-name>
    <servlet-class>org.springframework.web.servlet.
        DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>soba</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>soba</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.jpg</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>*.gif</url-pattern>
</servlet-mapping>
<error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/jsp/notfound.
        jsp</location>

```



```

        </error-page>
        <welcome-file-list>
        <welcome-file>
login.jsp
</welcome-file>
    </welcome-file-list>
    <!-- Spring jsp tag lib -->
    <jsp-config>
        <taglib>
            <taglib-uri>/spring</taglib-uri>
            <taglib-location>/WEB-INF/tld/spring-
            form.tld</taglib-location>
        </taglib>
    </jsp-config>

</web-app>

```

Note a few notable entries in this `web.xml` file as follows:

- Note the dispatcher servlet highlighted in boldface. Essentially, this XML element associates the `DispatcherServlet` with a name `soba`, which defines the name of the Web application as in a URL like `https://localhost:8443/soba`. Also note a series of servlet-mapping XML elements following the `DispatcherServlet`. For example, the servlet-mapping element containing the URL pattern of `*.htm` means that any HTTP request ending with `".htm"` will be routed to the `soba DispatcherServlet`.
- Note the definition of a `ContextLoaderListener` along with the definition of a context parameter named `contextCofigLocation`. Note further the three context parameter values of `soba-servlet.xml`, `soba-services.xml` and `soba-security.xml` in the directory of `WEB-INF`. In fact, upon initialization of a `DispatcherServlet`, the Spring MVC framework looks for a file named `<servlet-name>-servlet.xml` in the `WEB-INF` directory and creates beans defined there. So the inclusion of the `soba-servlet.xml` file is purely for the convenience of illustrating the concept of a Web context loader here. The `soba-servlet.xml` file defines handler mappings for a `DispatcherServlet`. The other two XML files will be explained later.

With a good understanding of a `DispatcherServlet` and a `WebApplicationContext`, we are ready to see the actual code snippets and configuration files associated with SOBA to illustrate how each constituent component of the Spring MVC framework, be it a model, a view, or a controller, is coded in Java/Spring in building SOBA. Next, let's explore a complete cycle of creating a new customer/login/account to see what Spring MVC components are used in SOBA. Then we'll walk through the code snippets for those Spring MVC components to get a deeper understanding of how Spring MVC framework is applied to SOBA.

Next, let's introduce the logic flow of SOBA defined in the Spring MVC framework.

15.4.2 Logic Flow of SOBA Defined in Spring MVC Framework

To facilitate the discussion on the Spring MVC framework, Figure 15.14 is presented to show the logic flow with the process of creating a new customer as demonstrated in the preceding section with a series of screenshots from Figure 15.6 through Figure 15.12. The steps involved include:

1. *User Gestures.* In this step, a user opens a Web page to start the interaction with the application. In this use scenario, a user opens up the home page of the application.
2. *Requests Sent to the Spring Dispatcher Servlet.* The gesture of a user is formatted into an HTTP request, which is routed to the Spring dispatching servlet.
3. *Requests Dispatched to the Application Controller.* The Spring dispatching servlet automatically sends the requests to the corresponding application controller, based on the wiring set up in an external XML configuration file that will be discussed shortly.
4. *User Data Validated by Calling a Validator.* It's very rare that a request would be sent directly to the corresponding service without being validated.

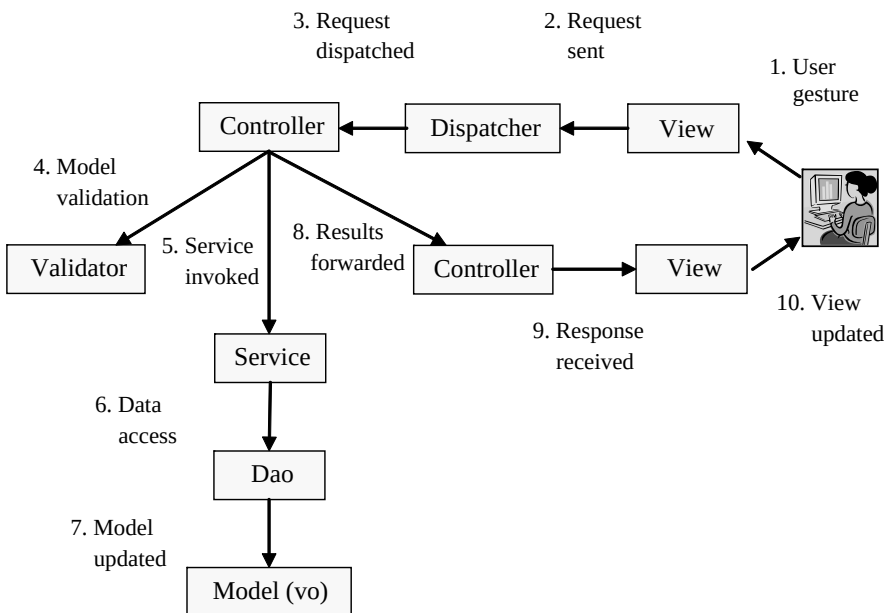


Figure 15.14 SOBA: The logic flow of creating a new customer.

This step must be followed rigorously in a real enterprise application, because the application must distinguish between valid and invalid user input data. This is what a validator would be responsible for.

5. *The Corresponding Service Gets Invoked.* After data validation, the corresponding service is called to execute the tasks on the user's behalf. The service at this step typically makes DAO calls to initiate data-related operations, as is discussed next.
6. *SQL Execution via a DAO.* The methods of a DAO class majorly consist of a standard set of SQL executions such as SELECT, INSERT, UPDATE, DELETE, and so on.
7. *The Data Model Gets Updated.* As the result of the DAO operations as stated in the preceding step, the data model gets updated, which represents the intended operations by the user.
8. *The Results Forwarded to Another Application Controller.* Typically, another application controller is invoked to handle the results returned from the service calls at step 5.
9. *Responses Rendered to a User View.* After this series of logic flow, the results are reformatted and rendered to a user view to be consumed by the user.
10. *The User Gets the Updated View.* At this step, the user finally sees the responses routed back from the application.

As is seen, a `DispatcherServlet` plays the vital role of routing requests from users to the application and responses from the application to the user. Since the `DispatcherServlet` inherits from the `HttpServlet` base class, it is declared in the `web.xml` file of a Web application, as discussed previously. This `web.xml` file illustrated in Listing 15.1 is the deployment descriptor file that describes how to deploy a Web application in a servlet container such as Tomcat. Next, let's start with the entry point to SOBA to trace how the Spring MVC framework governs a typical logic flow as shown in Figure 15.14.

15.4.3 A Web Entry Point Defined in a Spring MVC Web Form

To continue our discussion, note the `<welcome-file-list>` XML element in the previous `web.xml` file shown in Listing 15.1. This element specifies the home page of the application. In this case, the home page of SOBA can be invoked with either `https://localhost:8443/soba` implicitly or `https://localhost:8443/soba/login.jsp` explicitly. Either way, the `login.jsp` file is called. This file provides an entry point for SOBA.

Listing 15.2 shows the contents of the `login.jsp` file. Note the segment of `<a href="<c:url value="createCustomerForm.htm"/>"> Open Now. ,` highlighted in boldface. When clicked, this link would start the process of creating a new customer, as was demonstrated previously. The exact semantics of this segment is defined in the `jsp/jstl/core` tag library, as is indicated by the

first line of this `login.jsp` file. We are less concerned with it now, but we would like to know the exact implication of the part of `createCustomerForm.htm`. We know that since it ends with `.htm`, it would be routed by the `DispatcherServlet`. But what destination will it be directed to by the `DispatcherServlet`? The answer lies in the `soba-servlet.xml` file mentioned previously, as is discussed next.

Listing 15.2 `login.jsp`

```

<%@ include file = "WEB-INF/jsp/include.jsp" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/
core" %>
<html>
<head>
<title>Login</title>
</head>
<%@ include file = "WEB-INF/jsp/banner.jsp" %>
<script language="javascript">
function focusOnUsername () {
    document.loginForm.j_username.focus();
}
</script>
<body onLoad="focusOnUsername ()"><center>

<table><tr>
<td>Prospective Customers: <i>Don't have an account? </i>
</td>
<td><a href="<c:url value="createCustomerForm.htm"/>">
Open Now. </a></td><tr>
<td>Established Customers: <i>Don't have a user ID or
password? </i></td>
<td><a href="<c:url value="createLoginUserForm.htm"/>">
Register </a></td></tr>
</tr></table>
<hr>
    <br><br>

<form name="loginForm" method="POST" action="<c:url value=
"/j_spring_security_check"/>">
<div align="center">
<table align="center" width="300" border="7" CELLPADDING=
"7" CELLSPACING="10" BGCOLOR="#C6EFF7">
<th colspan="2" bgcolor="#00184A"><FONT COLOR="#FFFFFF">
Existing User Login </FONT></th>
<tr>
    <td>Username: </td>
    <td><input type="text" name="j_username"/></td>
</tr>

```

```
|  |  |
| --- | --- |
| Password: </td>  <input type="password" name="j_password"/></td> | |
| <B></B></td><td> <select name="signInRole"> <option value="customer" selected> An Established Customer </option> <option value="rep"> A Rep</option> <option value="admin"> A System Admin</option> </select> </td> </tr> |  |  | | --- | --- | | <input type="submit" value="Login"/> <input type="reset" value="Reset"/> </td> </tr> </table> </form> </div> </center> <%@ include file = "WEB-INF/jsp/showLoadTime.jsp" %> </body> </html> | | |

```

15.4.4 Handler Mapping

To understand handler mapping, we need to explore the `soba-servlet.xml` file. The content of the `soba-servlet.xml` file is shown in Listing 15.3 below. First of all, we see Spring beans defined in each `<bean ..> </bean>` XML element. By default, the `DispatcherServlet` uses `BeanNameUrlHandlerMapping` as its default handler mapping. Thus, a URL pattern like `manageTx.htm` will be mapped to the corresponding class `ManageTxController`. However, this can be simplified since Spring 2.5 by using annotation based configuration and auto-detection. Note the lines of `context:component-scan base-package=<...>` in the `soba-servlet.xml` file. These lines enable all annotated Spring beans to be auto-detected without having to specify the URL mapping in an XML configuration file like `soba-servlet.xml`. This is the case with the URL and the handler that it is mapped to, as is discussed next.

Listing 15.3 soba-servlet.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance"
       xmlns:context="http://www.springframework.
       org/ schema/context"
       xsi:schemaLocation="http://www.springframework.
       org/schema/beans
       http://www.springframework.org/schema/beans/
       spring-beans-3.0.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/
       spring-context-3.0.xsd">

    <!-- the application context definition for the soba
    DispatcherServlet -->
    <context:component-scan base-package="com.perfmath.
    odps.soba.web"/>
    <context:component-scan base-package="com.perfmath.
    odps.soba.model"/>
    <context:component-scan base-package="com.perfmath.
    odps.soba.service"/>
    <context:component-scan base-package="com.perfmath.
    odps.soba.restservice"/>
    <context:annotation-config/>

    <bean id="myAuthenticationManager"
          class="com.perfmath.odps.soba.util.
          MyAuthenticationManager">

    </bean>
    <bean id="messageSource"
          class="org.springframework.context.support.
          ResourceBundleMessageSource">
        <property name="basename" value="messages"/>
    </bean>

    <bean name="/customerList.htm" class="com.perfmath.odps.
    soba.web.CustomerController">
        <property name="customerManager"
          ref="customer-Manager"/>
    </bean>
    <bean name="/accountList.htm" class="com.perfmath.odps.
    soba.web.AccountController">
        <property name="accountManager"
          ref="account-Manager"/>
    </bean>

```

```

<bean name="/transactionList.htm" class="com.perfmath.
odps.soba.web.TxController">
    <property name="txManager" ref="txManager"/>
</bean>

<bean name="/manageTx.htm" class="com.perfmath.odps.
soba.web.ManageTxController">
    <property name="aclTxManager" ref="aclTxManager"/>
</bean>

<bean name="/reverseTx.htm" class="com.perfmath.odps.
soba.web.ReverseTxController">
    <property name="aclTxManager" ref="aclTxManager"/>
</bean>
<bean name="/disputeTx.htm" class="com.perfmath.odps.
soba.web.DisputeTxController">
    <property name="aclTxManager" ref="aclTxManager"/>
</bean>
<!-- spring 3 restful begin -->
<bean id="jaxbMarshaller" class="org.springframework.
oxm.jaxb.Jaxb2Marshaller">
    <property name="classesToBeBound">
        <list>
            <value>com.perfmath.odps.soba.model.vo.
Transaction</value>
        </list>
    </property>
</bean>

<bean id="restTxList"
    class="org.springframework.web.servlet.
view.xml.MarshallingView">
    <constructor-arg ref="jaxbMarshaller"/>
</bean>

<bean class="org.springframework.web.servlet.view.
ContentNegotiatingViewResolver">
    <property name="mediaTypes">
        <map>
            <entry key="html" value="text/html"/>
            <entry key="xml" value="application/xml"/>
            <entry key="json" value="application/json"/>
        </map>
    </property>
    <property name="viewResolvers">
        <list>
            <bean id="viewResolver"
                class="org.springframework.web.servlet.
view.UrlBasedViewResolver">

```

```

        <property name="viewClass"
            value="org.springframework.web.
                servlet.view.JstlView"/>
        <property name="prefix" value="/WEB-INF/
            jsp/" />
        <property name="suffix" value=".jsp"/>
    </bean>
    <bean class="org.springframework.web.servlet.view.
        BeanNameViewResolver"/>
    </list>
</property>

<property name="defaultViews">
    <list>
    <bean class="org.springframework.web.servlet.view.json.
        MappingJacksonJsonView">
        <property name="prefixJson" value="false"/>
    </bean>
    </list>
    </property>
</bean>

</beans>

```

Now let's come back to `login.jsp` file and note again the segment of “`<a href="<c:url value="createCustomerForm.htm"/>"> Open Now . ,`” highlighted in boldface. The URL pattern `createCustomerForm.htm` will be mapped to `CreateCustomerFormController` by default if it's not specified explicitly in the configuration file. This is possible only if the Spring bean `CreateCustomerFormController.java` is annotated properly, as is discussed next.

15.4.5 Implementing Spring Controllers

Controllers or handlers are configured externally and so are view resolutions, providing developers with options for building flexible, modularized applications. In particular, since Spring 2.5, the default handler is based on the `@Controller` and `@RequestMapping` annotations specified in the Java code implementing the handler. Annotation has become more and more popular since Java 5, providing an extra dimension for flexibility. Another benefit of the annotation feature is to allow building RESTful Web sites and applications painlessly, thanks to the method argument level annotation through the `@PathVariable` annotation and other features. We'll see such examples with SOBA later.

To see how Spring Controller is implemented with the help of Spring's annotation feature, Listing 15.4 shows the actual code of the controller class of `CreateCustomerFormController.java` defined in the package of `com.perfmath.odps.soba.web`.

Listing 15.4 CreateCustomerFormController.java

```

package com.perfmath.odps.soba.web;

import java.sql.Timestamp;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.bind.support.SessionStatus;

import com.perfmath.odps.soba.model.vo.Customer;
import com.perfmath.odps.soba.service.CustomerManager;
import com.perfmath.odps.soba.service.CreateCustomerValidator;
import com.perfmath.odps.soba.util.RandomID;

@Controller
@RequestMapping("/createCustomerForm")
@SessionAttributes("customer")
public class CreateCustomerFormController {
    private CreateCustomerValidator validator;
    private CustomerManager customerManager;

    @Autowired
    public CreateCustomerFormController(CustomerManager
        customerManager,
        CreateCustomerValidator validator) {
        this.customerManager = customerManager;
        this.validator = validator;
    }

    @RequestMapping(method = RequestMethod.GET)
    public String setupForm(
        @RequestParam(required = false, value =
            "username") String username,
        Model model) {
        Customer customer = new Customer();
        model.addAttribute("customer", customer);
        return "createCustomerForm";
    }
}

```

```

@RequestMapping(method = RequestMethod.POST)
public String submitForm(
    @ModelAttribute("customer") Customer
    customer,
    BindingResult result, SessionStatus
    status) {
    validator.validate(customer, result);

    if (result.hasErrors()) {
        return "createCustomerForm";
    } else {
        customerManager.createCustomer
        (customer);
        status.setComplete();
        return "redirect:createCustomer
        Success/" + customer.getCustomerId();
    }
}
}

```

By examining the above `CreateCustomerFormController.java` file, we notice the following annotations:

- **@Controller.** This annotation indicates the annotated class serves the role of a controller. In this case, the controller class does not have to extend any controller base class or reference the Servlet API. We can also say that the `@Controller` annotation acts as a stereotype for the annotated class, indicating its role (in UML vocabulary, a *stereotype* is an extension mechanism for defining a new kind of model element based on an existing model element. It is expressed by placing its name as a string around a pair of angle brackets or *guillemets* in French, for example, `<<Stereotype>>`. So a class with the stereotype `<<Controller>>` is read as “a class of the Controller stereotype.” The particular characteristics a Controller class must have are defined when the stereotype is defined. Also note in Java we use `@` instead of *guillemets* to express stereotypes). The annotated beans can be defined explicitly in a configuration file using the URL mapping mechanism. However, they can be more conveniently auto-detected or scanned if it belongs to one of those packages specified in the `<context:component-scan base-package=<...>` XML element. In particular, the controller `CreateCustomerFormController` in the package of `com.perfmath.odps.soba.web` is auto-scanned when the application starts up.
- **@RequestMapping.** This mapping is used to map URLs onto an entire class or a particular handler method. Typically, the class-level annotation maps a specific request path or path pattern onto a form controller, for example, the URL `/createCustomerForm` is mapped to the form controller of

CreateCustomerFormController. We also see RequestMappings associated with HTTP GET and POST methods in Listing 15.4.

- *@SessionAttributes*. This annotation declares session attributes used by a specific handler. It typically lists the names of model attributes that should be maintained in the session, serving as form-backing beans between subsequent requests.
- *@Autowired*. This annotation autowires the class with its dependent classes. For example, the class CreateCustomerFormController depends on two classes: CustomerManager and CreateCustomerValidator. In this case, it's equivalent to the `property` element of a bean definition explicitly specified in its associated configuration file.
- *@RequestParam*. This annotation binds the annotated parameter to the corresponding HTTP request parameter if it exists.
- *@ModelAttribute*. This annotation provides a link to data in the model. When used with the `submitForm` method of a controller, this annotation binds the specified model attribute to the parameter following it. This is how the controller gets a reference to the data entered in the form.
- *@PathVariable*. This annotation binds a method parameter with the value of a URI template variable. We'll see such examples with the SOBA classes that implement RESTful Web services later.

Note that the form controller CreateCustomerFormController has two methods: `setupForm` and `submitForm`. When a URL that contains the destination to this form controller as embedded in the `login.jsp` file is clicked, control is routed to the `DispatcherServlet`, which routes control to this form controller based on the URL mapping it knows about. Then the `setupForm` method of this form controller is invoked first. This is where you can prepopulate some of the entries of the form before the control is turned over to the form, which is `createCustomerForm.jsp` as specified in the `return` statement of the `setupForm` method.

After a user enters all required entries on the form and clicks the `Submit` button, the control is returned to the form controller, and the validator is invoked to validate the data entered onto the form. This is another point of time that you can decide how you want to set some of the entries on the form and how you want to validate the data on the form (in this sample implementation, validation logic is only for illustrative purposes, which should be beefed up significantly in a real application). See Listing 15.5 for the implementation of the CreateCustomerFormValidator class.

Listing 15.5 CreateCustomerFormValidator.java

```
package com.perfmath.odps.soba.service;

import java.sql.Timestamp;
import java.util.Calendar;
import java.util.Date;
```

```

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import org.springframework.stereotype.Component;

import com.perfmath.odps.soba.model.vo.Customer;
import com.perfmath.odps.soba.util.RandomID;

import org.springframework.security.core.context.
SecurityContextHolder;
import org.springframework.security.core.userdetails.*;
@Component
public class CreateCustomerValidator implements Validator {

    public boolean supports(Class clazz) {
        return Customer.class.isAssignableFrom(clazz);
    }

    public void validate(Object target, Errors errors) {
        Object principal = SecurityContextHolder.
            getContext().getAuthentication().getPrincipal();
        if (principal instanceof UserDetails) {
            UserDetails ud = (UserDetails)principal;
            String username = ud.getUsername();
            System.out.println ("current user name:" + username);
            if (ud.isEnabled()) {
                System.out.println ("{current user is enabled:");
            } else {
                System.out.println ("{current user is not
                enabled:");
            }
        } else {
            String username = principal.toString();
            System.out.println ("current user details:" +
            username);
        }
        ValidationUtils.rejectIfEmptyOrWhitespace(errors,
        "firstName",
        "required.firstName", "firstName is required.");
        ValidationUtils.rejectIfEmpty(errors, "lastName",
        "required.lastName", "lastName is required.");
        ValidationUtils.rejectIfEmpty(errors, "phone",
        "required.phone", "phone is required.");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors,
        "address",
        "required.address", "address is required.");
        ValidationUtils.rejectIfEmpty(errors, "city",
        "required.city", "city is required.");
    }
}

```

```

ValidationUtils.rejectIfEmpty(errors, "state",
    "required.state", "state is required.");

Customer customer = (Customer) target;
    customer.setCustomerId( (new RandomID(9)).
        getId());
    customer.setStatus(0);
    customer.setCreateDate(new Timestamp
        (System.currentTimeMillis()));

    String state = customer.getState();

    if (state.length() != 2) {
        errors.reject("invalid.stateNameLength", "State name
            must be two letters.");
    }

}
}

```

If the form data validation is passed, the `createCustomer` service is called, a new customer would be created if everything goes well, and the control is turned over to the `createCustomerSuccess.jsp` file that we'll take a look after we look at the `createCustomerForm.jsp` file next.

15.4.6 A Typical View Defined in a Spring MVC Web Form

The `createCustomerForm.jsp` file shown in Listing 15.6 below illustrates a typical view defined in a Spring MVC Web form (we consider a `.jsp` file a Spring MVC Web form if it defines a form with the Spring `jsp` tag library used). Let's go over those lines highlighted in boldface.

Listing 15.6 `createCustomerForm.jsp`

```

<%@ include file="/WEB-INF/jsp/include.jsp" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/
tags/form" %>

<html>
<head>
</head>
<%@ include file = "banner.jsp" %>
    <style>
        .error { color: red; }
    </style>
</head>

```

```

<body><center>
<h1><fmt:message key="createcustomer.heading" /></h1>
<div align = "center">
<bform:form align="center" method="post" commandName=" customer">

<table align = "center" width="600" bgcolor="#94D6E7"
border="3" cellspacing="10" cellpadding="2">

    <tr>
        <td align="right" width="100">First Name:</td>
        <td width="100">
            <bform:input align="center" path="firstName" />
        </td>
        <td width="400">
            <form:errors path="firstName" cssClass="error" />
        </td>
    </tr>
    <tr>
        <td align="right" width="100">Last Name:</td>
        <td width="100">
            <form:input path="lastName" />
        </td>
        <td width="400">
            <form:errors path="lastName" cssClass="error" />
        </td>
    </tr>
    <tr>
        <td align="right" width="100">Phone:</td>
        <td width="100">
            <form:input path="phone" />
        </td>
        <td width="400">
            <form:errors path="phone" cssClass="error" />
        </td>
    </tr>
    <tr>
        <td align="right" width="100">Address:</td>
        <td width="100">
            <form:input path="address" />
        </td>
        <td width="400">
            <form:errors path="address" cssClass="error" />
        </td>
    </tr>
    <tr>
        <td align="right" width="100">City:</td>
        <td width="100">
            <form:input path="city" />
        </td>

```

```

        <td width="400">
            <form:errors path="city" cssClass="error"/>
        </td>
    </tr>
    <tr>
        <td align="right" width="100">State:</td>
        <td width="100">
            <form:input path="state"/>
        </td>
        <td width="400">
            <form:errors path="state" cssClass="error"/>
        </td>
    </tr>
    <tr>
        <td align="right" width="100">Zipcode:</td>
        <td width="100">
            <form:input path="zipcode"/>
        </td>
        <td width="400">
            <form:errors path="zipcode" cssClass="error"/>
        </td>
    </tr>
    <tr align="center">
        <td align="right" width="100">Email:</td>
        <td width="100">
            <form:input align="center" path="email"/>
        </td>
        <td width="400">
            <form:errors path="email" cssClass="error"/>
        </td>
    </tr>
</table>

<br>
<input type="submit" align="center" value="Submit">
</form:form>
</div>
<%@ include file = "showLoadTime.jsp" %>
</body>
</center>
</html>

```

The first line defines an `include.jsp` file, which contains all common needs shared among all `jsp` files. The content of the `include.jsp` file is shown as follows. Note that the first line specifies that page session is not maintained, which is a common `jsp` performance and scalability practice. The next four lines specify the various `jsp` tag libraries to be used. The last line defines a `JAVA` statement that saves the begin time of

the `jsp` file. When used with the line containing `showLoadTime.jsp` at the bottom of the `createCustomerForm.jsp`, the total elapsed time associated with this form can be timed and displayed to the user. Since both the begin and end timing calls are made on the server, the measured elapsed time is the time spent on the server only without including network latency between the server and the client.

```
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="security" uri="http://www.springframework.
org/security/tags" %>
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<% long beginPageLoadTime = System.currentTimeMillis(); %>
```

Next, note the line containing `fmt:message` in Listing 15.6. This provides an option for specifying text output defined in an external file. For SOBA, this file is named `messages.properties` and stored at the root class path of `/WEB-INF/classes`. The content of this file is shown below.

```
title=SOBA (Safe Online Banking Application)
heading=SOBA :: Safe Online Banking Application
greeting=Greetings, it is now
createcustomer.heading=SOBA :: create a new customer
createloginuser.heading=SOBA :: create a new login user
createaccount.heading=SOBA :: create a new account
createtx.heading=SOBA :: Post a transaction to an account
required=Entry required.
typeMismatch=Invalid data.
```

Next, note the line containing `<form:form...>`. This line defines that the HTTP method to be used to send the form to the `CreateCustomerFormController` would be POST, and that the command object to be invoked would be a customer object, which is defined in the `Customer.java` file in the VO package for the definition of the Customer class as shown in Listing 15.7 below.

Listing 15.7 Customer.java

```
package com.perfmath.odps.soba.model.vo;

import java.io.Serializable;
import java.sql.Timestamp;

public class Customer implements Serializable {

    private String customerId;
    private String firstName;
```



```

private String lastName;
private String phone;
private String address;
private String city;
private String state;
private String zipcode;
private String email;
private int status;
private Timestamp createDate;

// getters/setters are omitted to save space
.....
public String toString() {
    StringBuffer buffer = new StringBuffer();
    buffer.append(" customerId: " + customerId + ";");
    buffer.append(" firstName: " + firstName);
    buffer.append(" lastName: " + lastName);
    buffer.append(" phone: " + phone);
    buffer.append(" address: " + address);
    buffer.append(" city: " + city);
    buffer.append(" state: " + state);
    buffer.append(" zipcode: " + zipcode);
    buffer.append(" email: " + email);
    buffer.append(" status: " + status);
    buffer.append(" createDate: " + createDate);
    return buffer.toString();
}
}

```

Then the form contains many lines like, for example, `<form:input align="center" path="firstName">`. There is a one-to-one corresponding relationship between a path defined on the form and the property of the class to be targeted. This is how a form and a VO class gets associated with each other. Also note that for each line of `<form:input ...>` there is a corresponding line of `<form:errors ...>`, which associates the entry with the errors found during validation.

Finally, the line containing “Submit” defines the exit of this jsp file, which returns the control to the form’s form controller `CreateCustomerFormController`. After the transaction of creating a new customer is completed successfully, the control is returned to the `CreateCustomerSuccessController`, which is discussed next.

15.4.7 A Typical Form Success Controller and its Resultant View

Listing 15.8 shows a typical form success controller associated with the transaction of creating a new customer. It is as simple as just returning a `ModelAndView` object with the return URL of the `createCustomerSuccess` form, which is shown in Listing 15.9. The `createCustomerSuccessForm.jsp` file displays a message

showing that the transaction is completed successfully. It then waits for the user to initiate the next transaction with a link embedded that is mapped to another controller using the same mapping mechanism discussed previously.

Listing 15.8 CreateCustomerSuccessController.java

```
package com.perfmath.odps.soba.web;

import java.util.HashMap;
import java.util.Map;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class CreateCustomerSuccessController {
    @RequestMapping(value="/createCustomerSuccess/{customerId}",
        method=RequestMethod.GET)
    public ModelAndView createCustomerSuccess (@PathVariable
        ("customerId") String customerId) {
        Map<String, Object> myModel=new HashMap<String, Object>();
        myModel.put("customerId", customerId);
        return new ModelAndView("createCustomerSuccess",
            "model", myModel);
    }
}
```

Listing 15.9 createCustomerSuccessForm.jsp

```
<%@ include file="/WEB-INF/jsp/include.jsp" %>
<%@ taglib prefix="form" uri="http://www.
springframework.org/tags/form" %>
<html>
<head>
<%@ include file = "banner.jsp" %>
<title>Create Customer Success</title>
</head>

<body> <center>
Your customer ID<c:out value="${model.customerId}"/>
has been created successfully.
```

```

<br>
    <br> Use your customer ID to
    <a href="<c:url value="/createLoginUserForm.htm?
    customerId=${model.customerId}"/>">
        create</a> your login ID for banking online.
    <br> <br>
    </center>
</body>
</html>

```

15.4.8 POJOs Referenced in the CreateCustomerFormController

It might be interesting at this point to review the POJOs used by the CreateCustomerFormController described previously. These POJOs include:

- CustomerManager.java and SimpleCustomerManager.java as shown in Listings 15.10 (a) and 15.10 (b), respectively.
- CustomerDao.java and JdbcCustomerDao.java as shown in Listings 15.11 (a) and 15.11 (b), respectively.
- Customer.java, which is shown previously in Listing 15.7.

Listing 15.10(a) CustomerManager.java

```

package com.perfmath.odps.soba.service;

import java.io.Serializable;
import java.util.List;
import com.perfmath.odps.soba.model.vo.Customer;

public interface CustomerManager extends Serializable{
    public void createCustomer(Customer customer);
    public List<Customer> getCustomers();
}

```

Listing 15.10(b) SimpleCustomerManager.java

```

package com.perfmath.odps.soba.service;

import java.util.List;
import com.perfmath.odps.soba.model.dao.CustomerDao;
import com.perfmath.odps.soba.model.vo.Customer;

```

```

public class SimpleCustomerManager implements CustomerManager {

    private CustomerDao customerDao;

    public List<Customer> getCustomers() {
        return customerDao.getCustomerList();
    }

    public void createCustomer(Customer customer) {
        customerDao.insert(customer);
    }

    public void setCustomerDao(CustomerDao customerDao) {
        this.customerDao = customerDao;
    }
}

```

Listing 15.11(a) CustomerDao.java

```

package com.perfmath.odps.soba.model.dao;

import java.util.List;
import java.util.Map;
import com.perfmath.odps.soba.model.vo.Customer;

public interface CustomerDao {
    public List<Customer> getCustomerList();
    public void insert(Customer customer);
    public void update(Customer customer);
    public void delete(Customer customer);
    public Customer findByCustomerId(String customerId);
    public void insertBatch(List<Customer> customers);
    public List<Map<String, Object>> findAll();
    public String getEmail(String customerId);
    public int countAll();
}

```

Listing 15.11(b) JdbcCustomerDao.java

```

package com.perfmath.odps.soba.model.dao;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.sql.Timestamp;

```

```

import org.springframework.jdbc.core.namedparam.
BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.
SqlParameterSource;
import org.springframework.jdbc.core.simple.
ParameterizedBeanPropertyRowMapper;
import org.springframework.jdbc.core.simple.
ParameterizedRowMapper;
import org.springframework.jdbc.core.simple.
SimpleJdbcDaoSupport;

import com.perfmath.odps.soba.model.vo.Customer;

public class JdbcCustomerDao extends SimpleJdbcDaoSupport im-
plements CustomerDao {
    public List<Customer> getCustomerList() {
        String sql = "SELECT CUSTOMER_ID, FIRST_NAME,
LAST_NAME, PHONE,
        ADDRESS, CITY, STATE, "
            + " ZIPCODE, EMAIL, STATUS, CREATE_DATE
FROM CUSTOMER";
        List<Customer> customers = getSimpleJdbcTemplate().
query(sql, new CustomerMapper());
        return customers;
    }

    public void insert(Customer customer) {
        String sql = "INSERT INTO CUSTOMER (CUSTOMER_ID,
FIRST_NAME,
        LAST_NAME, PHONE, ADDRESS, CITY, STATE, "
            + " ZIPCODE, EMAIL, STATUS, CREATE_DATE) "
            + "VALUES (:customerId, :firstName, :lastName, :phone,
:address, :city, "
            + ":state, :zipcode, :email, :status, :createDate)";

        SqlParameterSource parameterSource =
            new BeanPropertySqlParameterSource(customer);

        getSimpleJdbcTemplate().update(sql, parameterSource);
    }

    public void insertBatch(List<Customer> customers) {
        String sql = "INSERT INTO Customer (CUSTOMER_ID,
FIRST_NAME, LAST_NAME,
        PHONE, ADDRESS, CITY, STATE, "
            + " ZIPCODE, EMAIL, STATUS, CREATE_DATE) "
            + "VALUES (:customerID, :firstName, :lastName, :phone,

```

```

        :address, :city, "
        + ":state, :zipcode, :email, :status, :createDate)";

List<SqlParameterSource> parameters =
    new ArrayList<SqlParameterSource>();
for (Customer Customer : customers) {
    parameters.add(new BeanPropertySqlParameter
        Source-(Customer));
}
getSimpleJdbcTemplate().batchUpdate(sql,
    parameters.toArray(new SqlParameterSource[0]));
}

public Customer findByCustomerID(String customerID) {
    String sql = "SELECT * FROM Customer WHERE CUSTOMER_ID = ?";

    Customer Customer = getSimpleJdbcTemplate().
        queryForObject(sql,
            Customer.class, customerID);
    return Customer;
}

public void update(Customer customer) {}

public void delete(Customer customer) {}

public List<Map<String, Object>> findAll() {
    String sql = "SELECT * FROM CUSTOMER";

    List<Map<String, Object>> customers = getSimpleJdbc
        Template().queryForList(sql, Customer.class);
    return customers;
}

public String getEmail(String customerID) {
    String sql = "SELECT EMAIL FROM CUSTOMER WHERE
        CUSTOMER_ID = ?";

    String email = getSimpleJdbcTemplate().
        queryForObject(sql,
            String.class, customerID);
    return email;
}

public int countAll() {
    String sql = "SELECT COUNT(*) FROM CUSTOMER";

```

```

        int count = getJdbcTemplate().queryForInt(sql);
        return count;
    }

    private static class CustomerMapper implements
    ParameterizedRowMapper<Customer> {
        public Customer mapRow(ResultSet rs, int rowNum)
        throws SQLException {
            Customer cust = new Customer();
            cust.setCustomerId(rs.getString("CUSTOMER_ID"));
            cust.setFirstName(rs.getString("FIRST_NAME"));
            cust.setLastName(rs.getString("LAST_NAME"));
            cust.setPhone(rs.getString("PHONE"));
            cust.setAddress(rs.getString("ADDRESS"));
            cust.setCity(rs.getString("CITY"));
            cust.setState(rs.getString("STATE"));
            cust.setZipcode(rs.getString("ZIPCODE"));
            cust.setEmail(rs.getString("EMAIL"));
            cust.setStatus(rs.getInt("STATUS"));
            cust.setCreateDate(rs.getTimestamp("CREATE_DATE"));
            return cust;
        }
    }
}

```

Since these POJOs are ordinary Java objects and what they do are fairly self-explanatory, a more detailed coverage is omitted here.

Data access can be implemented either in JDBC or in some object-relational mapping (ORM) frameworks such as Hibernate. To demonstrate how Spring MVC framework works with an ORM framework, we discuss next on how the bill payment function of SOBA is implemented with Hibernate in place of JDBC which is used with all other functions of SOBA.

15.5 HIBERNATE OBJECT-RELATIONAL MAPPING (ORM) APPLIED TO SOBA

First of all, there is no barrier with using an ORM framework like Hibernate with the Spring MVC framework, thanks to the architectural pluggability of Spring. Secondly, in general, an ORM framework is favored over JDBC for accessing data stored in a database. Let's first explore what benefits an ORM framework like Hibernate provides over JDBC.

15.5.1 Benefits of Using Hibernate

As one of the most popular ORM frameworks, Hibernate provides the following benefits:

- Hibernate works by persisting data according to the mapping metadata defined in a mapping file such as the `BillPayment.hbm.xml` file shown in Listing 15.12 below. A mapping file defines mappings between a Java class and a database table by specifying how the attributes of the Java class and the columns of the table map to each other. This provides a natural bridge between the object-oriented programming model and the relational database model.
- Hibernate generates SQL statements at runtime, and therefore its implementation is database vendor neutral. This is especially important if your application has to support multiple database platforms from all major vendors.
- Hibernate goes beyond the basic ORM functionality. It supports additional features like caching, cascading, and lazy loading, which may help enhance the performance and scalability of your application.
- Hibernate defines a powerful query language of its own that is called HDL (Hibernate Query Language). It's very easy to write queries with the HDL.

Listing 15.12 `BillPayment.hbm.xml`

```
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.perfmath.odps.soba.model.vo">
  <class name="BillPayment" table="BILL_PAYMENT">
    <id name="id" type="long" column="ID">
      <generator class="assigned">
        <!-- param name="sequence">BILL_PYMNT_SEQ </param -->
      </generator>
    </id>
    <property name="fromAccount" type="string">
      <column name="FROM_ACCOUNT" length="9" not-null="true"/>
    </property>
    <property name="accountId" type="string">
      <column name="ACCOUNT_ID" length="9" not-null="true"/>
    </property>
    <property name="description" type="string">
      <column name="DESCRIPTION" length="500" not-null="true"/>
    </property>
    <property name="biller" type="string">
      <column name="BILLER" length="25" not-null="true"/>
    </property>
```



```

        <property name="address" type="string">
            <column name="ADDRESS" length="50" not-null="true"/>
        </property>
        <property name="city" type="string">
            <column name="CITY" length="25" not-null="true"/>
        </property>
        <property name="state" type="string">
            <column name="STATE" length="2" not-null="true"/>
        </property>
        <property name="zipcode" type="string">
            <column name="ZIPCODE" length="10" not-null="true"/>
        </property>
        <property name="status" type="string" column="STATUS"/>
        <property name="amount" type="double" column="AMOUNT"/>
        <property name="scheduleDate" type="date" column="SCHEDULE_
DATE"/>
        <property name="sendDate" type="date" column="SEND_DATE"/>
    </class>
</hibernate-mapping>

```

Next, let's see how metadata mapping works with Hibernate.

15.5.2 Metadata Mapping with Hibernate

Metadata mapping with Hibernate is straightforward. For example, with the `BillPayment.java` class shown in Listing 15.13, metadata mapping is achieved with a mapping file as shown previously in Listing 15.12. Note that this mapping file needs to be placed in the class target directory of the Java class it is mapped to, or in this case, together with the `BillPayment.class` file rather than the source file.

However, one needs to pay attention to how the ID field is mapped. First of all, this field is mandatory in the sense that both the Java class and the database table must have it. Secondly, its type must be *long*. Finally, note how a generator is defined with a property of *class* = "...". The values of this property can have include: *assigned*, *increment*, *identity*, *sequence*, *hilo*, *seqhilo*, *uuid*, *native*, *select*, *foreign*, and so on. Detailed discussion on what each setting means is beyond the scope of this text, except that the setting of *assigned* chosen in SOBA means that it's assigned by the application.

Listing 15.13 `BillPayment.java` (to save space, all getter/setter methods are omitted)

```

package com.perfmath.odps.soba.model.vo;

import java.io.Serializable;
import java.sql.Timestamp;

```

```

public class BillPayment implements Serializable {
    private Long id = null;
    private String accountId;
    private String description;
    private double amount;
    private String fromAccount;
    private String biller;
    private String address;
    private String city;
    private String state;
    private String zipcode;
    private String status;
    private Timestamp scheduleDate;
    private Timestamp sendDate;

    public BillPayment() {

    }

    / all getters and setters are omitted to save space
    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append(" Id: " + id + " ");
        buffer.append(" accountId: " + accountId + " ");
        buffer.append(" description: " + description);
        buffer.append(" amount: " + amount);
        buffer.append(" fromAccount: " + fromAccount);
        buffer.append(" biller: " + biller);
        buffer.append(" address: " + address);
        buffer.append(" city: " + city);
        buffer.append(" state: " + state);
        buffer.append(" zipcode: " + zipcode);
        buffer.append(" status: " + status);
        buffer.append(" scheduleDate: " + scheduleDate);
        buffer.append(" sendDate: " + sendDate);
        return buffer.toString();
    }
}

```

Next, let's take a look at how Hibernate is configured to communicate with Oracle.

15.5.3 Configuring Hibernate to Work with Oracle

How Hibernate communicates with Oracle is defined in the `hibernate.cfg.xml` file as shown in Listing 15.14 below (note the last line referring to the metadata mapping file described in the previous section. Also note that this file needs to be placed in the `WEB-INF/classes` directory). Given what we have covered in

this text so far about Oracle, everything should be clear except the last few lines defined as followed:

- *The Property `hibernate.show_sql= true`.* This entry specifies that all SQLs generated by Hibernate at runtime will be output to the console. This is a convenient feature for debugging purposes. However, it should be set to `false` in production due to performance concerns.
- *The Property `hibernate.hbm2ddl.auto= update`.* This entry specifies the intention for creating the database schema on deploy if it doesn't exist. It has many other possible settings and which one is most appropriate should be determined with the tests conducted with your application.
- *The Property `current_session_context_class = thread`.* This entry specifies the scope of the current session. Valid values include `jta`, `thread`, `managed`, and so on. The setting of `thread` specified here limits the session context to the thread level.

Listing 15.14 hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">oracle.jdbc.
      driver.OracleDriver</property>
    <property name="connection.url">jdbc:oracle:thin:
      @p6620f:1521:Ora11GR2</property>
    <property name="connection.username">OBAdmin</property>
    <property name="connection.password">OB#Admin</property>
    <property name="hibernate.dialect">org.hibernate.dialect.
      OracleDialect</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="current_session_context_class">thread-
      </property>
    <mapping resource="com/perfmath/odps/soba/model/vo/
      BillPayment.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Since this is not a text about Hibernate, please consult more in-depth documentations to learn more about various Hibernate settings. Next, we discuss how a Hibernate DAO class is implemented.

15.5.4 Hibernate DAO

Listings 15.15 (a) and (b) illustrate how a Hibernate DAO is implemented with the `BillPayment` domain object. Note the use of the `SessionFactory` interface as well as explicit commit and rollback with a transaction. Also note the use of the method of `saveOrUpdate` with a session. This method would perform an INSERT if the row to be inserted does not exist in the database or an UPDATE if the row exists.

Listing 15.15(a) `HibernateBillPaymentDao.java`

```
package com.perfmath.odps.soba.model.dao;

import java.util.List;
import com.perfmath.odps.soba.model.vo.BillPayment;

public interface BillPaymentDao {

    public void store(BillPayment billPayment);
    public void delete(String id);
    public BillPayment findById(String id);
    public List<BillPayment> findAll();
}
```

Listing 15.15(b) `HibernateBillPaymentDao.java`

```
package com.perfmath.odps.soba.model.dao;

import com.perfmath.odps.soba.model.vo.BillPayment;
import com.perfmath.odps.soba.model.dao.BillPaymentDao;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import java.util.List;

public class HibernateBillPaymentDao implements
BillPaymentDao {
    private SessionFactory sessionFactory;

    public HibernateBillPaymentDao() {
        Configuration configuration = new
        Configuration().configure();
```

```

        sessionFactory = configuration.
            buildSessionFactory();
    }
    public void store(BillPayment billPayment) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            session.saveOrUpdate(billPayment);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public void delete(String id) {
        Session session = sessionFactory.openSession();
        Transaction tx = session.getTransaction();
        try {
            tx.begin();
            BillPayment billPayment =
                (BillPayment) sessionFactory
                    .getCurrentSession().
                    get(BillPayment.class, id);
            session.delete(billPayment);
            tx.commit();
        } catch (RuntimeException e) {
            tx.rollback();
            throw e;
        } finally {
            session.close();
        }
    }

    public BillPayment findById(String id) {
        Session session = sessionFactory.openSession();
        try {
            return (BillPayment) sessionFactory.
                getCurrentSession().get(
                    BillPayment.class, id);
        } finally {
            session.close();
        }
    }

    public List<BillPayment> findAll() {
        Session session = sessionFactory.openSession();

```

```
try {  
    Query query = sessionFactory.  
        getCurrentSession().createQuery  
            ("from Bill_Payment");  
    return query.list();  
} finally {  
    session.close();  
}  
}
```

Note that whether a DAO is implemented in JDBC or Hibernate, it's used the same way in a service object. Because of this transparency, we won't show the `BillPaymentManager` class to illustrate how a `HibernateBillPaymentDao` object is invoked in a `BillPaymentManager` object.

An example of Hibernate in action is the Bill Payment feature of SOBA. Figures 15.15 to 15.16 show the sequence of actions taking place with a bill payment cycle. It is summarized as follows:

1. A customer (in this case user1001) logs in and clicks Bill Payment tab. The customer then enters the bill, and clicks Submit. See Figure 15.15 for this step.



The image shows a web application header for SOBA (A Secure Online Banking Application). Below the header, the main heading is "SOBA :: Post a transaction to an account". The form contains the following fields:

Pay From Account:	118978398
Receiver AccountID:	604712383
Description:	bill pay to user99
Amount:	7.99
Biller:	user99
Address:	123 way
City:	any city
State:	ca
Zipcode:	95999

At the bottom right of the form is a "Submit" button.

Figure 15.15 SOBA: A customer logs in and clicks Bill Payment tab. The customer then enters the bill (note that in this simulated case, the customer pays \$7.99 to user99), and clicks Submit.



**A Secure Online
Banking Application®**

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

[Open an Account](#) | [Create a Tx](#) | [Create an ACL Tx](#) | [Bill Payment](#) | [Transfers](#) | [Dispute Transactions](#) | [Messages](#)

Logged in as: user1001

Customer Account Details

Customer ID:	173666039
Account ID:	118976398
Account Name:	my checking
Account Type:	Checking
Account Balance:	1512.01

Account Name	Type	Range (date input format: yyyy-mm-dd)
Premium Checking ▾	Checking ▾	☒ Last 30 days ▾ ☐ From: To: Get History Reset

Date	Type	Description	Debit	Credit	Balance
2011-04-16	bill pay	pay to user99	-7.99		1512.01
2011-04-16	acl tx	Reversed: acl tx test	-29.99		1520.0
2011-04-16	acl tx	Customer disputed (txId = 391117893): acl tx test	0.0		1549.99

Figure 15.16 SOBA: Now the customer sees \$7.99 debited from his account.

- The bill payment is processed successfully. The user clicks the View Account Activity link and verifies that an amount of \$7.99 is posted, as shown in Figure 15.16.

Next, a brief coverage of how RESTful Web services are applied to SOBA is provided.

15.6 RESTFUL WEB SERVICES APPLIED TO SOBA

RESTful Web services refers to a new software architectural style for building software that provides services using the HTTP/HTTps protocols or any other application layer protocols (note that REST doesn't have to use HTTP). The term REST stands for REpresentational State Transfer that originated from Roy Fielding's doctorate dissertation (University of California, Irvine, 2000) titled "Architectural Styles and the Design of Network-based Software Architectures." It's claimed that REST has displaced SOAP and WSDL-based Web services architectures because of its simplicity, resource-oriented advantages. Next, a brief introduction to RESTful Web services is given.

15.6.1 Introduction to RESTful Web Services

The core concept of REST is *resource*. The most representative format of a resource is a *document*. The content of a document may not matter much from an

Table 15.1 Mappings between REST and HTTP

REST	HTTP	Comment
CREATE	POST	Create a resource
READ	GET	Retrieve a resource
UPDATE	PUT	Change the state of a resource
DELETE	DELETE	Remove a resource

architectural point of view, although it certainly matters from an application point of view. What makes REST come into the picture of software design is that a document as a resource is *dynamically* exchanged between a server and a client using the well-understood request/response model. In order for this to happen, a document as a resource must be *addressable* or *identifiable*. How a document is transferred between a client and a server is the main concern of REST. Although REST doesn't have to be exclusively tied to HTTP, it's easiest to explain REST with HTTP, as is described next.

REST calls for four types of operations on resources: CREATE, READ, UPDATE, and DELETE (CRUD), typically from a client to a server. These types of operations happen to be what an HTTP protocol can do well, governed by RFC 2616, which was a Request For Comments (RFC) document on Hypertext Transfer Protocol—HTTP/1.1—authored by Fielding and associates (<http://tools.ietf.org/pdf/rfc2616.pdf>). Table 15.1 maps each of those REST CRUD operations to its corresponding HTTP transfer method. Note that the REST CRUD operations are similar to the SQL operations of CREATE, SELECT, UPDATE, and DELETE. However, REST is beyond just using HTTP or another protocol to achieve those CRUD operations, as is discussed below.

15.6.2 RESTful Constraints

The REST style imposes the following constraints at the system level:

- *Client-Server*. This constraint imposes that clients and servers are separated by a uniform interface. Clients are not concerned with, for example, how data is stored on the server, and servers are not concerned with user state so that servers can be simpler and more scalable. Clients and servers can change independently as long as the interface remains intact.
- *Stateless*. This constraint requires no client context should be stored on the server between requests. However, this is a constraint on the client, not necessarily on the server. Each client request should be self-complete, but the server can be stateful, for example, caching the ID of a client, and so on. This makes client/server communications more reliable if network failures occur, and it also helps scalability.
- *Cacheable*. This constraint means that server responses to client requests are cacheable on the client side, which further increases performance and scalability.

- *Transparency.* A client cannot tell or does not need to know whether a response is directly from the end server or an intermediary server. Intermediary servers help improve system performance and scalability by adopting load balancing and providing shared caches.
- *Uniform Interface.* The uniform interface simplifies and decouples client/server interactions, which allows each part to evolve independently.
- *Code on Demand (Optional).* A server is able to temporarily extend/customize the functionality of a client by transferring logic to it that it can execute.

According to the above constraints, a service is RESTful only if all required constraints are satisfied while leaving code on demand optional. These RESTful constraints are helpful for developing distributed hypermedia systems that can easily collaborate with each other and achieve the desired non-functional requirements such as performance, scalability, simplicity, reliability, portability, maintainability, modifiability, visibility, and so on. Retrospectively, these constraints originated from Fielding as he described in his dissertation as follows:

REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.

Next, let's discuss some of the guiding principles for designing RESTful interfaces.

15.6.3 RESTful Interface Design Principles

The uniform interface is the key part of designing any REST interface of a REST service. The following design principles are recommended:

- *Separating Resource Identifications from Representations.* With Web-based REST systems, this can be achieved by using URIs (Uniform Resource Identifiers—a string of characters used to identify a resource on the Internet) to identify resources while using payload to contain representations. For example, a server does not send its database connection information, but rather, for example, some HTML, XML, or JSON that represents data in its databases.
- *Control of Resources through Representations.* A representation of a resource including any available metadata should be self-complete to allow a client to access the resource on the server, for example, creating or deleting the resource if the client has necessary permission.

- *Self-descriptive Messages.* A message should contain sufficient information on how to process the message. One example is to explicitly indicate the type of the message.
- *Hypermedia as the Engine of the Application State.* The representation returned from the server should provide hypertext links to allow a client to access any related resources.

The above REST interface design principles help the designers of a distributed hypermedia system achieve the goals of: scalability at the system level; simplicity with generic interfaces; flexibility with independent deployment of components or modules; and performance with intermediary components for reducing latency, enforcing security, and encapsulating legacy systems.

Next, we explore how Spring supports developing RESTful Web services.

15.6.4 Spring's Support for RESTful Web Services

A RESTful Web service is a service that uses HTTP protocol to exchange resources in a manner that is fully compliant to the RESTful constraints as discussed in the previous section. A RESTful Web service is also called a RESTful Web API. Spring makes developing RESTful Web APIs easy by providing a mechanism of URI templates, which is a simple way of identifying a resource.

An example URI template may look like

```
http://<host:port>/transaction/{id}
```

In this case, the part in the curly brackets is a variable named *id*. When the variable *id* is filled with a specific value, the above URI template yields an actual URI, for example, a URI that represents a transaction id in SOBA: `http://localhost:8443/soba/transaction/909957218`. As is seen, this is conceptually indeed very simple.

To bind a URI with a method, it's required to use `@RequestMapping` in conjunction with `@PathVariable` with one of the following options:

- `// variable name and parameter name same`

```
@RequestMapping (value = "/tx/{txId}", method =
Request-Method.GET)

Public String getTransaction (@PathVariable ("txId")
String txId, Model model) {

...

}
```

- `// variable name and parameter name different`


```
@RequestMapping (value = "/tx/{txId}", method =
RequestMethod.GET)

Public String getTransaction (@PathVariable ("txId")
String the TxId, Model model) {

...

}
```
- `//multiple variables`


```
@RequestMapping (value = "/account/{accountId}/tx/
{txId}", method = RequestMethod.GET)

Public String getTransaction (@PathVariable
("accountId") String accountId, @PathVariable ("txId")
String txId, Model model) {

.....

}
```

The method parameters annotated with `@PathVariable` can be of any simple type such as `int`, `long`, `Date`, and so on. Spring does type conversion automatically and throws a `TypeMismatchException` if the type mismatches.

Next, let's see some actual REST code on the server side with SOBA.

15.6.5 Server Code

A RESTful Web API is implemented in SOBA for retrieving a transaction based on a transaction ID. It can be invoked internally with a link in a jsp page as follows (note that the transaction ID 791111196 is a real one in SOBA's Oracle database):

```
<a href="/soba/restTx/txId/791111196">791111196</a>
```

When the user clicks the above embedded link, the RESTful Web API `getTransactionById` implemented in the controller `RestTxController` is called. Listing 15.16 below shows how this RESTful Web API is implemented. Note that we did not include the URI `/restTx/txId/{txId}` in the annotation `@RequestMapping` placed immediately prior to this method. By placing the

URI prior to class definition, the embedded URI is mapped to this controller, and the API is mapped with the annotation `@RequestMapping` (method = `RequestMethod.GET`).

Listing 15.16 RestTxController.java

```
package com.perfmath.odps.soba.restfulweb;

import java.io.StringReader;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;
import org.springframework.security.annotation.Secured;
import org.springframework.stereotype.Controller;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.servlet.ModelAndView;

import com.perfmath.odps.soba.model.dao.TransactionDao;
import com.perfmath.odps.soba.model.vo.Transaction;
import com.perfmath.odps.soba.service.CreateTxValidator;
import com.perfmath.odps.soba.service.TxManager;

@Controller
@RequestMapping("/restTx/txid/{transactionId}")
@SessionAttributes("restTx")
public class RestTxController implements TxManager {

    private TransactionDao transactionDao;

    private Jaxb2Marshaller jaxb2Marshaller;

    @Autowired
    public RestTxController(TransactionDao transactionDao,
                            Jaxb2Marshaller jaxb2Marshaller) {
        super();
        this.transactionDao = transactionDao;
        this.jaxb2Marshaller = jaxb2Marshaller;
    }
}
```

```

    public void setJaxb2Marshaller(Jaxb2Marshaller jaxb2Marshaller) {
        this.jaxb2Marshaller = jaxb2Marshaller;
    }

    //private static final String XML_VIEW_NAME = "restTxList";
    private static final String XML_VIEW_NAME = "disputeTx";
    @Secured("ROLE_CUST")
    @RequestMapping(method = RequestMethod.GET)
    public ModelAndView getTransactionById(@PathVariable
    String transactionId) {

String now = (new java.util.Date()).toString();

Map<String, Object> myModel = new HashMap<String, Object>();
myModel.put("now", now);

        Transaction transaction = transactionDao
            .findByTransactionID(transactionId);
myModel.put("transaction", transaction);
return new ModelAndView(XML_VIEW_NAME, "model", myModel);
        //return new ModelAndView(XML_VIEW_NAME,
        "transaction", transaction);
    }
    @RequestMapping(method = RequestMethod.PUT, value =
    "/restTx/txId/{id}")
    public ModelAndView updateTransaction(@RequestBody String
    body) {

        Source source=new StreamSource(new StringReader(body));
        Transaction tx = (Transaction) jaxb2Marshaller.
            unmarshal(source);
        transactionDao.update(tx);
        return new ModelAndView(XML_VIEW_NAME, "object", tx);
    }

    @RequestMapping(method=RequestMethod.POST, value = "/restTx")
    public ModelAndView createTransaction(@RequestBody String
    body) {
        Source source = new StreamSource(new StringReader
        (body));
        System.out.println("Rest create: body =" + body);
        Transaction tx = (Transaction) jaxb2Marshaller.
            unmarshal(source);
        transactionDao.insert(tx);
        return new ModelAndView(XML_VIEW_NAME, "tx", tx);
    }

    public void setTransactionDao(TransactionDao
    transactionDao) {
        this.transactionDao = transactionDao;
    }

```

```

    }

    @Transactional
    @Secured("ROLE_CUST, ROLE_REP")
    public void createTransaction(Transaction tx) {
        transactionDao.insert(tx);
    }

    // @Secured("ROLE_REP")
    @Transactional
    @Secured("ROLE_REP")
    public void deleteTransaction(String txId) {
        transactionDao.delete(txId);
    }

    // @Secured({"ROLE_USER", "ROLE_GUEST",
    "AFTER_ACL_READ"})

    public Transaction findByTransactionID(String txId) {
        return transactionDao.findByTransactionID(txId);
    }

    public List<Transaction> getTransactions() {
        return transactionDao.getTransactionList();
    }

    public List<Transaction> getTransactions(String
customerId) {
        return transactionDao.getTransactionList
(customerId);
    }

    public void updateTransaction(Transaction tx) {
        transactionDao.update(tx);
    }
}

```

Although it's only one RESTful Web API, it's sufficient for demonstrating how Spring supports RESTful Web APIs, as all other RESTful Web APIs can be developed similarly.

How this RESTful Web API can be accessed by an external client is discussed next.

15.6.6 Client Code

Listing 15.17 below shows how the RESTful Web API discussed in the preceding section can be accessed external to the server hosting the application (in this case,

SOBA). It's indeed as simple as promised. The client program uses Apache http client API. Note the following steps involved:

- *Loading SSL Certificate.* Since SOBA is secured with SSL and should be accessed with the secure HTTP protocol (HTTPS), the first step is to set up SSL. Refer to Section 15.1 about how to create an SSL certificate for Tomcat. Also note that if you want to test it out on your machine, you need to replace the hard-coded path to your Tomcat SSL certificate.
- *Setting Security Provider.* The second step is to set up a security provider to get prepared for authentication.
- *Setting Login Credentials.* In this step, make sure proper hostname, HTTPS port, user name, and password are entered.
- *Making the REST Web API Call.* Note the URI used in creating an HttpGet object. Also note the statement `httpget.addHeader ("Accept", "application/json");`. The acronym "json" stands for JavaScript Object Notation. It's a format for representing simple data structures and associative array (map, dictionary, etc.). As an alternative to XML, it is primarily used to transmit data between a Web server and a client.

Listing 15.17 RestWebAPITest.java

```
package com.perfmath.odps.soba.test;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.security.KeyStore;
import java.security.Security;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;

/**
 * This example demonstrates:
 * 1) how to create secure connections with a custom SSL context
 * 2) how to test RESTful web API
 */
public class RestWebAPITest {
```

```

public final static void main(String[] args) throws
Exception {
    DefaultHttpClient httpClient = new DefaultHttpClient();
    // SSL setup begin
    KeyStore trustStore = KeyStore.getInstance(KeyStore.
getDefaultType());
    FileInputStream instream = new FileInputStream(new File
("C:\\Users\\henry\\.keystore"));
    try {
        trustStore.load(instream, "changeit".toCharArray());
    } finally {
        instream.close();
    }

    SSLSocketFactory socketFactory = new SSLSocket
Factory(trustStore);
    Scheme sch = new Scheme("https", socketFactory, 8443);
    httpClient.getConnectionManager().getSchemeRegistry().
register(sch);
    // SSL setup end

    // set security provider
    String secProviderName = "com.sun.crypto.provider.SunJCE";
    java.security.Provider secProvider =
        (java.security.Provider)Class.forName
(secProviderName).newInstance();
    Security.addProvider(secProvider);

    httpClient.getCredentialsProvider().setCredentials(
        new AuthScope("localhost", 8443),
        new UsernamePasswordCredentials("user1001",
"user1001"));
    // issue REST web API call
    HttpGet httpget = new HttpGet("https://localhost:8443/
soba/restTx/txId/791111196");
    httpget.addHeader("Accept", "application/json");
    System.out.println("Executing request: " + httpget.
getRequestLine());

    HttpResponse response = httpClient.execute(httpget);
    HttpEntity entity = response.getEntity();

    System.out.println("-----");
    System.out.println(response.getStatusLine());

    if (entity != null) {
        System.out.println("Response content length: " +
entity.getContentLength());
    }
}

```



```

        System.out.println("Response content type: " + entity.
            getContentType().getValue());
        BufferedReader reader = new BufferedReader
            (new InputStreamReader (entity.getContent()));
        String line = reader.readLine();
        while (line != null) {
            System.out.println (line);
            line = reader.readLine ();
        }
    }
    if (entity != null) {
        entity.consumeContent();
    }

    // When HttpClient instance is no longer needed,
    // shut down the connection manager to ensure
    // immediate deallocation of all system resources
    httpClient.getConnectionManager().shutdown();
}
}

```

After the successful execution, the above program yielded the following output. Note the json document at the end of this output.

```

Executing request: GET https://localhost:8443/soba/
restTx/txId/791111196 HTTP/1.1
-----
HTTP/1.1 200 OK
Response content length: -1
Response content type: application/json;charset=UTF-8
{"model":{"transaction":{"id":791111196,"type":"deposit",
"description":"test","status":"complete","accountId":
"118976398","amount":500.0,"transactionId":791111196,
"balance":500.0,"transDate":1301880518087,"initiator":
"user1001"},"now":"Sun Apr 03 21:02:19 PDT 2011"}}

```

Next, let's take a look at how security is implemented with SOBA.

15.7 SPRING SECURITY APPLIED TO SOBA

Security is a serious issue that every enterprise application must enforce one way or the other. It's a broad topic that we won't be able to cover it in detail. Instead, I'll help you gain a basic understanding of this subject using the code/configuration examples implemented on SOBA. Let's start with some basic concepts first in the next section.

15.7.1 Basic Concepts

Some basic concepts related to security in general include the following:

- *Authentication.* This is the process of verifying the identity of a principal against the credentials provided. This is similar to an airport security check-in process: You show a picture ID and your boarding ticket and then (mostly) you are allowed to pass. Two most commonly encountered types of authentications are:
 - *Form-based login authentication.* In this case, a login form is used to authenticate a user. This is the login authentication type used with SOBA.
 - *HTTP Basic authentication.* In this case, user credentials are embedded in HTTP request headers. It typically is used for authenticating requests made from remoting protocols and Web services. The REST client example presented in the previous section uses this type of authentication.
- *Authorization.* Note that authentication and authorization are two different concepts. Authorization is the process of determining and granting authorities to an authenticated user or principal for accessing intended resources. Authorities typically are granted in terms of *roles*. You might have noticed some entities like `ROLE_CUST`, `ROLE_REP`, `ROLE_ADMIN`, and so on, in the previous code snippets, and they are the roles defined in SOBA for authorizing users.
- *A Principal.* A principal is an object to be authenticated, which could be a user, an application, or a device.
- *Access Control.* This is a process to determine *who can access what resources*. It's done by comparing the permission bits of a resource with the roles of the principal attempting to access the resource. This subject will be discussed in the next section.

Next, let's see how Spring security framework works.

15.7.2 Security Configured in web.xml

The entry to invoking Spring security API for a Web application is defined in the `web.xml` file of the application. Refer back to Listing 15.1 for SOBA's `web.xml` file and note the following two XML elements defined there:

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>
    org.springframework.web.filter.
      DelegatingFilterProxy
  </filter-class>
</filter>
```

```

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```

Here, when the login page of SOBA is accessed, the filter named `springSecurityFilterChain` is invoked, which is a `DelegatingFilterProxy` security bean that delegates security checking to other Spring security beans. The URL pattern of “/*” associated with the filter `springSecurityFilterChain` specifies at the system level that any user who wants to access any page of a Web application needs to be authenticated first. Web security authentication is configured using the `<http>` element, as is discussed next.

15.7.3 Security Configured in `soba-security.xml`

This section explains what Spring security features are applied to SOBA. These features are configured in the `soba-security.xml` file as shown in Listing 15.18 below. The ACL part is omitted as it will be presented and discussed in the next section when we discuss ACL.

The measures taken to secure SOBA are reflected in the `soba-security.xml` file as follows:

- *HTTPS Channel Required.* Locate the `<http>` section and note the line of `<intercept-url pattern="/"**" requires-channel="https"/>`. This specifies that accessing SOBA is enforced with the secure HTTP protocol. The section `<concurrency-control ...>` specifies that at most 10 sessions can be opened by a same user. It can be set to 1 so that when a user is logged in, the second login with the same user credentials will log off the first user so that the first user would get alerted.
- *Authentication Types.* The two authentication types of `basic` and `form-based` are also specified in the `<http>` section. However, they are configured outside the `<http>` section separately. We'll describe each of them shortly.
- *Authentication Manager.* Locate the `<authentication-manager>` section and note the authentication provider configured with proper password encoder and `jdbc-user-service`. A table named `USERS` is used for storing user's credentials, as described in the preceding chapter.
- *Spring Security Filter Chain.* Locate the `<beans:bean id = "springSecurityFilterChain" ...>` section and note that the following three filter chains are defined (keep in mind that the order in which the filters are specified matter):
 - Images require no filters.
 - URLs with `/rest*/**` are authenticated with the HTTP basic method as explained previously. This pattern is for all external invocation of RESTful Web APIs.

- URLs with all other patterns are authenticated with form-based authentication.
- *Form-Based Authentication.* Locate the section for form-based authentication and note the properties of the default target URL, filterProcessesUrl, loginFormUrl, forceHttps, and error page, and so on.
- *HTTP Basic Authentication.* Locate the section for basic type of authentication and note that the realm property set to perfmath.com. You can specify your own realm here.
- *Access Decision Manager.* The outcome of an authentication action is determined by the voting policies. In the access decision manager section, RoleVoter and AuthenticatedVoter are specified. Each of these two voting mechanisms are described as follows:
 - *RoleVoter.* This method votes on an access control decision based on a user's roles. The access is not granted unless the user has all required roles.
 - *AuthenticatedVoter.* This method votes on an access control decision based on a user's authentication level. It votes to grant the access if the user's authentication level is equal or exceeds the required authentication levels. The valid levels are: IS_AUTHENTICATED_FULLY, IS_AUTHENTICATED_REMEMBERED, and IS_AUTHENTICATED_ANONYMOUSLY.
- *Filter Security Interceptor.* Locate the <beans:bean id = "filterSecurityInterceptor" ...> section and note the intercept-url patterns specified there. Each of these URL patterns has a special intention as is denoted there. For example, the administrator page can only be accessed by a user with the ROLE_ADMIN role. Also, the RESTful Web APIs cannot be accessed anonymously. For a real application, it might be necessary to divide further.
- *Secured-Annotations.* Finally, locate the first section of <global-method-security ...> and note the specification of secured-annotations = "enabled". This means that you can put something like @Secured ("ROLE_CUST", "ROLE_REP") prior to a method in a class to make the method secured. We'll see such examples in the next section when we discuss ACLs.

We have seen how to secure URL access using intercept-url patterns and secure methods using annotated security. Next, let's take a look at how Spring security can be implemented in views.

Listing 15.18 soba-security.xml (ACL part moved to Listing 15.19)

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xmlns:util="http://www.springframework.org/schema/util"
xmlns:security="http://www.springframework.org/schema/security"
xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-2.5.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-
    3.0.3.xsd">

<global-method-security pre-post-annotations="enabled"
    secured-annotations="enabled">
    <expression-handler ref="expressionHandler"/>
</global-method-security>

<http use-expressions="true">
    <intercept-url pattern="/**" requires-channel="https" />
    <http-basic/>
    <custom-filter position="FORM_LOGIN_FILTER" ref=
    "formLoginFilter"/>
    <session-management>
        <concurrency-control max-sessions="10"
            error-if-maximum-exceeded="true"/>
    </session-management>
    <logout logout-success-url="/logout.jsp" />

</http>

<authentication-manager alias="authenticationManager">
    <authentication-provider>
        <password-encoder hash="sha">
            <salt-source user-property="username"/>
        </password-encoder>
        <jdbc-user-service data-source-ref="dataSource"
            users-by-username-query="SELECT username, password,
            enabled FROM users WHERE username = ?"/>
    </authentication-provider>
</authentication-manager>
<beans:bean id="springSecurityFilterChain"
    class="org.springframework.security.web.FilterChainProxy">
    <filter-chain-map path-type="ant">
        <filter-chain pattern="/images/**" filters="none"/>
        <filter-chain pattern="/rest/**"
            filters="
            securityContextPersistenceFilterWithASCFALSE,
            basicAuthenticationFilter,
            exceptionTranslationFilter,
            filterSecurityInterceptor"/>
    </filter-chain-map>
</beans:bean>

```

```

    <filter-chain pattern="/*"
        filters="
            securityContextPersistenceFilterWithASCTrue,
            formLoginFilter,
            exceptionTranslationFilter,
            filterSecurityInterceptor"/>
    </filter-chain-map>
</beans:bean>
<beans:bean id="securityContextPersistenceFilterWithASCFalse"
    class="org.springframework.security.web.context.
        SecurityContextPersistenceFilter">
</beans:bean>

<beans:bean id="securityContextPersistenceFilterWithASCTrue"
    class="org.springframework.security.web.context.
        SecurityContextPersistenceFilter">
</beans:bean>

<!-- form based authentication -->

<beans:bean id="formLoginFilter"
class="org.springframework.security.web.authentication.
UsernamePasswordAuthenticationFilter">
    <beans:property name="authenticationManager"
        ref="authenticationManager"/>
    <beans:property name="authenticationSuccessHandler">
    <beans:bean class="org.springframework.security.web.
        authentication.
            SimpleUrlAuthenticationSuccessHandler">
    <beans:property name="defaultTargetUrl"
        value="/loginBroker.htm"></beans:property>
    </beans:bean>
    </beans:property>
    <beans:property name="filterProcessesUrl" value="/j_spring_
        security_check"/>
</beans:bean>

<beans:bean id="formAuthenticationEntryPoint"

class="org.springframework.security.web.authentication.
LoginUrlAuthenticationEntryPoint">
    <beans:property name="loginFormUrl" value="/login.jsp"/>
    <beans:property name="forceHttps" value="true"/>
</beans:bean>

<beans:bean id="formExceptionTranslationFilter"

class="org.springframework.security.web.access.
ExceptionTranslationFilter">
    <beans:property name="authenticationEntryPoint"
        ref="formAuthenticationEntryPoint"/>

```

```

    <beans:property name="accessDeniedHandler" ref=
        "formAccessDeniedHandler"/>
</beans:bean>

<beans:bean id="formAccessDeniedHandler"
    class="org.springframework.security.web.access.
        AccessDeniedHandlerImpl">
    <beans:property name="errorPage" value="/login.jsp?error=true"/>
</beans:bean>

<!-- basic authentication -->
<beans:bean id="basicAuthenticationFilter"

class="org.springframework.security.web.authentication.
www.BasicAuthenticationFilter">
    <beans:property name="authenticationManager">
        <beans:ref bean="authenticationManager"/>
    </beans:property>
    <beans:property name="authenticationEntryPoint">
        <beans:ref bean="basicAuthenticationEntryPoint"/>
    </beans:property>
</beans:bean>

<beans:bean id="basicAuthenticationEntryPoint"

class="org.springframework.security.web.authentication.
www.BasicAuthenticationEntryPoint">
    <beans:property name="realmName" value="perfmath.com"/>
</beans:bean>

<beans:bean id="basicExceptionTranslationFilter"
    class="org.springframework.security.web.access.
        ExceptionTranslationFilter">
    <beans:property name="authenticationEntryPoint" ref=
        "basicAuthenticationEntryPoint"/>

    <beans:property name="accessDeniedHandler" ref="
        basicAccessDeniedHandler"/>
</beans:bean>
<beans:bean id="basicAccessDeniedHandler"
    class="org.springframework.security.web.access.
        AccessDeniedHandlerImpl">
</beans:bean>

<!-- security -->
<beans:bean id="filterSecurityInterceptor"
    class="org.springframework.security.web.access.intercept.
        FilterSecurityInterceptor">
    <beans:property name="authenticationManager"
        ref="authenticationManager"/>

```

```

<beans:property name="accessDecisionManager"
ref="accessDecisionManager"/>
<beans:property name="securityMetadataSource">
  <filter-security-metadata-source>
    <intercept-url pattern="/rest*/**"
      access="ROLE_REST, ROLE_CUST, ROLE_REP"/>
    <intercept-url pattern="/login.jsp"
      access="ROLE_ANONYMOUS, ROLE_CUST,
        ROLE_REP, ROLE_ADMIN"/>
    <intercept-url pattern="/images/**"
      access="ROLE_ANONYMOUS, ROLE_CUST,
        ROLE_REP, ROLE_ADMIN"/>
    <intercept-url pattern="/admin.htm*" access="ROLE_ADMIN"/>
    <intercept-url pattern="/*"
      access="ROLE_CUST, ROLE_REP, ROLE_ADMIN"/>
  </filter-security-metadata-source>
</beans:property>

</beans:bean>

<beans:bean id="accessDecisionManager"
  class="org.springframework.security.access.vote.
  AffirmativeBased">
  <beans:property name="decisionVoters">
    <beans:list>
      <beans:bean class="org.springframework.security.access.
        vote.RoleVoter"/>
      <beans:bean
        class="org.springframework.security.access.vote.
        AuthenticatedVoter"/>
    </beans:list>
  </beans:property>
</beans:bean>

<beans:bean id="expressionHandler"
  class="org.springframework.security.access.expression.method.
  DefaultMethodSecurityExpressionHandler">
  <beans:property name="permissionEvaluator"
    ref="permissionEvaluator"/>
</beans:bean>

<beans:bean id="permissionEvaluator"
  class="org.springframework.security.acls.AclPermissionEvaluator">
  <beans:constructor-arg ref="aclService"/>
</beans:bean>

<!-- ACL setup. See Section 15.8 -->

</beans:beans>

```


15.7.4 Implementing Spring Security in Views

The resources such as embedded URLs in jsp pages can be protected similarly using the Spring Security Framework. The procedure is as follows:

- Enabling access to Spring security jsp tags. At the beginning of a jsp page, add the line `<%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>`.
- You can use the `Spring security:authorize jsp` tag to limit the access to a URL, for example,

```
<security:authorize ifAnyGranted="ROLE_REP">
<td><a href="<c:url value="loginBroker"/>">Rep Console</a></td>
<td/>
<td><a href="<c:url
value="manageTx.htm?customerId=${customerId}&accountId=
${accountId}" />">Manage
Transactions</a> </td>
</security:authorize>
```

This is an excerpt from the `activityList.jsp` file of SOBA to limit access to the Rep Console to representatives (i.e., authorized personnel) only. The other option is to use `ifNotGranted` in place of `ifAnyGranted` to handle certain situations more pertinently.

We wrap up this SOBA project with a discussion on how access control is applied to SOBA domain objects next.

15.8 SPRING ACL APPLIED TO SOBA

For complicated enterprise applications, in addition to securing Web pages, it's also necessary to control access at the domain object level. For example, a customer should not be allowed to view the transactions of other customers; also, a customer should not be allowed to reverse any of his own transactions—only authorized bank personnel should be allowed to do so. Such requirements demand applying security at the domain object level.

In this section, using the `transaction` domain object of SOBA, we demonstrate how Spring domain object security can be applied to SOBA based on the concept of Access Control List (ACL). (Note that it's necessary to distinguish the context of the term “transaction” between two different situations: one is a business transaction like deposit, withdraw, bill payment, and so on, in SOBA's context, and the other is the usual definition of “either all or none” in a non-business context. Which one is implied should be clear according to the proper context in question.). The same approach can be applied to other domain objects of SOBA or any other enterprise applications.

Since ACL entries are maintained in separate tables from the domain objects, let's first demonstrate how to set up ACL in Oracle.

15.8.1 Creating ACL Tables in Oracle

Spring ACL requires the following four ACL tables to be created in a relational database:

- *ACL_SID*. The *ACL_SID* table is designed to uniquely identify any principal or authority in the system ("SID" stands for "security identity"). It has only three columns: the ID, a textual representation of the SID, and a flag to indicate whether the textual representation refers to a principal name or a *GrantedAuthority*. Thus, there is a single row for each unique principal or *GrantedAuthority*. When used in the context of receiving a permission, a SID is generally called a "recipient."
- *ACL_CLASS*. The *ACL_CLASS* table is designed to uniquely identify any domain object class in the system. It has only two columns: the ID and the Java class name. Thus, there is a single row for each unique class for which ACL permissions will be created.
- *ACL_OBJECT_IDENTITY*. The *ACL_OBJECT_IDENTITY* table stores information for each unique domain object instance in the system. It has columns including the ID, a foreign key to the *ACL_CLASS* table, a unique identifier to indicate which *ACL_CLASS* instance the information is for, the parent, a foreign key to the *ACL_SID* table to represent the owner of the domain object instance, and whether ACL entries are allowed to inherit from any parent ACL. There is a single row for every domain object instance for which relevant ACL permissions are created.
- *ACL_ENTRY*. The *ACL_ENTRY* table stores the individual permissions assigned to each recipient. It has columns including a foreign key to the *ACL_OBJECT_IDENTITY*, the recipient (i.e., a foreign key to *ACL_SID*), whether auditing succeeded, and the integer bit mask that represents the actual permission being granted or denied. There is a single row for every recipient that receives a permission to work on a domain object.

With SOBA, the ACL tables are created using the `create_acl.sql` script as shown in Section 14.6.4.

The next step is to configure Spring ACL, as discussed next.

15.8.2 Configuring Spring ACL

To make it more manageable, one can have all ACL-related configuration settings contained in one XML file, and in our case, it would be `soba-acl.xml` file.

Then, this `soba-acl.xml` file can be added in the `web.xml` file under `contextConfigLocation`, as is shown below:

```
<web-app ...>
...
<context-param> <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/soba-servlet.xml
    /WEB-INF/soba-services.xml
    /WEB-INF/soba-security.xml
    /WEB-INF/soba-acl.xml
  </param-value>
</context-param>
...
</web-app>
```

However, for SOBA, majority of the ACL configuration settings stayed in the `soba security.xml` file, as is shown in Listing 15.19 below. The reason is that ACL is really a part of the entire security of an application, and it's more convenient to put ACL settings with other security settings into one file.

The core bean here is an ACL service. The Spring Security Framework has two ACL interfaces: `AclService`, which defines methods for reading ACLs, and `MutableAclService`, which defines methods for creating, updating, and deleting ACLs. The corresponding implementations of these two interfaces are `JdbcAclService` and `JdbcMutableAclService`.

Listing 15.19 `soba-security.xml` (ACL part)

```
<beans:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:util="http://www.springframework.org/schema/util"
  xmlns:security="http://www.springframework.org/schema/security"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util-2.5.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/
    spring-security-3.0.3.xsd">

  <global-method-security pre-post-annotations="enabled"
    secured-annotations="enabled">
    <expression-handler ref="expressionHandler"/>
  </global-method-security>

  <!-- Security setup... See Listing 15.17 -->
```

```

<!-- ACL start: ACL SERVICE DEFINITIONS -->

<beans:bean id="aclCache"
  class="org.springframework.security.acs.domain.
  EhCacheBasedAclCache">
  <beans:constructor-arg>
    <beans:bean class="org.springframework.cache.ehcache.
    EhCacheFactoryBean">
      <beans:property name="cacheManager">
        <beans:bean
          class="org.springframework.cache.ehcache.
          EhCacheManagerFactoryBean"/>
        </beans:property>
      <beans:property name="cacheName" value="aclCache"/>
    </beans:bean>
  </beans:constructor-arg>
</beans:bean>

<beans:bean id="lookupStrategy"
  class="org.springframework.security.acs.jdbc.
  BasicLookupStrategy">
  <beans:constructor-arg ref="dataSource"/>
  <beans:constructor-arg ref="aclCache"/>
  <beans:constructor-arg>
    <beans:bean
      class="org.springframework.security.acs.domain.
      AclAuthorizationStrategyImpl"><beans:constructor-arg>
        <beans:list>
          <beans:bean class="org.springframework.security.core.authority.
          GrantedAuthorityImpl">
            <beans:constructor-arg value="ROLE_ADMIN"/>
          </beans:bean>
          <beans:bean class="org.springframework.security.core.authority.
          GrantedAuthorityImpl">
            <beans:constructor-arg value="ROLE_ADMIN"/>
          </beans:bean>
          <beans:bean class="org.springframework.security.core.
          authority.
          GrantedAuthorityImpl">
            <beans:constructor-arg value="ROLE_ADMIN"/>
          </beans:bean>
        </beans:list>
      </beans:constructor-arg>
    </beans:bean>
  </beans:constructor-arg>
  <beans:bean class="org.springframework.security.acs.domain.
  ConsoleAuditLogger"/>
</beans:bean>

```

```

<beans:bean id="aclService" class="org.springframework.
security.acls.jdbc.
    JdbcMutableAclService">
<beans:constructor-arg ref="dataSource"/>
<beans:constructor-arg ref="lookupStrategy"/>
<beans:constructor-arg ref="aclCache"/>
<beans:property name="sidIdentityQuery"
    value="SELECT ACL_SID_ID_SEQ.CURRVAL FROM DUAL"/>
<beans:property name="classIdentityQuery"
    value="SELECT ACL_CLASS_ID_SEQ.CURRVAL FROM DUAL"/>
</beans:bean>

</beans:beans>

```

The `aclService` as is shown above requires three constructor arguments: (1) a `dataSource`, which defines a JDBC source as configured in the `soba-services.xml` file, (2) a `lookupStrategy`, which performs lookup for an ACL service, and (3) an `aclCache` for caching ACLs (in this case, a third-party product named `Ehcache` is used).

Note that the `lookupStrategy` specified above indicates that only a user who has the `ROLE_ADMIN` role can modify an ACL. Note also that the `aclService` specifies two property entries of `sidIdentityQuery` and `classIdentityQuery`, indicating how these access control entry (ACE) IDs should be created in JDBC. The semantics of these two queries should be clear based on how those two Oracle sequences were created in the previous chapter.

The next step is to implement create/update/delete ACLs in a domain object service, as is discussed next.

15.8.3 Maintaining ACLs for SOBA Domain Objects

In SOBA's context, the ACLs are applied to managing banking transactions of customers. Listing 15.20 shows the `SimpleAclTxManager.java` service that has the following methods:

- *createTransaction (Transaction tx)*. After a transaction is posted, the `addPermission` method is called to insert ACEs for the transaction created. Then the `updateAcl` method of the `mutableAclService` is called to update the ACL. The `addPermission` method is called three times: once for the principal (the customer in this case) for the READ permission, and twice for the granted authority with the role of `ROLE_REP` for ADMINISTRATION and DELETE permissions, respectively.
- *disputeTransaction (String txId)*. In this case, a zero amount transaction is inserted into the customer's account with the transaction status set to "disputed."

- **reverseTransaction (String txId).** In this `reverseTransaction` method, the amount of the disputed transaction is credited back to the customer's account and the transaction status is set to "reversed."

Listing 15.20 SimpleAclTxManager.java

```
package com.perfmath.odps.soba.service;

import java.sql.Timestamp;
import java.util.List;

import com.perfmath.odps.soba.model.dao.AclTransactionDao;
import com.perfmath.odps.soba.model.dao.LoginUserDao;
import com.perfmath.odps.soba.model.dao.AccountDao;
import com.perfmath.odps.soba.model.vo.Transaction;
import com.perfmath.odps.soba.util.RandomID;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

import org.springframework.security.access.annotation.Secured;
import org.springframework.security.acls.domain.BasePermission;
import org.springframework.security.acls.domain.GrantedAuthoritySid;
import org.springframework.security.acls.domain.ObjectIdentityImpl;
import org.springframework.security.acls.domain.PrincipalSid;
import org.springframework.security.acls.model.AccessControlEntry;
import org.springframework.security.acls.model.MutableAcl;
import org.springframework.security.acls.model.MutableAclService;
import org.springframework.security.acls.model.NotFoundException;
import org.springframework.security.acls.model.ObjectIdentity;
import org.springframework.security.acls.model.Permission;
import org.springframework.security.acls.model.Sid;

import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.
SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.annotation.Transactional;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.context.support.ApplicationObjectSupport;

import org.springframework.util.Assert;
import org.springframework.transaction.annotation.Transactional;
```

```

import org.springframework.transaction.support.TransactionCallback;
import com.perfmath.odps.soba.util.RandomID;
public class SimpleAclTxManager implements AclTxManager {
    private AclTransactionDao aclTransactionDao;
    private LoginUserDao loginUserDao;
    private AccountDao accountDao;
    private MutableAclService mutableAclService;
    public LoginUserDao getLoginUserDao() {
        return loginUserDao;
    }

    public void setLoginUserDao(LoginUserDao loginUserDao) {
        this.loginUserDao = loginUserDao;
    }

    public AccountDao getAccountDao() {
        return accountDao;
    }

    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    public MutableAclService getMutableAclService() {
        return mutableAclService;
    }

    public void setMutableAclService(MutableAclService
mutableAclService) {
        this.mutableAclService = mutableAclService;
    }

    @Secured("AFTER_ACL_READ")
    public Transaction findByTransactionID(String txId) {
        return aclTransactionDao.findByTransactionID(txId);
    }

    @Secured("AFTER_ACL_COLLECTION_READ")
    public List getTransactions() {
        return aclTransactionDao.getTransactionList();
    }

    public void updateTransaction(Transaction tx) {
        aclTransactionDao.update(tx);
    }

    @Transactional
    @Secured({"ROLE_REP", "ACL_TRANSACTION_DELETE", "ACL_TX_DELETE"})

    public void deleteTransaction(String txId) {

```

```

aclTransactionDao.delete(txId);
    ObjectIdentity oid = new ObjectIdentityImpl(Transaction.class,
        txId);

    mutableAclService.deleteAcl(oid, false);

}

public List<Transaction> getTransactions(String accountId) {
    return aclTransactionDao.getTransactions(accountId);
}

private String getCustomerUsername (Transaction tx) {
    String username = "";
    String authority = SecurityContextHolder.getContext()
        .getAuthentication().getAuthorities().toString();
    if (authority.contains ("ROLE_CUST")) {
        Object principal =
            SecurityContextHolder.getContext().getAuthentication().
                getPrincipal();
        if (principal instanceof UserDetails) {
            username = ((UserDetails)principal).getUsername();
        } else {
            username = principal.toString();
        }
    } else {
        String accountId = tx.getAccountId();
        String customerId = accountDao.getCustomerId(accountId);
        username = loginUserDao.getUsernameByCustomerId(customerId);
    }
    return username;
}

@Transactional
//@Secured("ROLE_USER")
public void createTransaction(Transaction tx) {
    aclTransactionDao.insert(tx);
    addPermission(tx, new PrincipalSid(getCustomerUsername(tx)),
        BasePermission.READ);
    addPermission(tx, new GrantedAuthoritySid ("ROLE_REP"),
        BasePermission.ADMINISTRATION);
    addPermission(tx, new GrantedAuthoritySid ("ROLE_REP"),
        BasePermission.DELETE);
}

@Transactional
public void disputeTransaction(String txId) {
    Transaction tx = aclTransactionDao.findByTransactionID(txId);
    System.out.println("dispute Tx " + txId);
    tx.setAmount(0.0);
    tx.setDescription("Customer disputed (txTd=" + txId + "): " +
        tx.getDescription());
    tx.setStatus("disputed");
    tx.setTransactionId(Integer.parseInt((new RandomID(9)).

```



```

        getId());
        tx.setTransDate(new Timestamp(System.currentTimeMillis()));
        aclTransactionDao.insert(tx);
        addPermission(tx, new PrincipalSid(getCustomerUsername(tx)),
            BasePermission.READ);
        addPermission(tx, new GrantedAuthoritySid("ROLE_REP"),
            BasePermission.ADMINISTRATION);
        addPermission(tx, new GrantedAuthoritySid("ROLE_REP"),
            BasePermission.DELETE);
    }

    @Transactional
    public void reverseTransaction(String txId) {
        Transaction tx = aclTransactionDao.findByTransactionID(txId);
        System.out.println("Reverse Tx " + txId);
        tx.setAmount(-tx.getAmount());
        tx.setDescription("Reversed: " + tx.getDescription());
        tx.setTransactionId(Integer.parseInt((new RandomID(9)).
            getId()));
        tx.setTransDate(new Timestamp(System.currentTimeMillis()));
        aclTransactionDao.insert(tx);
        addPermission(tx, new PrincipalSid(getCustomerUsername(tx)),
            BasePermission.READ);
        addPermission(tx, new GrantedAuthoritySid("ROLE_REP"),
            BasePermission.ADMINISTRATION);
        addPermission(tx, new GrantedAuthoritySid("ROLE_REP"),
            BasePermission.DELETE);
    }

    public void setAclTransactionDao(AclTransactionDao
aclTransactionDao) {
        this.aclTransactionDao = aclTransactionDao;
    }

    public void addPermission(Transaction tx, Sid recipient,
Permission permission) {
        MutableAcl acl;
        ObjectIdentity oid = new ObjectIdentityImpl(Transaction.class,
tx.getId());

        try {
            acl = (MutableAcl) mutableAclService.readAclById(oid);
        } catch (NotFoundException nfe) {
            acl = mutableAclService.createAcl(oid);
        }

        acl.insertAce(acl.getEntries().size(), permission, recipient, true);
        mutableAclService.updateAcl(acl);
    }

    public void deletePermission(Transaction tx, Sid recipient,

```

```

Permission permission) {
    ObjectIdentity oid = new ObjectIdentityImpl(Transaction.class,
        tx.getTransactionId());
    MutableAcl acl = (MutableAcl) mutableAclService.readAclById(oid);

    // Remove all permissions associated with this particular recipient
    (string equality to KISS)
    List<AccessControlEntry> entries = acl.getEntries();
    for (int i = 0; i < entries.size(); i++) {
        if (entries.get(i).getSid().equals(recipient) && entries.get(i).
            getPermission().equals(permission)) {
            acl.deleteAce(i);
        }
    }
    mutableAclService.updateAcl(acl);
}
}

```

You might have noticed the annotation `@Transactional` placed in front of each ACL-related methods introduced above. This is specific to the Spring Security Framework that the `JdbcMutableAclService` requires all of the methods containing ACL insert/update/delete operations to run in a JDBC transactional context. To enable the `@Transactional` annotation, it's necessary to add the following elements in the service XML configuration file (in this case, the `soba-services.xml` file), together with the JDBC data source transaction manager and the ACL Tx Manager beans added as well:

```

<...>
<tx:annotation-driven/>

<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.
        DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<bean id="aclTxManager"
    class="com.perfmath.odps.soba.service.SimpleAclTx
        Manager">
    <property name="aclTransactionDao" ref=
        "aclTransactionDao"/>
    <property name="loginUserDao" ref="loginUserDao"/>
    <property name="accountDao" ref="accountDao"/>
    <property name="mutableAclService" ref="aclService"/>
</bean>
<...>

```

15.8.4 Applying ACLs to Business Operations

When a domain object is ACL-enabled, which means that it had ACL information created as well when the domain object was inserted into the database, you can use that ACL information to grant or hide the access privilege based on the permissions set to the domain object. For example, if you want to hide the `Reverse` link of a transaction that does not have the `ADMINISTRATION` permission implied in number “16,” you can make it happen by using the `<security:accesscontrollist>` tag as is shown in Listing 15.21 below in the `manageTx.jsp` file. This tag’s function is to filter the access based on a domain object’s ACL.

Listing 15.21 `manageTx.jsp`

```
<%@ include file="include.jsp"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="security"
    uri="http://www.springframework.org/security/tags"%>

<html>
<head>
<title>Tx List</title>
</head>
<%@ include file="banner.jsp"%>
<body>
<center>
<h2>You are logged in as <i> <security:authentication
    property="name"/> </i> with the following authorities:</h2>

<security:authentication property="authorities" var="authorities"/>
<ul>

    <c:forEach items="${authorities}" var="authority">
        <li>${authority.authority}</li>
    </c:forEach>
</ul>
<hr> <br>

<br>
<hr>
<security:authorize ifAnyGranted="ROLE_REP">
    Back to <a href="<c:url value="loginBroker"/>"> Rep Console</a>
</security:authorize>
<hr/>
<security:authorize ifAnyGranted="ROLE_REP, ROLE_CUST">
    <table>

        <tr>
            <c:forEach var="column"
                items="Date, Type, Description, Debit, Credit,
                    Balance, Tx ID, Action">
```

```

        <th align="left" bgcolor="#00184A"><FONT
            COLOR="#FFFFFF">${column}
        </FONT></th>
    </c:forEach>
</tr>

<c:forEach items="${txs}" var="tx">

    <tr>
        <!-- formatDate doesn't work if yyyy-mm-dd. must be MM-->
        <td width=100><fmt:formatDate value=
            "${tx.transDate}" pattern="yyyy-MM-dd"/></td>

        <td width=80>${tx.type}</td>
        <td width=300>${tx.description}</td>
        <c:choose>
            <c:when test="${tx.amount > 0.0}">
                <td width=100></td>
                <td width=100>${tx.amount}</td>
            </c:when>
            <c:otherwise>

                <td width=100><FONT
                    COLOR="#FF0000">${tx.amount} </FONT></td>
                <td width=100></td>

            </c:otherwise>
        </c:choose>
        <td width=100>${tx.balance}</td>
        <td>${tx.transactionId}</td>
        <security:accesscontrollist domainObject="${tx}"
            hasPermission="16">
            <td><a
                href="reverseTx.htm?txId=${tx.transactionId}
                    &accountId=${tx.accountId}">
                Reverse
            </a></td>
        </security:accesscontrollist>

    </tr>

</c:forEach>

</table>
</security:authorize><br>
<br>
<%@ include file="showLoadTime.jsp"%></center>
</body>
</html>

```

There are two more uses with ACL-enabled domain objects:

- Using the object's ACL to make access control decisions on the methods that operate on the object. In this case, it's necessary to configure an `AclEntryVoter` to help decide whether a method is allowed to be invoked.
- Making access control decisions based on the permissions of the domain objects returned from the methods that are subject to the ACL constraints. In this case, it's necessary to configure an `AclAfterInvocationProvider` if the method returns one domain object or an `AclAfterInvocationCollectionFilteringProvider` if the method returns a collection of domain objects to help decide if the domain object or the collection of domain objects is allowed to be returned to the user.

These two cases of applying ACL at the method level are significantly more complicated than the above jsp example, though.

15.8.5 Testing ACLs with SOBA

In this section, let's construct a more complete test case to see ACL in action with SOBA. Follow the step-by-step procedure below:

1. Login as an established customer, for example, as user1001 as shown in Figure 15.17.
2. Create a non-ACL transaction by following the below sequence of actions:
 - (a) Click the tab of `Create a Tx` and gets to the view for posting a transaction (see Figure 15.18). The customer then clicks `Submit` after entering all required info.



 The image shows a web form titled "Existing User Login". It has three input fields: "Username:" with the value "user1001", "Password:" with masked characters "*****", and a dropdown menu with the selected option "An Established Customer". Below the dropdown is a right-pointing arrow icon. At the bottom of the form are two buttons: "Login" and "Reset".

Figure 15.17 SOBA: The customer user1001 logs in as an established customer.



[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

SOBA :: Post a transaction to an account

Receiver AccountID:	118976398
Type:	test
Initiator:	user1001
Description:	non acd test
Amount:	99.99

Figure 15.18 SOBA: A customer clicks the *Create a Tx* tab and gets to the view for posting a transaction. Then the customer clicks *Submit* after entering all required info.

- (b) The customer gets a view showing that the transaction is completed successfully (see Figure 15.19). The customer then clicks the *View Account Activity* link.
 - (c) The customer gets back to the default view showing account activities. Note that \$99.99 transaction posted (see Figure 15.20).
3. Create an ACL transaction by following the below sequence of actions:
- (a) Click the tab *Create an ACL Tx*. The customer now gets to the view for posting a transaction again (see Figure 15.21).
 - (b) Once again, the customer succeeded with a view showing the success of the transaction (see Figure 15.22). The customer then clicks the *View Account Activity* link.



[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

The Transaction has been created successfully for the customer with ID 173666039.

[Create a New Transaction](#)

or

[View Account Activity](#)

Figure 15.19 SOBA: The customer gets a view showing the transaction is completed successfully. The customer then clicks the *View Account Activity* link.



A Secure Online Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

[Open an Account](#) | [Create a Tx](#) | [Create an ACL Tx](#) | [Bill Payment](#) | [Transfers](#) | [Dispute Transactions](#) | [Messages](#)

Logged in as: user1001

Customer Account Details

Customer ID: 173666039
 Account ID: 118976398
 Account Name: my checking
 Account Type: Checking
 Account Balance: 1612.0

Account Name	Type	Range (date input format: yyyy-mm-dd)
Premium Checking	Checking	☒ Last 30 days ☐ From: To: Get History Reset

Date	Type	Description	Debit	Credit	Balance
2011-04-16	test	non acl test		99.99	1612.0
2011-04-16	bill pay	pay to user99	-7.99		1512.01
2011-04-16	acl tx	Reversed: acl tx test	-29.99		1520.0

Figure 15.20 SOBA: The customer gets back to the default view showing account activities. Note the \$99.99 transaction posted.

- (c) Now the \$199.99 deposit is seen on the account activity view (see Figure 15.23).
4. Dispute an transaction by following the below sequence of actions:
- (a) The customer logs in and sees the account activities from the last 30 days. In this case, it is the same as Figure 15.23.



A Secure Online Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

SOBA :: Post a transaction to an account

Receiver Account ID:	118976398
Type:	test
Initiator:	user1001
Description:	acl tx test
Amount:	199.99

Figure 15.21 SOBA: The customer clicks the Create an ACL Tx tab and gets to the view for posting a transaction again.



The Transaction has been created successfully for the customer with ID 173666039.

[Create a New Transaction](#)

or

[View Account Activity](#)

Figure 15.22 SOBA: Once again, the customer succeeded with a view showing the success of the ACL transaction.

- (b) The customer clicks the `Dispute Transactions` tab and gets a list of transactions. The customer disputes a transaction by clicking the ID of the transaction. In this case, the customer clicks the transaction with ID 341941040 (see Figure 15.24).
- (c) The customer gets to the `Dispute` view and clicks the `Dispute` button to dispute the transaction (see Figure 15.25).
- (d) The customer sees the disputed transaction as highlighted. Note in Figure 15.26 that the customer dispute does not change the balance.

The screenshot shows the "Customer Account Details" page in the SOBA application. At the top, there's a navigation bar with links: "Open an Account", "Create a Tx", "Create an ACL Tx", "Bill Payment", "Transfers", "Dispute Transactions", and "Messages". Below this, it says "Logged in as: user1001". The main section is titled "Customer Account Details" and contains the following information:
Customer ID: 173666039
Account ID: 118976398
Account Name: my checking
Account Type: Checking
Account Balance: 1811.99
Below this information, there's a section for filtering transactions. It has a table with "Account Name" and "Type" columns. The "Account Name" is "Premium Checking" and the "Type" is "Checking". To the right, there's a "Range (date input format: yyyy-mm-dd)" section with radio buttons for "Last 30 days" (selected) and "From: To:". Below the range section are "Get History" and "Reset" buttons.
At the bottom, there's a table showing transaction history:

Date	Type	Description	Debit	Credit	Balance
2011-04-16	test	acl tx test		199.99	1811.99
2011-04-16	test	non acl test		99.99	1612.0
2011-04-16	bill pay	pay to user99	-7.99		1512.01

Figure 15.23 SOBA: Now note the \$199.99 deposit from the previous step.



SOBA :: Safe Online Banking Application

Greetings, it is now Sat Apr 16 21:46:40 PDT 2011

Account ID: 118976398

To dispute a transaction, click the transaction ID.

Txid (regular)	Txid (rest)	TxDate (yyyy-MM-dd)	Description	Amount (\$)	Balance (\$)	Status
341941040	341941040	2011-04-16	acl tx test	199.99	1811.99	complete
634248766	634248766	2011-04-16	non acl test	99.99	1612.0	complete
462027025	462027025	2011-04-16	pay to user99	-7.99	1512.01	complete
593190624	593190624	2011-04-16	Reversed: acl tx test	-29.99	1520.0	complete

Figure 15.24 SOBA: The customer clicks the Disputes Transactions tab and gets a list of transactions to dispute. The customer disputes a transaction by clicking the transaction.

5. A representative manages the disputed customer transactions as follows:
 - (a) The representative logs in as a Rep as shown in Figure 15.27.
 - (b) The representative then enters the Customer ID and gets to the customer's account (see Figure 15.28).
 - (c) The representative clicks the *Manage Transactions* link to retrieve the customer's activities (see Figure 15.29).
 - (d) The representative sees the customer's \$199.99 transaction as shown in Figure 15.30. Note that the non-ACL transaction created with an amount of



SOBA :: Safe Online Banking Application

Greetings, it is now Sat Apr 16 21:47:20 PDT 2011

Dispute the Transaction (341941040)

Review the transaction, then click Cancel or Dispute below.

Transaction Account ID:	118976398
Transaction Date:	2011-04-16 21:43:53.859
Transaction Description:	acl tx test
Transaction Amount:	199.99
Transaction Status:	complete
Cancel	Dispute

Figure 15.25 SOBA: The customer clicks the Dispute button to dispute the transaction.



A Secure Online
Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

You are logged in as **rep100** with the following authorities:

- ROLE_REP

Back to [Rep Console](#)

Date	Type	Description	Debit	Credit	Balance	Tx ID	Action
2011-04-16	test	Customer disputed (txId = 341941040): acl tx test	0.0		1811.99	 382902201	Reverse
2011-04-16	test	acl tx test		199.99	1811.99	341941040	Reverse
2011-04-16	test	non acl test		99.99	1612.0	634248766	
2011-04-16	bill pay	pay to user99	-7.99		1512.01	462027025	Reverse

Figure 15.26 SOBA: The customer sees the disputed transaction as highlighted. Note that the customer dispute does not change the balance.

\$99.99 is not reversible while the ACL transaction is. The representative clicks [Reverse](#) link to reverse the ACL-enabled transaction.

- (e) The representative sees that the transaction is successfully reversed (see Figure 15.31).

This wraps up our SOBA project, and I hope it’s not too challenging.



A Secure Online
Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

Prospective Customers: Don't have an account? [Open Now.](#)

Established Customers: Don't have a user ID or password? [Register](#)

Existing User Login

Username:

rep100

Password:



A Rep

Login

Reset

Figure 15.27 SOBA: A representative named 'rep100' logs in as a Rep as selected from the drop-down list.



SOBA

A Secure Online Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

Rep Console (Search for a Customer)

Customer ID:

173666039

Account Type:

Checking ▾

First Name:

Last Name:

Email:

Address:

Zipcode:

Submit

Reset

Figure 15.28 SOBA: The representative enters the customer ID to get to the customer's account.



SOBA

A Secure Online Banking Application®

[Contact Us](#)
[Privacy Policy](#)
[Log Off](#)

[Open an Account](#) | [Create a Tx](#) | [Create an ACL Tx](#) | [Bill Payment](#) | [Transfers](#) | [Dispute Transactions](#) | [Messages](#)

Logged in as: rep100 [Rep Console](#) [Manage Transactions](#)

Customer Account Details

Customer ID:

173666039

Account ID:

118976398

Account Name:

my checking

Account Type:

Checking

Account Balance:

1811.99

Account Name

Type

Range (date input format: yyyy-mm-dd)

Premium Checking ▾

Checking ▾

☒ Last 30 days ▾

☐ From:

To:

Get History

Reset

Date	Type	Description	Debit	Credit	Balance
2011-04-16	test	Customer disputed (txId = 341941040): acl tx test	0.0		1811.99
2011-04-16	test	acl tx test		199.99	1811.99
2011-04-16	test	non acl test		99.99	1612.0
2011-04-16	bill pay	pay to user99	-7.99		1512.01

Figure 15.29 SOBA: The representative then clicks the Manage Transaction link to retrieve customer's activities.

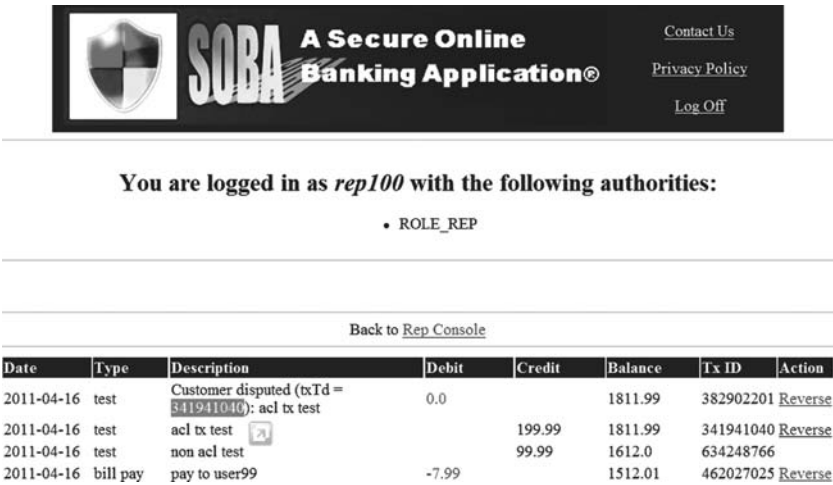


Figure 15.30 SOBA: The representative sees the customer's \$199.99 transaction and clicks Reverse link to reverse it.



Figure 15.31 SOBA: The representative sees that the transaction is successfully reversed.

15.9 SUMMARY

In this chapter, we have taken a step-by-step, end-to-end approach to demonstrating how SOBA was developed with the Spring MVC framework using Oracle as the backend. This sample application helped illustrate how the application tier and Web tier work with Oracle as the backend tier to deliver software solutions to meet today's business requirements.

Some of the newest technologies used for developing SOBA such as the Spring MVC Framework, the ORM Hibernate framework, and RESTful Web services illustrate how today's large-scale enterprise applications are built. This project also demonstrated how to secure and manage transactions outside Oracle not only to ease application development but also to prevent an Oracle server from being stressed excessively with additional loads arising from security and transaction management.

Part Three that follows focuses on how to optimize and tune Oracle performance and scalability. It's important to understand that the performance and scalability of an Oracle-based enterprise application typically span all the tiers. And each tier needs to be optimized and tuned to meet the required performance and scalability of an application at the system level. Such a goal can be achieved only if application developers understand Oracle performance and scalability and thus know how to build performance and scalability into their products through the complete development life cycle. This is one of the major objectives of this text.

RECOMMENDED READING

Texts recommended for learning more about Spring Framework and RESTful Web services:

Spring Documentations in General:

<http://www.springsource.org/documentation>.

Spring Reference Documentation:

<http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf>

Spring Security Reference Documentation:

<http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.pdf>

RESTful Web Services:

B. Burke, *RESTful Java with JAX-RS*, O'Reilly, Sebastopol, 2010.

EXERCISES

- 15.1 Describe the pros and cons of using JDBC versus Hibernate with Oracle.
- 15.2 Describe the respective roles of each part of the MVC architecture. Does a DAO (Data Access Object) belong to the model part of the MVC architecture?
- 15.3 Walk through the logic flow of creating an account with SOBA. Identify the model/view/controller elements involved in creating a new account with SOBA.
- 15.4 What's the difference between authentication and authorization? What's the sequence of these two processes?
- 15.5 Explain the concept of a security principal in the context of software. Does this concept apply to users only?

Part Three

Optimizing Oracle Performance and Scalability

A man paints with his brains and not with his hands.

—Michelangelo

Optimizing Oracle performance and scalability is a multi-faceted effort. Oracle itself rarely is the purpose. Instead, it's the application which Oracle supports that makes Oracle a useful product. Therefore, the application itself should be designed and implemented to run efficiently on Oracle. Oracle can make a well-designed application run better, but it can't make a poorly-designed application run fast.

With an Oracle-based enterprise application, its business logic design must be lean, waste-free, and more importantly, conscious of feeding Oracle with carefully-crafted SQLs. In my other text (Liu, 2009), I shared an experience that a real enterprise application was made 5.3 times faster simply by eliminating a database trigger that didn't have to be fired at all whenever an INSERT or UPDATE occurs. Although that was an extreme example, it does indicate that it's really the design and implementation of an application that determine the performance and scalability of the application itself. A well-designed application is tunable, whereas a poorly-designed one isn't.

What is Oracle? The simplest answer could just be that Oracle is a SQL execution engine. Applications feed Oracle with various types of SQL statements, and then Oracle processes them and returns the result set to the applications. After consuming

the result sets from Oracle, an application issues SQLs again to Oracle, and thus keeps going cycle by cycle until the application is end-of-life-cycled someday. Therefore, the bottom line is how one can write efficient SQLs, equipped with the knowledge of what built-in infrastructure Oracle has to help support running SQLs as efficiently as possible. That's the subject of this part.

This part consists of the following four chapters:

- Chapter 16 reveals how Oracle determines the optimum execution plans intelligently with its cost-based optimizer (CBO) that has become mature with Oracle 10g and 11g. It's important to understand that it's the CBO that works diligently behind the scene so that every SQL can be executed as fast as possible.
- Chapter 17 helps you understand how one can help a CBO choose an optimal execution plan for a SQL statement by properly composing the SQL statement.
- Chapter 18 dives deep on Oracle indexing, as indexing is one of the most critical techniques to optimize the performance and scalability of an Oracle-based application.
- Chapter 19 introduces various Oracle auto-tune features that can help diagnose Oracle performance and scalability issues significantly.

The objective of this part is to help you acquire a very basic set of skills in dealing with Oracle performance and scalability issues so that you can take care of such issues for your Oracle-based products.

Let's begin with explaining how an Oracle CBO works next.

16

Logistics of the Oracle Cost-Based Optimizer (CBO)

In activities other than purely logical thought, our minds function much faster than any computer yet devised.

—DANIEL CREVIER, *AI: The Tumultuous History of the Search for Artificial Intelligence*

In Chapter 9, we introduced the concept of an Oracle SQL optimizer while discussing Oracle Wait Interface (OWI). We also described how the original rule-based optimizer (RBO) was phased out and gave its place to the newly introduced optimizer—the cost-based optimizer or the CBO. In this chapter, we spend more time delving deeper into the logistics of the CBO to understand how a CBO determines an optimal execution plan for a SQL behind the scene. Equipped with sufficient knowledge of how a CBO works, we would be able to tune SQLs and optimize Oracle performance and scalability more effectively and efficiently.

The following subjects are covered in this chapter:

- Life of a SQL Statement in Oracle
- Oracle SQL Optimizer: Rule-Based versus Cost-Based
- What CBO Statistics Are
- Pivot Role of Gathering Database Statistics to CBO
- Methods of Gathering CBO Statistics
- Locking and Unlocking CBO Statistics
- EXPLAIN PLAN—a Handle to CBO
- Data Access Methods—CBO's Footprints
- Looking up CBO's Plan Hidden in V\$SQL_PLAN
- When CBO May Generate Suboptimum Execution Plans

Let's begin with explaining the life of a SQL statement in Oracle next.

16.1 LIFE OF A SQL STATEMENT IN ORACLE

Before exploring the life of a SQL statement executed in Oracle, we need to explain a very important concept in Oracle: `cursor`. Every SQL statement needs or has a cursor associated with it. Simply put, a cursor is a handle pointing to a private SQL area that contains a parsed SQL statement's execution plan and other information. Both shared and private sql areas are part of a library cache in the shared pool of an SGA. A cursor could also serve as a pointer pointing to a PGA's sub-area where the values of variables or the cursor's state (open, bound, executed, and closed) are stored.

There are two initialization parameters specific to the concept of a cursor:

- `OPEN_CURSORS`—Specifies the maximum number of open cursors a session can have initially. This sets an upper limit to how many open cursors a session can have to prevent a session from creating an excessive number of cursors. An open cursor can help prevent a SQL statement from being parsed at all, as being *open* means its associated, parsed SQL statement still is in the sql area and can be used as is.
- `SESSION_CACHED_CURSORS`—Specifies the number of session cursors to cache. A cursor for a SQL statement that has been parsed repeatedly is placed in the session cursor cache to prevent future parsing again.

The life of a SQL Statement in Oracle consists of the following stages:

1. `Cursor Logistics`. When a SQL statement is received, Oracle first checks the cursor cache to see if an open cursor already exists for the current SQL statement. If it does, then Oracle follows the cursor to pick up the execution plan stored in the library cache, which is part of a shared pool in the SGA. Then the SQL statement is executed with the existing execution plan. If a cursor exists in

the cursor cache but is not open, Oracle can still execute the SQL statement directly. Otherwise, if a cursor cannot be found in the cursor cache, Oracle opens a cursor for the current SQL statement and proceeds to the next stage of parsing.

2. **Parsing (Soft/Hard Parse).** Oracle first hashes the current SQL statement and looks it up in the sql area with the hashed values (it's possible that a parsed SQL statement is still in the sql area while the original cursor pointing to it is aged out and gone). If a sql area lookup is successful, it results in a *soft parse* that the current SQL can be executed directly. Otherwise, namely, there is neither a cached cursor in the cursor cache nor a parsed SQL statement in the library cache, the worst has occurred and a *hard parse* ensues. A hard parse consists of the following steps:
 - a. Checking syntax and semantics as well as the user's access permission. Common syntactic errors include invalid expressions and misspelled SQL keywords such as **SELECT**, **FROM**, **WHERE**, **GROUP BY**, **ORDER BY**, and so on. Common semantic errors include misspelled column names, misspelled table names, and so on. If there are either syntactic or semantic errors, the SQL statement is returned to the issuer and an error is thrown. User's permissions include access to the schemas and other referenced objects.
 - b. Acquiring locks on the required objects so that they will not change before parsing is completed.
 - c. Initiating the optimizer to determine the optimum execution plan.
 - d. Loading the hard-parsed SQL statement into a shared sql area.
3. **Result Description (SELECT SQL Statement Only).** This stage determines the characteristics of a query's result, for example, names, lengths, data types, and so on.
4. **Result Definition (SELECT SQL Statement Only).** This stage specifies the location, data types, and size of the variables defined to receive fetched values.
5. **Variable Binding.** This stage scans the SQL statement for variables and binds or fills these variables with their values. Then the execution plan becomes a bound execution plan with all variables determined with their actual values.
6. **Execution.** This stage swings into a full play by first checking if the data it needs for the query is already in the data block buffer cache. If yes, logical reads or buffer gets ensue, otherwise hard physical reads off the disks are hit. In the cases of **UPDATE** and **DELETE**, the relevant records are locked. It also needs to fill the redo log buffer and create a pointer to the undo/rollback segment; and then, the data is changed. Note that this also is the point to carry out array processing or batch processing of a number of SQL statements as described in Chapter 20.
7. **Result Fetch (SELECT SQL Statement Only).** This stage fetches the result set with array fetch, which retrieves multiple rows at a time.

8. Result Processing (SELECT SQL Statement Only). This stage sorts the result set if sorting was specified in the SQL statement with GROUP BY or ORDER BY conditions.

The life of a SQL statement in Oracle ends here. However, the challenge with SQL tuning has not started yet. The common notion is that the most challenging part of SQL tuning is to tune joins that have many tables involved, typically in data warehouse applications. This has always been an interesting subject to me, as there could be many variations about how a complex join can be executed. My advice is that let the mighty Oracle CBO do the job for you instead of trying to beat or outsmart the CBO yourself. However, it's necessary to understand the joins that deal with only a few tables, in order to be able to tune SQLs for best possible SQL performance. SQL tuning will be covered in the next chapter.

Let's next have a discussion on Oracle's SQL optimizers, which have been the optimization engines behind the execution of all SQL statements with optimal execution plans.

16.2 ORACLE SQL OPTIMIZER: RULE-BASED VERSUS COST-BASED

Every commercial quality database product has a SQL optimizer. The output from a SQL optimizer is an execution plan that represents the optimum method of execution out of many potential execution schemes or plans for the SQL statement in question.

Oracle has a long history of improving its SQL optimizer with every release. The earlier optimizer was called *rule-based*, as an optimal execution plan for a SQL statement was determined based on a simple set of rules of thumb (or heuristics). This had made SQL tuning using the rule-based optimizer (RBO) more of a black art than a science. The Oracle RBO, introduced in Oracle 5, was phased out formally in Oracle 10g, giving its throne to a new replacement—the Cost-Based Optimizer (CBO). The Oracle CBO was introduced in Oracle 7 and became mature enough to replace the RBO since Oracle 10g. It's interesting to note that the two coexisted for about three Oracle releases.

In contrast to the RBO, the CBO has made SQL tuning more of a science than an art. Rather than using empirical rules with fixed properties of tables and indexes as with the RBO, the CBO determines the optimal execution plan out of all possible plans based on the concept of *cost*, which is a quantitative measure of a step in executing a SQL statement. The execution plan with the minimum cost is chosen for a SQL statement to be executed.

When a SQL statement is received, it is sent to the Oracle CBO for the optimum execution plan if a reusable plan does not exist in the cache. The CBO uses the following procedure to determine an optimum execution plan:

1. Evaluating expressions and conditions in the SQL statement. This is more of a preparation step.

2. Transforming the SQL statement. This step is necessary only if the SQL statement is considered a complex one, for example, containing correlated subqueries or views, in which case, the original SQL statement is transformed into an equivalent join statement.
3. Choosing an optimizer objective. As described in section 3.3 of my other text (Liu, 2009), there are two types of computing tasks: batch jobs and OLTP applications. In terms of performance, batch jobs aim at maximum throughput, while OLTP applications aim at fastest possible responses. The CBO is hinted toward a performance objective with an initialization parameter named `OPTIMIZER_MODE`. By default, this parameter is set to `ALL_ROWS`, which favors throughput over response times. If this parameter is set to `FIRST_ROWS_n` where `n` represents the first `n` rows of a table, the CBO generates the optimum execution plan with best possible response times.
4. Choosing an access path. An access path determines how data is retrieved from the database. Batch jobs can be executed more efficiently with full table scans that retrieve data in large quantities, while OLTP applications perform better with index-guided retrievals, which retrieve only a small number of rows of a table.
5. Choosing the join orders. A SQL join statement may involve joining more than two tables. In this case, the CBO determines the sequence of tables to be joined as well as the joining method that leads to a minimum cost.

One can also influence the CBO in determining the optimum execution plan using `optimizer hints`, which are instructions to the optimizer. For example, the SQL statement `SELECT /*+ FIRST_ROWS (20) */ * FROM myTable;` instructs the CBO to choose the minimum cost plan that returns the first 20 rows from the table. However, one should realize that it's hard to beat or outsmart the CBO in every case.

A CBO determines optimal execution plans based on available database statistics. The next section helps you understand what CBO statistics exactly are.

16.3 CBO STATISTICS

CBO statistics are a collection of quantitative data that describe the statistical characteristics of a database and its objects at various levels (mostly tables, indexes, columns). The CBO uses these statistics as the basis for cost calculations to determine the optimum execution plan for each SQL query.

The database stores optimizer statistics in the static data dictionary. You can access those statistics using the following static data dictionary and dynamic views:

- `DBA_TABLES` and `DBA_OBJECT_TABLES` in a wider scope
- `DBA_TAB_STATISTICS` and `DBA_TAB_HISTOGRAMS` for table statistics
- `DBA_TAB_COL_STATISTICS` for column statistics
- `DBA_INDEXES` and `DBA_IND_STATISTICS` for indexes
- `V$SYSSTAT` for system statistics

Next, let's explore what statistics `DBA_TAB_STATISTICS` contains. `DBA_TAB_STATISTICS` displays optimizer statistics for all tables in the database. Its columns are the same as those in `ALL_TAB_STATISTICS`. Note that if you want to limit to a specific owner, use the view of `USER_TAB_STATISTICS` instead. If you do a `DESC DBA_TAB_STATISTICS` at a `sqlplus` command prompt, you would get a list of the columns for this view. From a statistical point of view, we would be most interested in those columns typed `NUMBER`, since they are used by the CBO to perform cost calculations. These numerical columns are summarized as follows:

- `NUM_ROWS`—Number of rows in the object
- `BLOCKS_NUMBER`—Number of used blocks in the object
- `EMPTY_BLOCKS`—Number of empty blocks in the object
- `AVG_SPACE`—Average available free space in the object
- `CHAIN_CNT`—Number of chained rows in the object
- `AVG_ROW_LEN`—Average row length, including row overhead
- `AVG_SPACE_FREELIST_BLOCKS`—Average freespace of all blocks on a freelist
- `NUM_FREELIST_BLOCKS`—Number of blocks on the freelist
- `AVG_CACHED_BLOCKS`—Average number of blocks in the buffer cache
- `AVG_CACHE_HIT_RATIO`—Average cache hit ratio for the object
- `SAMPLE_SIZE`—Sample size used in analyzing the table

Now, with a given table name (for example, the `Inventories` table of the sample schema `OE`), you can query these statistics by executing a command like `SELECT * FROM DBA_TAB_STATISTICS WHERE TABLE_NAME = 'Inventories'`. If you have set up an Oracle environment by following the Oracle database installation procedure detailed in Chapter 2 of this text, you can try it out on your own setup.

By following a similar procedure outlined as above, you can query the statistics of an index or a table column. The details are omitted here. We now move forward to explaining why CBO statistics are critical and how to avoid unrecoverable consequences resulting from misusing them in the next section.

16.4 PIVOT ROLE OF GATHERING DATABASE STATISTICS TO CBO

The CBO adjusts its cost calculations based on what it “sees” with the current state of a database. It is this dynamic nature of the CBO that makes it more adaptable than its predecessor of the rule-based optimizer. However, it is just this dynamic nature that can make it totally ineffective. That is because the CBO is not able to “think” like a human being. It takes *input* and generates *output*—just as simple as that. The input to the CBO is the collection of statistics at various levels of a database, from an entire database to an entire or only a few schemas, or to just a few tables that are heavily

pounded from time to time. If the CBO is given stale statistics, it won't generate dependable output—the optimum execution plan for a SQL statement. The rule of “GIGO (garbage in and garbage out)” still governs here. This is true with every piece of software that what it outputs depends on what input it is given.

The process of generating database statistics is called *gathering* or *collecting* database statistics. The hardest part of gathering database statistics is not learning the syntax of each command that has to be executed in a statistics gathering task, but rather knowing on which parts of the database to run gathering. With a high volume database, one can easily kill the performance of the database if gathering statistics is indiscriminately applied to the entire database on a full scale, because in this case, Oracle seems to be dedicating all the resources available to it to do nothing but to run the statistics gathering job. If you happen to make such a mistake with a high volume database, your nightmare ensues immediately: You can't easily stop it! My experience is that as soon as you start a statistics gathering job, and let's say it would take a long time to complete, Oracle would say “good bye” to you and never listen to you again until it is complete. It would not be that disastrous in a test environment, but it would be devastating in a production environment.

With enough said about the potential negative impact of gathering database statistics on the normal operation of a database, let's seriously consider how one can follow a disciplined approach to gathering database statistics. A suggested procedure contains the following three elements:

- *Scope.* Decide on which tables of the database that gathering statistics needs to be performed more frequently. Do not just run gathering on an entire schema or an entire database. One should distinguish between two types of computing tasks: batch jobs and OLTP applications. For batch jobs, not all tables are hit equally hard. Identify those tables that are hit hardest and limit gathering to them only. The same rule applies to OLTP applications as well except that you might have more tables to run gathering tasks against. Then how would one identify those hard-hit tables? One should rely on rigorous tests with representative workloads.
- *Sample Size.* Decide whether you want to run statistics gathering on an entire ensemble of a table or a schema or a database, or just a small sample of it. Here neither the rule of “the more the merrier” nor the rule of “the less the merrier” applies. It's like polling of the pre-presidential-election in the United States. It's said that every time only a few thousands potential voters were polled, and polling has never failed to predict which candidate would win eventually. The general rule of thumb for deciding on the size of the sample data is between 10% to 20% if the entire data set is sufficiently large.
- *Frequency.* Decide on how frequently you want to initiate statistics gathering tasks. Oracle gathers statistics on an hourly basis by default, but that's not a solution that fits all situations. Batch jobs typically run at night, while OLTP applications typically have their peak hours during day time. These are the factors to take into account when deciding on the frequency of statistics gathering for your Oracle database.

Once you have decided on the above three elements of scope, sample size, and frequency for gathering database statistics, you can either schedule your database statistics gathering job with the EM DBConsole, or run it manually from SQL*Plus. The former method is preferred in production, although it can also be used in test environments when needed at the desired points of time. The next section discusses the various methods of gathering CBO statistics in detail.

16.5 METHODS OF GATHERING CBO STATISTICS

Gathering CBO statistics can be enabled by default as one of the automated maintenance tasks during database creation time. Refer back to Figure 2.8 to see the maintenance tasks scheduled to run by default, including optimizer statistics gathering. You may edit the default settings to suit your needs on the EM DBConsole by following the navigation path of Server -> Query Optimizer/Manage Optimizer Statistics. The detailed steps for configuring automated statistics gathering will not be covered here, as it's obvious from the EM DBConsole. Instead, we will focus on how to manually gather CBO statistics next.

The CBO statistics can be managed manually with the DBS_STATS package. This package contains the following procedures:

- GATHER_INDEX_STATS for index statistics
- GATHER_TABLE_STATS for table, column, and index statistics
- GATHER_SCHEMA_STATS for statistics of all schema objects
- GATHER_SYSTEM_STATS for system statistics

There also are two additional procedures, GATHER_DICTIONARY_STATS and GATHER_DATABASE_STATS, for gathering statistics for all dictionary objects and all objects in a database, respectively. But we are less concerned with them, since they are less application specific.

To gather CBO statistics on an entire schema, use the following command:

```
Execute DBMS_STATS.GATHER_SCHEMA_STATS
('<schema_name>', DBMS_STATS.AUTO_SAMPLE_SIZE);
```

Note the use of the parameter DBMS_STATS.AUTO_SAMPLE_SIZE, which will let Oracle determine the proper sample size.

To gather CBO statistics at the table level for certain columns only, use the following command:

```
Execute DBMS_STATS.GATHER_TABLE_STATS
('<schema_name>', '<table_name>',
method_opt => 'FOR COLUMNS (<c1>, <c2>, ..., <cn>)');
```

Note `DBMS_STATS.AUTO_SAMPLE_SIZE` is not specified but actually used by default as the value for the parameter `estimate_percent`. If the `method_opt` parameter is not specified, it defaults to `FOR ALL COLUMNS SIZE AUTO`. The part of `FOR ALL COLUMNS` means statistics will be gathered on all columns, while the part of `SIZE AUTO` means Oracle determines the columns to gather histograms based on data distribution and the workload of the columns. One can also specify `FOR ALL INDEXED COLUMNS` to exclude the non-indexed columns, but it may skew the statistics for a table.

To gather CBO statistics at the index level, use the following command:

```
Execute DBMS_STATS.GATHER_INDEX_STATS
(' <schema_name> ', ' <index_name> ');
```

In this command, the sample size is determined by Oracle as well.

Next, we discuss how to lock and unlock CBO statistics gathered.

16.6 LOCKING AND UNLOCKING CBO STATISTICS

If you want to keep reusing the same statistics known to be good or you do not expect statistics to change significantly, your existing statistics for a table or schema can be locked. Then, no modifications to the statistics can be made until the statistics are unlocked.

The `DBMS_STATS` package provides two procedures for locking (`LOCK_SCHEMA_STATS` and `LOCK_TABLE_STATS`) and two procedures for unlocking statistics (`UNLOCK_SCHEMA_STATS` and `UNLOCK_TABLE_STATS`). Consult the Oracle Database PL/SQL Packages and Types release document to learn how to lock and unlock statistics.

Although it's hard to reason how the CBO reaches an optimum execution plan for a given SQL statement, Oracle has a tool named `EXPLAIN PLAN` for us to create and view *an* execution plan. Note that I emphasized the indefinite article *an* in the preceding sentence, because the execution plan displayed from the `EXPLAIN PLAN` command may not be the actual execution plan used for executing the SQL statement in question. The actual execution plan is stored in the performance view of `V$SQL_PLAN`. However, the `EXPLAIN PLAN` command is indeed a helpful handle that opens the door for us to see what the CBO may do with a given SQL statement. This is the subject of the next section.

16.7 EXPLAIN PLAN—A HANDLE TO CBO

Using the HR sample schema, we can demonstrate how to run the `EXPLAIN PLAN` command for a given SQL statement as follows (note: refer to Appendix B on how to generate `EXPLAIN PLAN`s using Oracle's `AUTOTRACE` feature):


```

-----
SQL> set autotrace traceonly
SQL> Select first_name from HR.employees;

```

107 rows selected.

Execution Plan

Plan hash value: 2228653197

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		107	749	1 (0)	00:00:01
1	INDEX FULL SCAN	EMP_NAME_IX	107	749	1 (0)	00:00:01

Statistics

```

-----
      0 recursive calls
      0 db block gets
      9 consistent gets
      0 physical reads
      0 redo size
2770 bytes sent via SQL*Net to client
596 bytes received via SQL*Net from client
      9 SQL*Net roundtrips to/from client
      0 sorts (memory)
      0 sorts (disk)
    107 rows processed

```

SQL>

The output is simple, but it gives what we are interested in here. It shows the access method of the employee first name data—the index full scan method. Since an explained plan mainly consists of such data access methods, it would be beneficial to have systematic coverage of all common data access methods. The next section explains all data access methods that often appear in explained plans.

16.8 DATA ACCESS METHODS—CBO'S FOOTPRINTS

Although the execution plan obtained with the EXPLAIN PLAN command may not be the exact plan used for executing the same SQL statement, it still has its merits in helping us identify potential problems. EXPLAIN PLAN is most useful in revealing data access methods of:

- *Table Full Scan.* This means that Oracle would read the entire table to find what it is looking for. This might be very bad for very large tables. The causes leading to full table scans could be:

- Missing indexes, which is curable. And in this case, it's not CBO's fault.
- The CBO has decided that it's cheaper to do a full scan rather than fetch with available indexes. This could be okay for small, fairly static tables.
- *Table Access by Index RowId.* A `rowid` uniquely identifies a row within a table. With a given `rowid`, Oracle can look up in which data file, which data block, and where in that data block the row is located. A `rowid` is made available to Oracle through one of the two approaches, either from the statement's `WHERE`-clause or from scanning one or more indexes of a table. *Table access by index rowid* is a data access method that rows are obtained with the `rowids` resulting from scanning a table's indexes.
- *Index Unique Scan.* This occurs when the query is looking for a unique row of a table and the query's search conditions specified in its `WHERE`-clause match an available unique index. This is exactly what a unique index is for and it's always good.
- *Index Range Scan.* This occurs when the query is looking for a range of rows as defined by the conditions in its `WHERE`-clause. In general, this is what the CBO is supposed to do with a given range-based query. However, this could happen with unique or non-unique indexes. Whether it's efficient or not should be evaluated with the actual elapsed time of the query, which can be easily identified in an AWR report.
- *Index Skip Scan.* This is a new feature with Oracle 10g. It allows a query to be executed more efficiently than a full table scan by skipping the leading part of an index and using only the last part of the index, which happens to match the query's search conditions in its `WHERE`-clause. The efficiency of this index data access method depends on the cardinality of the leading part of the index that is skipped. The fewer rows skipped due to the index's mismatching leading part, the smaller the impact of the mismatch between the query's search conditions and the mismatching index. However, do not let this lead you into believing that the order of the indexed columns does not matter anymore. Fix the mismatched index or add a new one, instead.
- *Index Fast Full Scan.* This method means that the table is not touched or that the query could be satisfied with all the columns indexed. This would be the case with either covering indexes or IOTs. There is nothing we should complain about in this case.

Note that Oracle provides more finer-grained tools such as SQL Trace and TKPROF, for example, for digging deeper into how data is fetched at the block level. However, one rarely needs to delve so deeply. Most SQL performance tuning problems can be resolved with AWR reports and built-in auto-tune features of the EM DBConsole. Those features have been beefed-up a lot by Oracle from release to release.

16.9 LOOKING UP CBO'S PLAN HIDDEN IN V\$SQL_PLAN

An execution plan alone does not distinguish well-tuned statements from those that perform poorly. For example, an `EXPLAIN PLAN` showing that an index is used does

not necessarily mean that the statement runs efficiently. Sometimes indexes are extremely inefficient. In this case, you should examine both the columns of the index being used and their selectivity as a fraction of table being accessed. Whether a query is performed efficiently can be easily checked with an AWR report. However, an AWR report may not have the execution plan of a query available in the report. In this case, you have to look it up from the V\$SQL_PLAN view.

The V\$SQL_PLAN view contains the execution plan for every statement stored in the cursor cache. Its definition is similar to the PLAN_TABLE. The V\$SQL_PLAN_STATISTICS view provides the actual execution statistics for every operation in the plan, such as the number of output rows and elapsed time. All statistics, except the number of output rows, are cumulative. The statistics in V\$SQL_PLAN_STATISTICS, however, are available only for cursors that have been compiled with the STATISTICS_LEVEL initialization parameter set to ALL.

In addition, the V\$SQL_PLAN_STATISTICS_ALL view enables side-by-side comparisons of the estimates that the optimizer provides for the number of rows and elapsed time. This view combines information from both V\$SQL_PLAN and V\$SQL_PLAN_STATISTICS. Feel free to explore those three views to get a better feel about what information they provide in analyzing a slow query. But still, the information provided in an AWR report would be sufficient most of the time. And if there is a need for digging deeply into those views, run a DESC command against each of them to learn what columns each view has, and then select the relevant columns.

16.10 WHEN CBO MAY GENERATE SUBOPTIMUM EXECUTION PLANS

Will the CBO always generate an optimum execution plan for every SQL statement? More than often it does, but not always. Here are some of my experiences with some real products to share with you:

- The CBO would fail to generate optimum execution plans for many statements if the statistics are stale. Empirically, whenever a database seems to be slow, the first thing to check out is whether the statistics are up-to-date. If not, manually run a statistics gathering job and see if the problem persists.
- The CBO would fail to generate optimum execution plans for the queries that are not indexed or not indexed properly even if the statistics are up-to-date. Note that the CBO does not have the ability to generate indexes per se. Those problems are fixable with proper indexes.

Keep in mind that the word *optimum* in the phrase “an optimum execution plan” is a misnomer. It’s accurate only with all given conditions and assumptions met. A slow query does not always indicate a non-optimum execution plan. There might be only

one execution plan that is optimum, but the factors of making a statement slow are abundant. There are two approaches to determining those affecting factors: the heavyweight way and the lightweight way. The heavyweight way is to keep querying all those static or dynamic data dictionary views, do your own analysis and arrive at your conclusions. That approach is perfectly okay if you have been doing things like that for many years and you still enjoy doing it. But for most of us who might have only a fraction of time that can be allocated to working on Oracle performance and scalability tunings in addition to many other duties, a better, lightweight approach is to rely on AWR reports and the built-in auto-tune features in Oracle for solving our Oracle performance and scalability issues. We have introduced AWR reports in Chapter 11, and Oracle's auto-tune features will be introduced in a later chapter.

16.11 SUMMARY

This chapter revealed the inner workings of the Oracle CBO as a preparation for discussing SQL tunings in the next chapter. I also emphasized that in general it's hard to outsmart the CBO. So instead of feeding CBO with poorly designed SQLs and then trying to influence Oracle on how those SQLs should be executed, one should focus on application logic design so that complex SQLs can be broken up into simpler ones to be executed more efficiently by Oracle. Keep in mind that good application design with efficient implementations such as array processing is the key to achieving high performance and high scalability for many of the large-scale enterprise applications.

Application developers and test engineers should have a modest understanding of how Oracle's cost-based optimizer works behind the scene to make SQLs run as efficiently as possible. Having learned how an Oracle CBO works behind the scene to help make every SQL statement run as fast as possible, in the next chapter, we will discuss how we can tune SQLs to the best of what a CBO can do for an Oracle-based enterprise application in terms of performance and scalability.

RECOMMENDED READING

The Part IV, "Optimizing SQL Statements," of the following Oracle document describes in detail about Oracle query optimizer as well as EXPLAIN PLANS:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11gRelease 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

The following texts cover Oracle query optimizer and EXPLAIN PLANS in detail as well:

Alice Rischert, *Oracle SQL by Example*, 4th edn, Prentice Hall, Boston, 2009.

Jason Price, *Oracle Database 11g SQL*, McGraw-Hill Osborne Media, New York, 2007.

Guy Harrison, *Oracle Performance Survival Guide: A Systematic Approach to Database Optimization*, Prentice Hall, Boston, 2009.

EXERCISES

- 16.1 Describe conceptually what the CBO statistics are. How can you access the statistics information for tables, columns, and indexes?
- 16.2 What are the potential consequences when database statistics do not exist or become stale?
- 16.3 How would you determine the scope, sample size, and frequency when you define and run a database statistics gathering job?
- 16.4 Describe the situations for which you might need to lock or unlock database statistics.
- 16.5 What information can you get from an EXPLAIN PLAN? Can you always depend on the EXPLAIN PLAN method to determine the optimum execution plan for a SQL statement?
- 16.6 What are database data access methods? Why are they important in determining the performance and scalability of an Oracle database?
- 16.7 Is a full table scan always bad? Use several examples to explain your conclusion.
- 16.8 How many database access methods are based on indexing? Explain the difference between the index range scan and index unique scan data access methods.
- 16.9 What's the difference between an index skip scan and an index fast full scan? How could you anticipate if an index skip scan or index fast full scan may occur.
- 16.10 Is the data access method of *table access by index rowid* an efficient data access method?
- 16.11 Which V\$ view contains a CBO's execution for a SQL? How do you look it up?
- 16.12 Under what circumstances does the CBO possibly generate suboptimal execution plans?

17

Oracle SQL Tuning

Art is the desire of a man to express himself, to record the reactions of his personality to the world he lives in.

—Amy Lowell

Oracle SQL tuning is a broad subject, which deserves a fully dedicated text on its own. In fact, both Oracle product documentations and some well-written texts are available for understanding Oracle SQL tuning. To clarify further, our coverage of SQL tuning in this chapter will be relatively light for a few reasons, including:

- If you are developing an Oracle-based enterprise application, it's likely that you have an application tier. And if your application tier is coded in Java, it's likely that you do not handwrite SQLs yourself. Instead, SQLs are issued from some standard Object-Relational Mapping (ORM) technologies such as *Hibernate* as we demonstrated in Chapter 15 of this text. In this case, you have less control of how SQLs are composed. Optimizing SQL has already been taken into account in the ORM products you use.
- Oracle SQLs can be tuned from multiple approaches. For example, you can turn on SQL tracing, create an EXPLAIN Plan to understand the data access paths involved, or generate an AWR report to capture hot SQLs, or even turn to Oracle's built-in SQL auto-tune advisors. Much of this has been covered in some other chapters of this text.

Given the context mentioned above, this chapter focuses on understanding conceptually two specific areas of SQL tuning: tuning Oracle join and subquery SQLs. The purpose of this chapter is to call out a set of principles and practices that can help yield highest possible returns for the performance and scalability of your Oracle-based application. If you need to learn more about SQL tunings, refer to the texts recommended at the end of this chapter.

Let’s begin with tuning joins first in the next section.

17.1 TUNING JOINS

Joins are SQL statements that form an aggregated result set from multiple tables. How the tables are joined is governed by one or more *join conditions*. A join condition is an expression that compares a common set of columns between two tables, namely, the driving (left) table and the driven (right) table. If more than two tables are involved, the result table becomes the new driving table, which is joined with the next table until no more tables remain.

To illustrate the types of joins, the sample schema Scott is a perfect choice because of its simplicity and not a lot of data, which made it more manageable. See Tables 17.1 to 17.7 for more details about this schema (note: the BONUS table has no data in it and is empty).

It’s seen that the Scott schema has four tables: EMP, DEPT, BONUS, and SALGRADE. The EMP table has the columns of EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, and DEPTNO, whereas the DEPT table has the columns of DEPTNO, DNAME, and LOC. It’s seen that these two tables share the same column of DEPTNO. The EMP table has 14 rows, while the DEPT table has 4 rows. Let’s use these two tables to illustrate the types of joins that Oracle supports as follows (note: you can manually verify the result of each join if you like):

- *Equijoins*. An equijoin is a join that has an equality operator in its WHERE clause. With the Scott schema, an equijoin of finding all employees working in the SALES department can be constructed as:

Table 17.1 SCOTT.EMP Table

COLUMN	NULL?	TYPE
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

Table 17.2 SCOTT.DEPT Table

COLUMN	NULL?	TYPE
DEPTNO	NOT NULL	NUMBER (2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

Table 17.3 SCOTT.BONUS Table

COLUMN	NULL?	TYPE
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
SAL		NUMBER
COMM		NUMBER

Table 17.4 SCOTT.SALGRADE Table

COLUMN	NULL?	TYPE
GRADE		NUMBER
LOSAL		NUMBER
HISAL		NUMBER

Table 17.5 Records of SCOTT.EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	20	
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Table 17.6 Records of SCOTT.DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Table 17.7 Records of SCOTT.SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

```
SQL> select ename, d.dname from emp e, dept d where e.deptno =
d.deptno and d.dname = 'SALES';
```

```
ENAME      DNAME
-----
WARD       SALES
TURNER     SALES
ALLEN      SALES
JAMES      SALES
BLAKE      SALES
MARTIN     SALES
6 rows selected.
```

- *Self Join.* A self join joins a table with itself. For example, the following self join lists the “work for” relationship based on the EMP table alone:

```
SQL> select e1.ename || 'works for' || e2.ename from emp e1,
emp e2 where e1.mgr = e2.empno;
```

```
E1.ENAME || 'WORKSFOR' || E2.ENAME
-----
FORD works for JONES
SCOTT works for JONES
TURNER works for BLAKE
ALLEN works for BLAKE
WARD works for BLAKE
JAMES works for BLAKE
MARTIN works for BLAKE
MILLER works for CLARK
ADAMS works for SCOTT
BLAKE works for KING
JONES works for KING
CLARK works for KING
SMITH works for FORD
13 rows selected.
```

- *Inner Joins (or simple joins).* An inner join creates a new result table by combining column values of two tables (driving and driven tables) strictly based on the join conditions or join-predicates. This is the most common type of joins. The following SQL statement constitutes an inner join:

```
SQL> select ename, d.dname from dept d, emp e where
d.deptno = e.deptno;
```

ENAME	DNAME
CLARK	ACCOUNTING
KING	ACCOUNTING
MILLER	ACCOUNTING
JONES	RESEARCH
FORD	RESEARCH
ADAMS	RESEARCH
SMITH	RESEARCH
SCOTT	RESEARCH
WARD	SALES
TURNER	SALES
ALLEN	SALES
JAMES	SALES
BLAKE	SALES
MARTIN	SALES

14 rows selected.

- *Outer Joins.* An inner join does not include rows that don't match the join condition, whereas an outer join can contain those extra rows that don't match the join condition because the column of a table in the join equality comparator is either empty or null. Outer joins can be further classified into left outer joins or left joins, right outer joins or right joins, and full outer joins or full joins. With a left join, the result table always contains all rows of the "left" table A, even if the join condition does not find any matching record in the "right" table, in which case a NULL will be returned and placed in the result table as the value of the column of the joined table. The previous inner join becomes a left join with the following change:

```
select ename, d.dname from dept d, emp e where d.deptno
= e.deptno (+) ;
```

The (+) signifies adding the row of DEPT even if no matching row can be found in table EMP. This left join returned 15 rows. A right join is just the reverse of a left join with two tables swapped. A full join combines the results of both left and right joins. For this reason, some databases do not support right and full

joins, because they both can be accomplished with left joins. Note that a left join is more commonly coded as:

```
select ename, d.dname from dept d left join emp e on
d.deptno = e.deptno;
```

The keyword `left` can be replaced with `right` or `full` to designate a right or full join. The right join and full join returned 14 and 15 rows, respectively.

- *Cartesian Joins.* A Cartesian join or a cross join is a join when two tables have no join conditions. In this case, every row from one of the source tables is joined with every row from the other table, creating the Cartesian product of the rows of the two tables. With the same example, a Cartesian join would look like:

```
select ename, d.dname from emp e, dept d;
```

This join returned 56 (14×4) rows.

- *Antijoins.* An antijoin returns the rows from the left side of the predicate for which there are no corresponding rows on the right side of the predicate. Using the same example, an antijoin is expressed as follows, which returned one row of OPERATIONS:

```
SQL> select d.dname from dept d where deptno NOT IN (select
deptno from emp) ;
```

```
DNAME
-----
OPERATIONS
```

Note that it's kind of stretching the sense to call it a *join* because the second table is actually replaced with a subquery. In this case, the columns of the EMP table is not visible to the SELECT part of the main SQL, while in a true join, all columns of all tables are visible to the SELECT part of the main SQL.

- *Semijoin.* A semijoin returns rows that match an EXISTS subquery without duplicating rows from the left side of the predicate when multiple rows of the right side satisfy the criteria of the subquery. Using the same example, a semijoin can be expressed as:

```
SQL> select d.dname from dept d where exists (select deptno
from emp) ;
```

Once again, a semijoin looks more like a subquery than a join. The factors related to optimizing joins are as follows:

- *Join Order.* The Oracle optimizer decides which two tables to start first. Then it uses the result set as the new table, which is joined with another table.

This process is repeated until no more tables exist and all the tables are processed.

- *Access Paths.* Oracle decides an access path to retrieve data from each table in the join statement.
- *Join Algorithms.* Oracle decides how a join operation is performed on each pair of inner/outer tables. Common join algorithms include hash joins, nested loop joins, and sort merge joins. We will explain each of these join algorithms briefly next.

A `hash join` is applied to joining large tables. The smaller table is used as the driver to build a hash table on the join key in memory. The larger table is scanned with the help of the hashed keys to find the joined rows. This method is favored over other join algorithms when the smaller table can fully fit in memory, and then the cost is limited to a single read pass over the data for the two tables. Because hash value comparison can only work with equality or inequality join conditions, hash joins don't apply to joins with inequality join conditions such as expressed with inequality comparators of `<`, `>`, `<=` and `>=`.

A `nested loop join` is favored when only small subsets of data are joined and the join condition implies an efficient access path for the second table. It works best if the access path to the inner table depends on the outer table and the outer table is used as the driving table. If the access path to the inner table is independent of the outer table, then essentially the two tables are independent of each other and a hash join is a better choice.

A `sort merge join` can also join two independent tables. It performs best if the tables are already sorted and the join condition is an inequality condition like `<`, `<=`, `>`, or `>=`. Sort merge joins explain why optimizers keep track of the sort order of tables for obvious reasons.

It's important to draw a clear distinction between join types and join algorithms. Join types are defined based on the result set, while join algorithms refer to the join implementation schemes for given join types, such as inner and outer joins. What join algorithms are used is largely determined by the CBO and the expected efforts for manually tuning joins are minimal. So my recommendation for tuning joins in general is to feed the CBO with good statistics and let CBO do the tuning for you.

Joins are closely related to subqueries to some extent, as we've already seen with antijoins and semijoins as described above. The next section discusses what subqueries are and how they are further related to joins.

17.2 TUNING SUBQUERIES

Subqueries are just another way to create result tables from multiple source tables by embedding queries at various positions of the main query, for example, between the keywords of `SELECT` and `FROM`, in the `WHERE`-clause in conjunction with the

keywords of IN, EXISTS, NOT IN, and NOT EXISTS, and so on. Subqueries can also be used with INSERT, UPDATE, and DELETE statements in addition to SELECT statements.

Subqueries are closely related to joins in the sense that they all create result tables from multiple source tables, and that a subquery can often be rewritten with a corresponding join to accomplish the same result. However, some subqueries may not have join counterparts to accomplish the same results.

With regard to subqueries, first, one needs to draw a distinction between uncorrelated and correlated subqueries. With uncorrelated subqueries, none of the columns of the main table appear in the WHERE-clause of the subquery. In contrast, with correlated subqueries, one or more of the columns of the main table appear in the WHERE-clause of the subquery. Using the Oracle HR sample schema, we can conjure up an uncorrelated subquery to retrieve a list of all employees who work in the IT department as follows:

```
SELECT first_name, last_name FROM Employees WHERE
       department_id IN (SELECT department_id FROM
       Departments WHERE
       department_name = 'IT');
```

Using the same HR sample schema, a correlated subquery for retrieving each employee's salary relative to his/her department average and company-wide average may look like this:

```
SELECT first_name, last_name, salary, (SELECT avg
(salary)
FROM Employees WHERE department_id=e.department_id)
       as de_avg, (select avg(salary) FROM Employees) as
       comp_avg
FROM Employees e;
```

The above correlated subquery uses the Employees table for both main query and subqueries. The following is an example of another correlated subquery that doesn't use the same Employees table in both main query and subqueries:

```
SELECT first_name, last_name, department_id, manager_id
FROM Employees e WHERE department_id IN (SELECT
department_id
FROM Departments WHERE manager_id = e.manager_id);
```

It returned 32 rows (sum of all department heads' direct reports) from 27 rows of the Departments table and 107 rows of the Employees table with the default HR Schema from Oracle 11g. The above subquery is equivalent to the following inner join:

```
SELECT first_name, last_name, e.department_id,
       e.manager_id
FROM Employees e, Departments d WHERE e.department_id =
       d.department_id AND e.manager_id = d.manager_id);
```

Note that a subquery is invisible to its main query, and therefore, the main query cannot reference any columns in the subquery. However, a main query is visible to its subquery so that the subquery can reference its main query's columns, making itself a correlated subquery.

Next, let's use some simple examples to illustrate the performance of subqueries versus joins and IN versus EXISTS. Due to the simplicity of the examples, the performance numbers we will see are at best ballpark numbers.

17.3 CASE STUDY: PERFORMANCE OF SUBQUERY VERSUS JOIN

In practice, if you have control of how SQLs are issued with your Oracle-based application, should you use a subquery or a join to accomplish the same task? Let's do a simple test with the sample schema OE (Order Entry) to help answer this question. Before we try it out, we need to execute the following commands after getting at the SQL*Plus command prompt:

- Set timing on (note: this command turns timing on for measuring the execution of a SQL. See Appendix B to learn more about this timing command).
- Set linesize 1000 (this command would extend the default line size to avoid breaking a line).
- Set autotrace on (this command allows an EXPLAIN PLAN to be output. If you see an error complaining about Cannot find the Session Identifier, ignore it. See Appendix B to learn more about this autotrace command).

Let's say we would like to know the products in inventory by querying two tables: Product_Descriptions and Inventories. First, we may just want to try it out with the following join SQL:

```
SQL> SELECT count (*) FROM Product_Descriptions p,
Inventories i WHERE p.product_id = i.product_id AND
i.quantity_on_hand > 0;
```

After executing the above SQL, you may get something similar to the following output:

```
COUNT(*)
-----
33330

Elapsed: 00:00:00.03

Execution Plan
-----

Plan hash value: 683685205

-----
| Id | Operation          | Name           | Rows  | Bytes | Cost (%CPU) | Time |
-----|-----|-----|-----|-----|-----|-----|
|  0 | SELECT STATEMENT   |                |      1 |      1 |      14 (8) | 00:00:01 |
|  1 | SORT AGGREGATE     |                |      1 |      1 |      12      |          |
|*  2 | HASH JOIN          |                | 33219 | 389K |     14 (8) | 00:00:01 |
|*  3 | TABLE ACCESS FULL | INVENTORIES    | 1107  | 8856 |      3 (0) | 00:00:01 |
|  4 | INDEX FAST FULL SCAN| PRD_DESC_PK    | 8640  | 34560 |     10 (0) | 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
2 - access("P"."PRODUCT_ID"="I"."PRODUCT_ID")
3 - filter("I"."QUANTITY_ON_HAND">0)
```

And then try it out with the following subquery SQL:

```

SELECT count (*) FROM Product_Descriptions WHERE product_id IN
(SELECT product_id FROM Inventories i WHERE product_id = i.pro
duct_id AND i.quantity_on_hand > 0);

COUNT(*)
-----
        6240

Elapsed: 00:00:00.01

Execution Plan
-----
Plan hash value: 3959551680

-----
| Id | Operation                      | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT                |                |      1 |    12 |    14 (8)| 00:00:01 |
|  1 |   SORT AGGREGATE                |                |      1 |    12 |           |           |
* |  2 |    HASH JOIN RIGHT SEMI         |                | 6240 | 74880 |    14 (8)| 00:00:01 |
* |  3 |      TABLE ACCESS FULL         | INVENTORIES    | 1107 |  8856 |      3 (0)| 00:00:01 |
|  4 |        INDEX FAST FULL SCAN     | PRD_DESC_PK    | 8640 | 34560 |     10 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
 2 - access("PRODUCT_ID"="PRODUCT_ID")
 3 - filter("I"."QUANTITY_ON_HAND">0)

```

Note that the join SQL and subquery SQL did not return the same number of rows (33330 versus 6240). That's because the join SQL returns all products in all warehouses (even for the same product), while the subquery SQL just returns products in inventory. The underlying cause is that the common column of `product_id` is neither a primary key nor a foreign key. Therefore, these two SQLs are not the same. One more thing we need to explain is we used `count (*)` to avoid lengthy text output, which would skew the execution time spent on the Oracle Server significantly. We could have executed the command of `set termout first` and then spool the output to a text file, but that would skew the test results as well as the time spent on writing the returned textual result to a local drive may far exceed the time spent on the Oracle Server. It's always necessary to make sure first that a test is planned carefully so that the test data would make sense and match with the goal of the test set out.

To correct the problem, we need to add `DISTINCT product_id` inside `count` as shown in the join SQL below:

```

SELECT count (DISTINCT p.product_id) FROM Product_Descriptions p,
Inventories i WHERE p.product_id = i.product_id AND
i.quantity_on_hand > 0;

```



```

COUNT(DISTINCTP.PRODUCT_ID)
-----
                208

```

Elapsed: 00:00:00.03

Execution Plan

Plan hash value: 3050726013

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	13	16 (19)	00:00:01
1	SORT AGGREGATE		1	13		
2	VIEW	VW_DAG_0	208	2704	16 (19)	00:00:01
3	SORT GROUP BY NOSORT		208	2496	16 (19)	00:00:01
4	MERGE JOIN		33219	389K	16 (19)	00:00:01
5	SORT JOIN		1107	8856	4 (25)	00:00:01
* 6	TABLE ACCESS FULL	INVENTORIES	1107	8856	3 (0)	00:00:01
* 7	SORT JOIN		8640	34560	11 (10)	00:00:01
8	INDEX FAST FULL SCAN	PRD_DESC_PK	8640	34560	10 (0)	00:00:01

Predicate Information (identified by operation id):

```

6 - filter("I"."QUANTITY_ON_HAND">0)
7 - access("P"."PRODUCT_ID"="I"."PRODUCT_ID")
   filter("P"."PRODUCT_ID"="I"."PRODUCT_ID")

```

And then try it out with the following subquery SQL with DISTINCT product_id added inside count:

```

SELECT count (DISTINCT product_id) FROM Product_Descriptions WHERE
product_id IN (SELECT product_id FROM Inventories i WHERE
product_id = i.product_id AND i.quantity_on_hand > 0);

```

```

COUNT(DISTINCTPRODUCT_ID)
-----
                208

```

Elapsed: 00:00:00.03

Execution Plan

Plan hash value: 1567148646

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	13	15 (14)	00:00:01
1	SORT AGGREGATE		1	13		
2	VIEW	VM_NWVW_1	208	2704	15 (14)	00:00:01
3	HASH GROUP BY		208	2496	15 (14)	00:00:01
* 4	HASH JOIN RIGHT SEMI		6240	74880	14 (8)	00:00:01
* 5	TABLE ACCESS FULL	INVENTORIES	1107	8856	3 (0)	00:00:01
6	INDEX FAST FULL SCAN	PRD_DESC_PK	8640	34560	10 (0)	00:00:01

Predicate Information (identified by operation id):

```

4 - access("PRODUCT_ID"="PRODUCT_ID")
5 - filter("I"."QUANTITY_ON_HAND">0)

```

Note that both SQLs returned with a same execution time of 0.03 seconds or 30 milliseconds. That's probably because both SQLs were executed with same data access paths: One index fast full scan on the driving table and one full table scan on the driven table. It's also interesting to notice that internally the subquery SQL was executed with a `HASH JOIN RIGHT SEMI`. However, after the same SQLs were executed multiple times, the join SQL returned with an execution time of 30 milliseconds, while the subquery SQL returned with an execution time of 10 milliseconds. That's probably because the join SQL returned a total row of 33219 with a `MERGE JOIN` while the subquery returned a total row of 6240 with a `HASH JOIN RIGHT SEMI`, which might actually be fetched from the buffer cache. If both SQLs returned the same number of rows before `DISTINCT` was applied, they would have performed the same. So whether a subquery and a join perform similarly depends on whether the join predicate is based on columns that are primary/foreign keys. If that's the case, both would perform the same, otherwise, the subquery could potentially perform better than its join counterpart. Also, note a major difference between a subquery and a join that one can return columns of both tables with a join SQL, while one can return only the columns of main query—not the subquery—of a subquery SQL. So the first rule on deciding which one to use should be based on whether you need columns from both tables.

17.4 CASE STUDY: PERFORMANCE OF IN VERSUS EXISTS

Regarding subqueries, should one use `IN` or `EXISTS` when composing a subquery? Continuing with the last subquery introduced in the previous section, the same query was executed with `IN` replaced with `EXISTS` as follows:

```
SQL> SELECT count (DISTINCT product_id) FROM Product_Descriptions p WHERE EXISTS
(SELECT product_id FROM Inventories WHERE product
_id = p.product_id);
```

```
COUNT(DISTINCTPRODUCT_ID)
-----
208
```

Elapsed: 00:00:00.01

Execution Plan

Plan hash value: 1213464253

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	8	14 (8)	00:00:01
1	SORT GROUP BY		1	8		
* 2	HASH JOIN RIGHT SEMI		6240	49920	14 (8)	00:00:01
3	INDEX FAST FULL SCAN	INVENTORY_IX	1112	4448	3 (0)	00:00:01
4	INDEX FAST FULL SCAN	PRD_DESC_PK	8640	34560	10 (0)	00:00:01

Predicate Information (identified by operation id):

```
2 - access("PRODUCT_ID"="P"."PRODUCT_ID")
```

It is seen that the subquery with EXISTS returned the same result of 208 rows with an execution time of 10 milliseconds. Once again, the same SQL was executed again and an execution time of 10 milliseconds was returned, being the same as the execution time of 10 milliseconds for the first run. Comparing the first run of the EXISTS-based subquery with that of the IN-based subquery, it seems that EXISTS-based subquery was faster than the IN-based subquery. My recommendation is to favor EXISTS if you can accomplish your task with either IN or EXISTS. Software development has been trending toward more convention-centric, namely, if you can accomplish the same task with multiple options, then one should pick a convention and then stick to it. I'd like to emphasize again that Oracle has a very elaborate built-in SQL optimizer, and how a SQL is twisted by rewriting it this way or that way may not always yield a noticeable performance gain at the end.

17.5 CASE STUDY: A SQL TUNING YIELDED A 12X PERFORMANCE GAIN

Drastic SQL query performance improvement typically can be achieved with proper indexing. However, sometimes, the Oracle CBO may not be smart enough to arrive at what indexes to use when the SQL statements are too convoluted. In such cases, full table scans occur, which is known to be bad for the efficient execution of the SQLs in question most of the time. Then, in order to achieve the best execution path, manual SQL rewriting is necessary. This case study demonstrates how rewriting a SQL statement yielded a 12X performance gain for a real product catalog used in a large enterprise application.

The original SQL was a hierarchical SQL as shown below:

```
SELECT package_id, packaged_product_id, is_subj_to_
availability, product_name, is_package_flg
FROM retailer_pkg_item_price, product
WHERE packaged_product_id = product_id
START WITH package_id = :1 AND pc_ret_id = :2
CONNECT BY PRIOR packaged_product_id = package_id
AND pc_ret_id = :3;
```

Note that the START WITH segment in the above SQL specifies a condition that identifies the row(s) to be used as the root(s) of a hierarchical query, whereas the CONNECT BY PRIOR segment in the above SQL specifies a condition that identifies the relationship between parent rows and child rows of the hierarchy.

The above SQL was called in an API of checkProductAvailability coded in Java. With customer data fully populated in an Oracle 9i database, my performance tests showed that the API call took over three seconds, which was considered too long. In order to tune this SQL, I first obtained its EXPLAIN PLAN as shown below:

EXPLAIN PLAN

Operation	PHV/Object Name	Rows	Bytes	Cost
SELECT STATEMENT	----- 1930379050 -----			355
CONNECT BY WITH FILTERING				
FILTER				
COUNT				
HASH JOIN		196K	9M	355
TABLE ACCESS FULL	PRODUCT	12K	388K	31
TABLE ACCESS FULL	RETAILER_PKG_ITEM_PR	196K	3M	170
HASH JOIN				
CONNECT BY PUMP				
COUNT				
HASH JOIN		196K	9M	355
TABLE ACCESS FULL	PRODUCT	12K	388K	31
TABLE ACCESS FULL	RETAILER_PKG_ITEM_PR	196K	3M	170

As is seen, the two tables, PRODUCT and RETAILER_PKG_ITEM_PR, were accessed with full table scans, respectively, despite the fact that the proper index was in place.

During the tuning process, the above SQL query was split into the following two queries in the application implementation. The EXPLAIN PLAN was obtained for each query as shown below.

a) Hierarchical SQL

```

SELECT package_id, packaged_product_id, is_subj_to_
availability
FROM retailer_pkg_item_price
START WITH package_id = :1 AND pc_ret_id = :2
CONNECT BY PRIOR packaged_product_id = package_id
AND pc_ret_id = :3;

```

EXPLAIN PLAN

Operation	PHV/Object Name	Rows	Bytes	Cost
SELECT STATEMENT	----- 2201265882 -----			4
CONNECT BY WITH FILTERING				
NESTED LOOPS				
INDEX RANGE SCAN	PKITPR_UK	1	9	3
TABLE ACCESS BY USER ROWID	RETAILER_PKG_ITEM_PR			
NESTED LOOPS				
BUFFER SORT		1	17	
CONNECT BY PUMP				
TABLE ACCESS BY INDEX ROWID	RETAILER_PKG_ITEM_PR	1	17	4
INDEX RANGE SCAN	PKITPR_UK	1		3

b) Product SQL

```
SELECT product_id, product_name, is_package_flg
FROM product
WHERE PRODUCT_ID = :1;
```

EXPLAIN PLAN

Operation	PHV/Object Name	Rows	Bytes	Cost
SELECT STATEMENT INLIST ITERATOR	---- 3654310918 ----			25
TABLE ACCESS BY INDEX ROWID	PRODUCT	107	2K	25
INDEX RANGE SCAN	PRODCT_PK	107		2

As is seen, the cost of the original SQL was 355, while the costs for the two new SQLs were 4 and 25, respectively. The cost reduction (CR) gained from rewriting the original SQL into two new queries can be calculated as follows:

$CR = 355 / (4 + 25) = 12$

This was a 12X reduction in *cost*. Table 17.8 shows the actual performance improvement for the API of checkProductAvailability with three runs of the same test. The elapsed time of the API was reduced from about 3200 milliseconds to about 260 milliseconds on average, indicating also a 12X performance gain measured from the API elapsed time perspective.

To make sure the SQL rewriting was indeed needed, the original SQL was executed again with an index hint added as follows (note that the index hinted is the same index chosen by the optimizer for executing the rewritten hierarchical SQL that resulted in a cost of 4 as illustrated in the EXPLAIN PLAN for that SQL):

```
SELECT /* + (RETAILER_PKG_ITEM_PRICE PKITPR_UK) */
package_id, packaged_product_id, is_subj_to_
availability, product_name, is_package_flg
FROM retailer_pkg_item_price, product
WHERE packaged_product_id = product_id
START WITH package_id = '12599' AND pc_ret_id = '63'
CONNECT BY PRIOR packaged_product_id = package_id AND
pc_ret_id = '63';
```

Table 17.8 Actual Performance Improvement for the API of checkProductAvailability

Run #	with Original SQL (ms)	with New SQLs (ms)	Gain (times)
1	3204	257	12x
2	3227	261	12x
3	3163	263	12x

Operation	Cost	IO Cost	CPU Cost	Cardinality	Bytes
SELECT STATEMENT	365.0		365	196896	8663424
HASH JOIN					
CONNECT BY (WITH FILTERING)					
CONNECT BY PUMP					
COUNT					
HASH JOIN	365.0		365	196896	8663424
TABLE ACCESS (FULL)	37.0		37	17903	483381
TABLE ACCESS (FULL)	170.0		170	196896	3347232
FILTER					
COUNT					
HASH JOIN	365.0		365	196896	8663424
TABLE ACCESS (FULL)	170.0		170	196896	3347232
TABLE ACCESS (FULL)	37.0		37	17903	483381

Figure 17.1 EXPLAIN PLAN of a convoluted SQL with an index hint added.

Figure 17.1 illustrates the EXPLAIN PLAN of the above SQL. It is seen that the hinted index was not taken by the optimizer and the cost was 365, which was even slightly higher than the cost value of 355 for the original SQL. The development team was assured that rewriting the original SQL was the right action to take, and it was implemented accordingly.

17.6 SUMMARY

In this chapter, we focused on understanding Oracle joins and subqueries that are most typical types of SQLs that need to be tuned. Keep in mind that Oracle rewrites a SQL automatically internally to help yield the best possible performance, so it may not be necessary for you to rewrite the poorly performing SQLs you identified yourself. Remember if Oracle can find a hot SQL, it has recommendations for you as well with predicted quantitative improvements if implemented. This has worked quite well for me in a few cases.

Whether it's a join or a subquery, adding indexes to database tables may play a critical role in speeding up query executions. Being able to understand various indexing schemes and their potential impacts on SQL query performance is a very necessary skill not only for full-time DBAs but also for both developers and performance engineers. Indexing will be covered in the next chapter.

RECOMMENDED READING

To learn more about Oracle joins and subqueries, refer to Chapter 9, "SQL Queries and Subqueries," of the following Oracle documentation:

Oracle Corp, *Oracle Database SQL Language Reference*, 11g Release 1 (11.1) B28286-05 (1446 pages), September 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28286.pdf. This document is a complete reference of Oracle SQL.

Part IV, “Optimizing SQL Statements,” of the following Oracle documentation describes in detail about optimizing Oracle SQLs:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

The following texts cover Oracle query optimizer and EXPLAIN PLANs in detail as well:

Alice Rischert, *Oracle SQL by Example*, 4th edn, Prentice Hall, Boston, 2009.

Jason Price, *Oracle Database 11g SQL*, McGraw-Hill Osborne Media, New York, 2007.

Guy Harrison, *Oracle Performance Survival Guide: A Systematic Approach to Database Optimization*, Prentice Hall, Boston, 2009.

EXERCISES

- 17.1 Explain conceptually the differences among various join schemes. How do you accomplish a right join or a full join with a left join?
- 17.2 What are the potential consequences when database statistics do not exist or become stale with an Oracle-based application that depends on a lot of complicated joins and subqueries to function?
- 17.3 Under what circumstances should you choose a join or a subquery when you compose an Oracle SQL query?

18

Oracle Indexing

Science is what you know; philosophy is what you don't know.
—**Bertrand Russell**

Very often, a simple index could be as powerful as a silver bullet in resolving an Oracle performance and scalability issue. I have had many such experiences, and some of which are presented in the next part of this text as real world case studies. Yet, many times, people mis-judge, for example, an Oracle performance issue that could have been resolved with a single proper index decisively could be perceived as a hardware issue and the advice of upgrading the hardware was given “expertly” under a seemingly wise excuse that “hardware is cheaper than development time.” Keep in mind that creating an index may take only a few minutes, while acquiring a new piece of hardware may take weeks or even months because of the tight procurement process that is in place in almost every organization.

In this chapter, we cover some of the most commonly used Oracle indexing schemes that are absolutely necessary for every Oracle-based enterprise application. We even go further into discussing some of the unusual indexing schemes that should be used cautiously if you have to. However, this chapter will not be a lengthy coverage of all details about Oracle indexing for a few reasons:

- If you cannot fix your Oracle performance and scalability issue with simple indexing schemes introduced here, then most likely it's not an indexing issue.

Instead, you should review your data model design and application logic coding and find your silver bullets there.

- Based on my experience, nailing down to a proper index could be a laborious task. Oracle has been improving many auto-tune features, and riding on what Oracle could do for you could make your work a lot easier. And the reason is simple: Oracle has an elaborate SQL optimizer built-in to rely on, but none of us have anything similar in our brains. So if you feel you couldn't do it, try to let Oracle do it for you.

Given what is said above, let's have a practical exploration of the following topics about Oracle indexing:

- Rules of Thumb on Indexing
- Creating and Using Ubiquitous B-Tree Indexes
- Advanced Indexing Scheme I: Covering Indexes versus Index-Organized Tables
- Advanced Indexing Scheme II: Function-Based Indexes (FBIs)
- Unusual Indexing Scheme I: BITMAP Indexes
- Unusual Indexing Scheme II: Reverse Key Indexes
- Unusual Indexing Scheme III: Compressed Composite Indexes
- How to Create Oracle Indexes

Let's start with some rules of thumb on Oracle indexing next.

18.1 RULES OF THUMB ON INDEXING

It might not be uncommon that whenever there is a database performance problem, someone will ask: Do we have indexes missing? It's true that indexes can help speed up search queries significantly if used properly. But on the other hand, indexing may create performance problems as well. In this section, some rules of thumb related to indexing are shared as follows:

- *Order of Indexed Columns Matters.* If you need to create an index for a query which has C1, C2, and C3 columns appearing in its WHERE-clause condition in that order, then create the index `tableName_C1_C2_C3_IX` with the C1 – C3 columns following the same order. This is a full index. It will also cover the queries that partially use C1 or C1 and C2 but not the queries that partially use C2, or C3, or C2 and C3.
- *Do Not Over-Index.* It's very possible that too many indexes have been created over time by developers. Each developer may only focus on what he/she needs without considering reusing some of the existing indexes. Then when it comes to performance testing, one might be confused about why there are so many indexes and some of them are seemingly redundant. Remember that whenever an index is

created, it increases write operations due to that index. For example, inserting a single row into a table with n indexes will result in $n + 1$ changes to the database with an overhead of $n/(n + 1)$. This is why one needs to be prudent when considering adding a new index.

- *Exclude Columns Largely Valued with NULLs.* If a column has too many NULL values, including it in an index is pointless, as NULLs are typically not included in an index. If you have to, then make sure it's well-justified.
- *Full Table Scans Might Work Better Without Indexes sometimes.* This is generally true if a table is small in size and also nearly static. In this case, even if indexes are created, they may just be ignored by the Oracle optimizer when determining the optimum execution plan.
- *Be Conscious About the Index-to-Table Ratio.* If a table has n columns out of which m columns are included in an index, the index-to-table ratio is defined as m/n . It's a no-brainer that this ratio should be kept as small as possible. This ratio should rarely exceed 20% to 30% or even much smaller if the table has more than 10 columns.
- *Select the Proper Index Type.* This is mostly a question of whether one should use a b-tree index or a bitmap index. This is determined by the *selectivity* of the columns to be indexed upon. The selectivity of a column is measured by the ratio of all distinct values divided by the total number of rows. B-tree indexes and bitmap indexes work at the two extreme ends in terms of selectivity, respectively, namely, a b-tree index is more efficient toward higher selectivity, while a bitmap index is more efficient toward lower selectivity. We'll elaborate on this further in the section about bitmap indexes later.

These are just some of the basic rules of thumb on indexing. We'll learn more as we expand into the next few sections about indexing.

18.2 CREATING AND USING UBIQUITOUS B-TREE INDEXES

Oracle b-tree indexes are the most commonly used type of indexes. When people say "index," most likely they mean a b-tree index.

First let's clarify the definition of a b-tree. A b-tree is a data structure. It is defined with the following attributes:

- *Node.* A node is a basic element of a b-tree. The top-most node is called the root of the tree; the nodes at the bottom are called leaves; and the nodes in-between are called internal nodes.
- *Order.* The order of a b-tree defines the maximum number of children per node. Note that the concept of *order* applies to non-leaf nodes only. Or in other words, leaf nodes do not have to obey the same order as for non-leaf nodes do, namely, a leaf node can contain an arbitrary number of entries for itself.

- *Depth.* The depth of a b-tree is the total number of hops from the root to a leaf node along a consecutive path. Depth is also called level.
- *Balance.* This is a measure of the uniformity across all nodes. A b-tree is called a balanced b-tree if every node has the same order and every leaf node has the same depth.

A b-tree search structure is based on a binary algorithm extended to m -ternary search, with $m > 2$. For this reason, a b-tree is also called a b+ tree or b* tree in other texts. Next, let's see how a b-tree index works in Oracle with an example.

A b-tree index structure uses its internal nodes to guide the search down to the exact leaf node that contains the data entries pointed by the ROWID pointer. For example, with the Oracle HR sample schema, we can add an index named EMP_LNAME_IX to allow searching an employee by last name. The syntax of the SQL statement for creating this index named HR.EMP_LNAME_IX is shown below:

```
CREATE INDEX HR.EMP_LNAME_IX
ON HR.EMPLOYEES (LAST_NAME)
TABLESPACE EXAMPLE
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 64K BUFFER_POOL DEFAULT) NOLOGGING
```

A b-tree index is popular for quite a few reasons. First, it's resilient to inserts and deletes in the sense that the tree structure can stay balanced with such dynamic operations. Secondly, it's efficient that it rarely goes beyond more than three to four levels to find an entry in the leaf nodes. And lastly, a b-tree index works well with both read and write types of I/O operations. Many other unusual index types such as bitmap indexes and reverse key indexes are essentially for read-only operations, as we will explain later.

Next, I'll introduce some of the more advanced indexing schemes to help expand your knowledge about this important subject of indexing, which is one of the keys to achieving high performance and scalability with Oracle-based enterprise applications. I'll start with covering indexes in the next section.

18.3 ADVANCED INDEXING SCHEME I: COVERING INDEXES VERSUS INDEX-ORGANIZED TABLES

A covering index is different from a regular index because not only the index key columns but also non-key columns are built into the index. I coined a name for covering index, which is data in index (DII). The name of DII can reflect the fact that the index does not only contain the index key as one would expect normally but also the columns representing the data items that an application query is looking for. One might think that the index key part alone would be sufficient and there was no need to include extra, non-key data columns in the index, but it turned out otherwise. This will be illustrated further in Chapter 24 with a case study based on a real product.

At this point, you might wonder if covering indexes are the same as the index-organized tables. They are similar but not exactly the same. The difference between the two is clear: covering indexes are indexes and IOTs are tables. Perhaps a more pertinent question is what the difference is between an ordinary table and an index-organized table. This question is partially answered with the following SQL statement of creating an IOT:

```
CREATE TABLE my_iot_table (
    my_iot_id NUMBER,
    my_c1 VARCHAR2(10),
    my_c2 VARCHAR2(10),
    my_c3 VARCHAR2(10),
    CONSTRAINT my_pk_iot PRIMARY KEY (my_iot_id, my_c1 )
)
```

ORGANIZATION INDEX;

Note the **PRIMARY KEY** and **ORGANIZATION INDEX** keywords highlighted in boldface. An ordinary table can optionally have a **PRIMARY KEY** and doesn't have the **ORGANIZATION INDEX** keyword. Table 18.1 further illustrates all similarities and differences between an ordinary table and an IOT.

Note that a covering index is still a b-tree index. A b-tree index is a much broader concept in general. In the next section, I'll present another more advanced indexing scheme of a function-based index.

18.4 ADVANCED INDEXING SCHEME II: FUNCTION-BASED INDEXES (FBIs)

A function-based index or an FBI is designed for speeding up search queries that contain expressions of columns or functions in their WHERE-clause conditions. For example, with a query like:

```
SELECT last_name from EMPLOYEES WHERE LOWER (last_name) = 'smith';
```

Table 18.1 Comparison of Index-Organized Tables with Ordinary Tables

Task	Ordinary Table	Index-Organized Table
A row is uniquely identified by PK	rowid optional	PK mandatory
Building secondary indexes with	physical rowid	logical rowid
Access is based on	physical rowid	logical rowid
All rows returned by	sequential scan	full index scan
Can be clustered	yes	no
LONG and LOBs data types	yes	LOBs only
Virtual columns	yes	no

One would create the corresponding FBI using the same LOWER function as follows:

```
CREATE INDEX EMP_LNAME_FBI on EMPLOYEES (LOWER(last_name)) ;
```

A function-based index can be either a b-tree index or a bitmap index. When used properly, a function-based index is as effective as a b-tree index in terms of the magnitude of performance improvement it may help achieve, so never hesitate to use it when applicable.

B-tree indexes and function-based indexes are used widely and therefore considered usual indexes, whereas other types of indexes are used less commonly and therefore need justification before making decisions on using them. The next three sections discuss some of those unusual indexing types such as bitmap indexing, reverse key indexing, and compressed composite indexing. Let's start with introducing bitmap indexing in the next section.

18.5 UNUSUAL INDEXING SCHEME I: BITMAP INDEXES

As we mentioned previously, bitmap indexes are designed for speeding up search queries based on a low-selectivity column valued with only one of the two possible options in the domain out of many such rows. An ideal situation is a search query based on a column such as GENDER with the domain of {MALE, FEMALE}. However, it can go beyond only a two-member domain column, namely, a column with more than two possible distinct values, or a few such columns.

A bitmap index stores the index key values as bitmaps of 0s and 1s. Therefore, it takes much less space than a corresponding b-tree index. However, a bitmap index applies locking at the data block level, making it inappropriate for related DELETE, UPDATE, and INSERT (DUI) operations.

From various texts, you might find many different opinions about b-tree indexes versus bitmap indexes. Some suggest never using bitmap indexes, and some suggest that even with queries based on high-selectivity columns bitmap indexes work better than b-tree indexes. My opinion is that they are all correct based on their assumptions, which might have not been made explicit or clear. My recommendation is that one should use bitmap indexes only if both of the following conditions are met:

- The candidate index column is indeed of low selectivity, ideally with two or only a few unique domain values.
- You know for sure that your application will issue statements that are mostly of SELECT type and the DUI statements are not expected to happen frequently.

However, keep in mind that nothing can be more reliable than your own tests carried out with extra care and precision against your application. If such tests are hard to conduct, take the advice of treating it as an unusual indexing scheme, as suggested with the title of this section.

In the next section, I'll present another such unusual indexing scheme—reverse key indexes.

18.6 UNUSUAL INDEXING SCHEME II: REVERSE KEY INDEXES

Reverse key indexes are designed for speeding up search queries with search keys distinguishable by the last parts of a key value only. For example, with an index created with sequence generated IDs, it's very likely that large group of adjacent key values are stored in the same data block. This may create a situation that the same data block is accessed concurrently by many INSERT statements, thus turning that block into a "hot block." If the key values are stored in the reverse order, then those adjacent key values will be spread into multiple blocks, cooling a "hot block" into multiple cooler data blocks.

That sounds a good idea, but the main drawback of a reverse key is that it does not preserve the highly valuable index range scan property of a b-tree index. Once again, the applicability of reverse key indexes depends on your applications, and the answers would be with the thorough tests you should perform.

In the next section, we'll discuss the last type of unusual indexing scheme—compressed composite indexing.

18.7 UNUSUAL INDEXING SCHEME III: COMPRESSED COMPOSITE INDEXES

We have learned that under ideal situations, one should use b-tree indexes for queries based on high-selectivity columns, and bitmap indexes for queries based on low-selectivity columns. However, there could be mixed situations with composite indexes, which were discussed earlier in Section 4.4, that some columns have high selectivity and some have low selectivity. Using the OE sample Schema as an example, a composite index on the columns of `order_id`, `order_status`, `promotion_id`, and `customer_id` creates a mixed situation that the `order_id`, `customer_id` and `promotion_id` columns are highly selective, whereas the `order_status` column would be common to many orders and thus low cardinal. By compressing a composite index, all duplicate values of the low-selectivity columns are removed, thus saving space.

The issue with compressing composite indexes is that it may not save much space since low-selectivity columns typically have short, fixed-length data types, while slowing down various types of operations due to the extra complexity introduced. Once again, one should ask the question of if it's really worthwhile to trade potential performance deterioration for the space saved. A quick back-of-the-envelope calculation can be performed to determine how much space can be saved. With the above example, `order_id`, `customer_id`, `promotion_id`, and `order_status` take 12, 6, 6, and 2 bytes, respectively. So the maximum efficiency would be $2/(12 + 6 + 6 + 2) = 7.7\%$, which probably is not worth to pursue.

Having covered so many different types of indexing, whether usual or unusual, it's time to look at how various indexes are created. This is the subject of the next section.

18.8 HOW TO CREATE ORACLE INDEXES

The harder part of creating an index is to decide what types of indexes to create and what columns to include. We have provided many rule of thumb guidelines and presented each indexing scheme with pros and cons to help you make decisions. In this section, the focus is to take a look at different syntactical variations in creating various types of indexes when a decision has been made.

Still, one needs to have a thorough understanding of the index types as summarized here again:

- *Unique*. Specifies that the value of the column (or combination of columns) in the table to be indexed must be unique.
- *Bitmap*. Specifies that the index is to be created as a bitmap rather than as a b-tree. This option does not apply when creating a global partitioned index.
- *Unsorted (Default)*. If selected, indicates to Oracle that rows are stored in the database in ascending order and therefore do not have to be sorted when creating the index.
- *Reverse*. If selected, reversed key indexes are created. Creating a reverse key index, compared to a standard index, reverses the bytes of each column indexed (except the ROWID) while keeping the column order.

Using the HR sample Schema and the feature of Show SQL available from the OEMJC, the following four statements illustrate how a unique, bitmap, unsorted, and reverse key index can be created with the same index of HR.EMP_LNAME_IX that indexes on the last_name column only. Note that this example is only for illustrating how the syntax would be different for different types of indexes with the boldface keywords in each statement. It does not indicate which index type is suggested for this particular case.

DEFAULT :

```
CREATE INDEX HR.EMP_LNAME_IX
ON HR.EMPLOYEES (LAST_NAME)
TABLESPACE EXAMPLE PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
2147483645 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
NOLOGGING;
```

UNIQUE :

```
CREATE UNIQUE INDEX HR.EMP_LNAME_IX
ON HR.EMPLOYEES (LAST_NAME)
TABLESPACE EXAMPLE PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
2147483645 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
NOLOGGING;
```

BITMAP:

```
CREATE BITMAP
  INDEX HR.EMP_LNAME_IX
  ON HR.EMPLOYEES (LAST_NAME)
  TABLESPACE EXAMPLE PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE (INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
  2147483645 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
  NOLOGGING;
```

UNSORTED:

```
CREATE INDEX HR.EMP_LNAME_IX
  ON HR.EMPLOYEES (LAST_NAME)
  TABLESPACE EXAMPLE PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE (INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
  2147483645 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
  NOLOGGING NOSORT;
```

REVERSE KEY:

```
CREATE INDEX HR.EMP_LNAME_IX
  ON HR.EMPLOYEES (LAST_NAME)
  TABLESPACE EXAMPLE PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE (INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
  2147483645 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1)
  NOLOGGING REVERSE;
```

Note the location of the keywords in each case. Of course, the above example does not cover all index schemes. Consult other Oracle texts such as those recommended at the end of this chapter for any index types not covered here.

18.9 SUMMARY

In this chapter, we explained how to properly index those tables that contain large volumes of data to speed up the execution of the relevant SQLs. Some effective rules of thumb in indexing tables are described and illustrated with examples. However, Oracle started enhancing many auto-tune features a few years ago, and it's beneficial to know what auto-tune features exist so that one can try to exploit those built-in tuning features rather than devoting too much time doing manual tuning/tweaking unnecessarily. The next chapter discusses all major auto-tune features built into Oracle 11g.

RECOMMENDED READING

The following Oracle document provides the most authoritative information about Oracle SQL:

Oracle Corp, *Oracle Database SQL Language Reference*, 11g Release 1 (11.1) B28286-05 (1446 pages), September 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28286.pdf.

The Part IV, “Optimizing SQL Statements,” of the following Oracle document describes in detail about Oracle SQL optimizations, including such topics as using indexes and clusters, and so on:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 18.1 List and explain the general rules of thumb about indexing.
- 18.2 Explain if the order of columns appearing in an index matters.
- 18.3 Why are some indexing schemes characterized as *usual* while some others as *unusual*? Match each of the indexing schemes (usual or unusual) with a corresponding situation that it’s most applicable.
- 18.4 Explain what a b-tree index is with a graphical representation of a b-tree index. How is a b-tree index defined structurally? Does a b-tree index deal with DML operations effectively? What does it mean by the term of *order* of an Oracle b-tree-index?
- 18.5 Explain why a bitmap index is not necessarily limited to the table columns with two unique values only.
- 18.6 What’s the difference between a covering index and an index-organized table? Which approach is more desirable?
- 18.7 What’s a reverse key index conceptually? Explain what side effect it may have.
- 18.8 What’s the benefit of using a compressed composite index? How do you determine if the potential gain might be marginal or significant?
- 18.9 What is an unsorted index? What are the pros and cons of an unsorted index?

19

Auto_Tune Features

The empires of the future are the empires of the mind.
—WINSTON CHURCHILL, *speech at Harvard University, September 6, 1943*

Starting from Oracle 10g and 11g, Oracle has added many auto-tune features to make it easier to automatically diagnose and tune the performance and scalability of an Oracle Server or an Oracle RAC (Real Application Cluster). If you open up an Oracle 11g EM DBConsole, click the `Performance` tab at the top, then scroll down to the bottom and click `Advisor Central` under `Related Links`. Then you should see something similar to Figure 19.1, which lists all advisors.

The following topics are covered in this chapter:

- Oracle Automatic Database Diagnostics Monitor (ADDM)
- Automatic Undo Management
- Data Recovery Advisor
- Memory Advisors
- MTTR Advisor
- Segment Advisor
- SQL Advisors
- SQL Performance Analyzer (SPA)

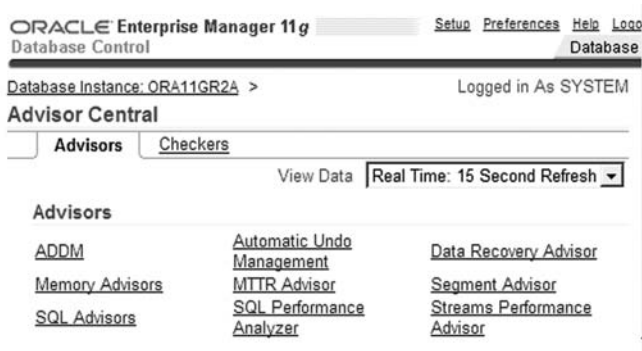


Figure 19.1 Oracle advisors in 11g accessible from the Advisor Central link on the Enterprise Manager Database Control Console.

We will focus more on ADDM, Memory Advisors, SQL Advisors, and SPA, since they are more pertinent to Oracle performance and scalability. Let's start with ADDM next.

19.1 ORACLE AUTOMATIC DATABASE DIAGNOSTIC MONITOR (ADDM)

The ADDM feature was introduced in Oracle 10g and enhanced further in Oracle 11g. It is a self-diagnostic engine that automatically diagnoses and reports the performance issues of a database, thus freeing DBAs from this complex and arduous task. The ADDM first identifies the top, time-consuming activities and then performs root-cause analysis (RCA) based on a sophisticated problem classification tree, which encapsulates the collective knowledge and experience of Oracle's own performance experts in the various areas such as CPU/IO bottlenecks, undersized memory, resource intensive SQL statements, poor connection management, lock contention, and so on. The diagnosed problems and analysis reports can be presented at the instance level of a standalone instance or at the cluster level of an Oracle RAC.

The ADDM works by analyzing AWR (Automatic Workload Repository) reports, which contain statistical information about a database from multiple perspectives. AWR reports are generated with database snapshots taken at regular intervals (once an hour by default). They serve as the basis for all the self-management and self-tuning features of Oracle 11g. In Part Four, we will introduce some real-product-based case studies to illustrate how Oracle performance and scalability issues can be resolved with AWR reports, which provide statistical information about the activities of an Oracle database for a given duration.

ADDM has a key component called Active Session History (ASH), which samples the state of all active sessions every second and stores such state information in memory. The sampled data can be accessed either via a V\$ view or from an AWR report that covers the duration of interest. As shown in Figure 19.2, both an

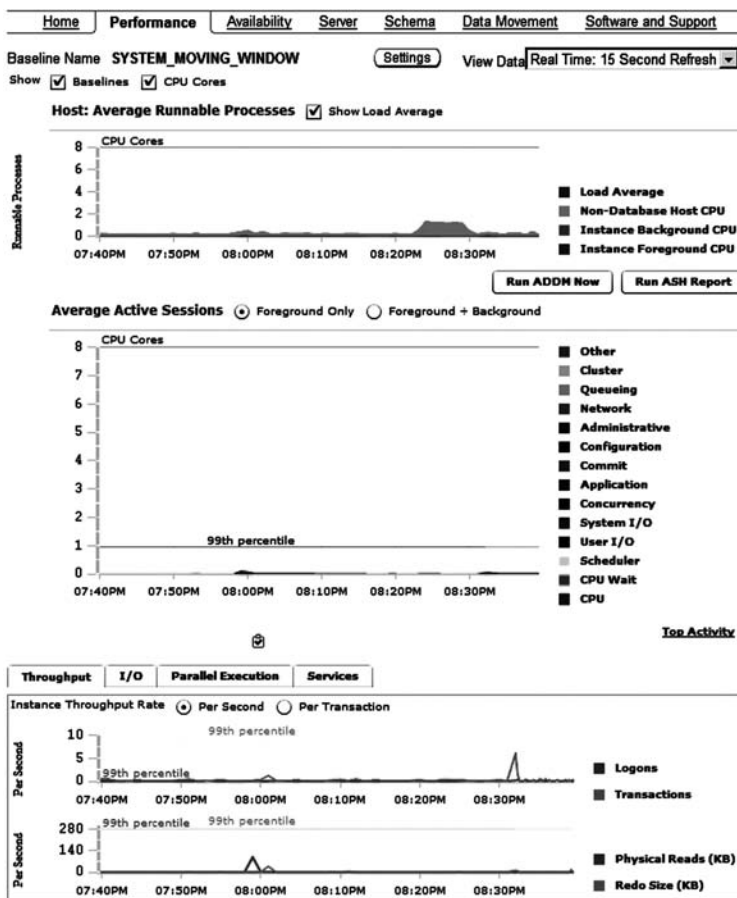


Figure 19.2 The Performance tab on the Oracle Enterprise Manager Database Control Console. The Automatic Database Diagnostic Monitor (ADDM) feature is accessible from the link **Run ADDM Now**. Also note the links such as **Run ASH Report**, **Top Activity**, as well as 14 other performance categories including **CPU**, **I/O**, **Concurrency**, **Application**, **Configuration**, **Network**, **Queuing**, and so on. Other metrics such as **Throughput**, **I/O**, **Parallel Executions**, **Services**, **Logons**, **Transactions**, **Physical Reads (KB)**, and **Redo Size (KB)** are shown at the bottom of the page.

ADDM report and an ASH report can be generated on the fly by clicking the links of **Run ADDM Now** and **Run ASH Report** under the Performance tab on the Enterprise Manager Database Control Console, respectively. This page serves as a rendezvous for displaying almost all Oracle performance aspects, which can be explored further by such categories as **CPU**, **I/O**, **Concurrency**, **Application**, **Configuration**, **Network**, **Queuing**, and so on. Other metrics such as **Throughput**, **I/O**, **Parallel Executions**, **Services**, **Logons**, **Transactions**, **Physical Reads (KB)**, and **Redo Size (KB)** are shown at the bottom of the page as well.

Also note that there is a link labeled `Top Activity` at the bottom right corner in Figure 19.2. This is a very useful feature and I have used it to troubleshoot and resolve many Oracle performance and scalability issues with real products. Keep in mind that whenever you have an Oracle performance and scalability issue, this is the place to check out first.

Next, let's review what Automatic Undo Management does.

19.2 AUTOMATIC UNDO MANAGEMENT

This feature is introduced in the latest version of Oracle 11g to simplify UNDO tablespace management. If you have followed the procedure presented in Part One and created an Oracle 11g database using the Database Configuration Assistant (DCA), you probably have noticed that an auto-extending undo tablespace named UNDOTBS1 was created by default. If your database was created without using the DCA, make sure that you have an UNDO tablespace created for your database, otherwise, Oracle will use the SYSTEM tablespace to store undo records, which is not recommended. You can check the initialization parameter of UNDO_TABLESPACE and make sure that it has a value assigned to it.

Although Oracle provides options for manually managing undo tablespace as well as sizing/extending fixed-sized undo tablespace, it's better to have the undo tablespace auto-extensible and managed automatically by Oracle. If you are interested in manually managing a fixed-sized undo tablespace, refer to Oracle's product documentation for more information.

19.3 DATA RECOVERY ADVISOR

The Data Recovery Advisor is responsible for automatically diagnosing corrupted data on disk, determining the appropriate repair options, and executing repairs at the user's request. It helps reduce the complexity of recovery process, which used to be manual and error-prone. This is more of an administrative task for DBAs than a performance and scalability feature. Refer to Oracle product documentation for more information if you are interested in knowing how to use it.

19.4 MEMORY ADVISORS

Memory management is one of the areas that continue to receive enhancements from release to release. As described in the previous chapters, Oracle 11g automatically manages the total amount of physical memory allocated to it as one chunk, within which both SGA and PGA are managed automatically. Retrospectively, in Oracle 9i, only PGA was made automatically manageable. In Oracle 10g, SGA was made automatically manageable in addition to PGA.

Figure 19.3 shows the sections available on the `Memory Advisors` page. Since the entire page is too large to fit onto one page, it is divided into 3 sections. The first

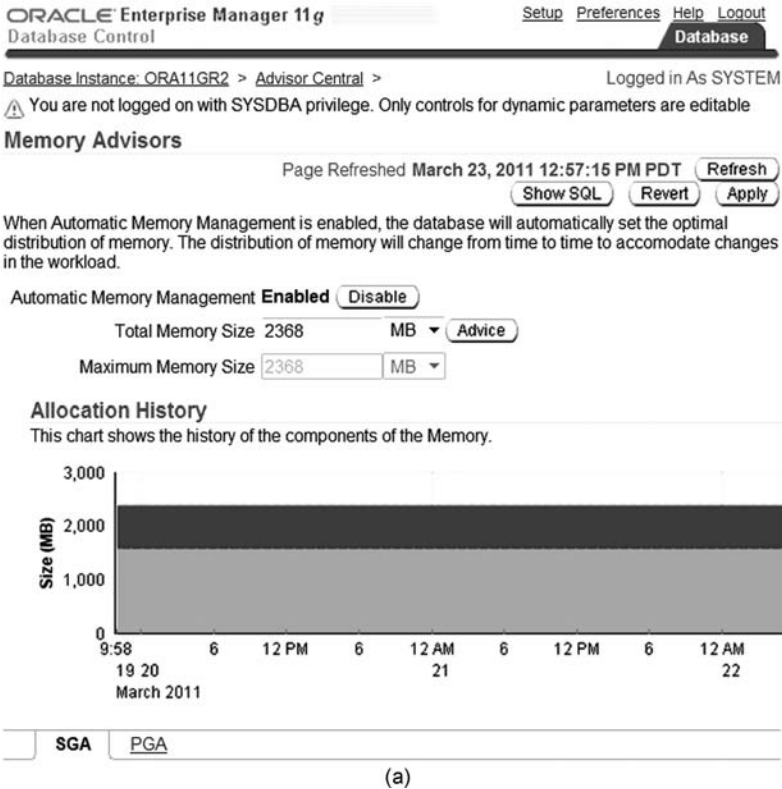


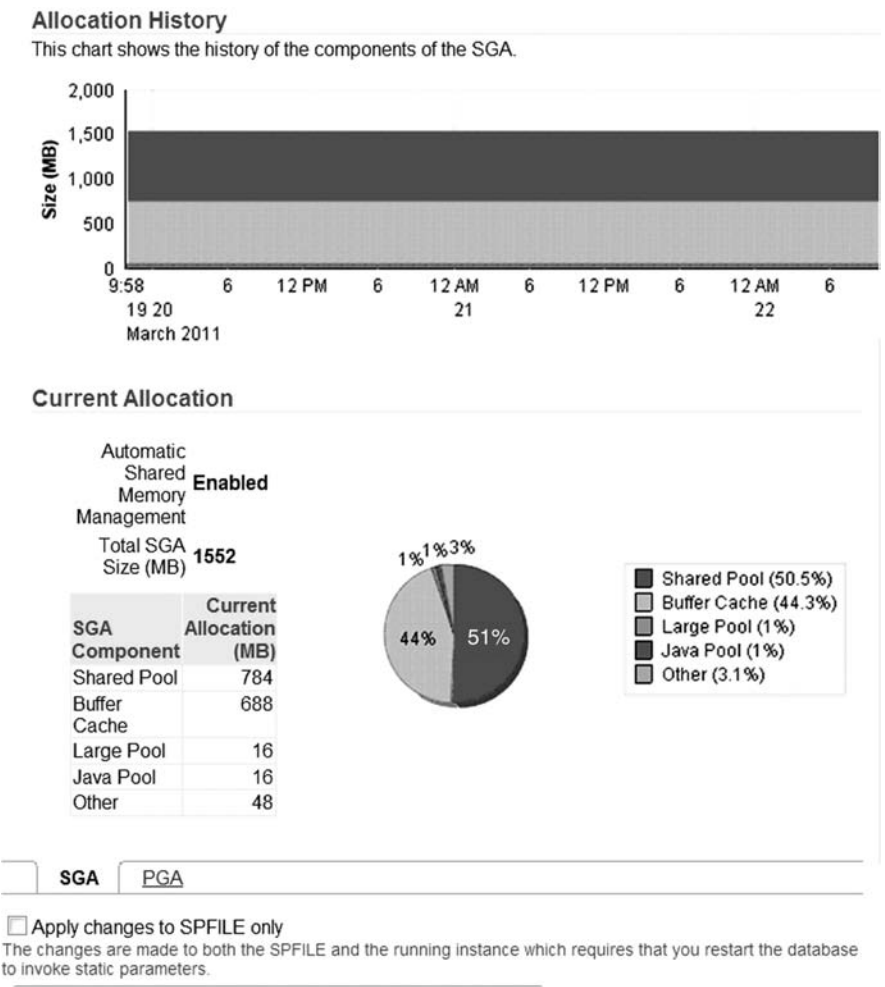
Figure 19.3 Oracle Memory Advisor: (a) showing whether AMM is enabled as well as PGA and SGA evolution, (b) SGA allocation (note various SGA sub-areas illustrated with a form and a pie chart), and (c) PGA allocation.

section is shown in Figure 19.3 (a). From this section, you can quickly verify whether Automatic Memory Management is enabled, with the option to enable or disable it. You can see the Total Memory Size allocated as one chunk as well as the Maximum Memory allocated since the startup of the instance. Also note the Allocation History, which shows the memory distribution between SGA and PGA with time. The purpose of the AMM is to make sure the memory distribution between SGA and PGA is optimally adjusted based on the workload at any given point of time.

Also note the Advice button right next to the Total Memory Size box. There are three implications with this feature:

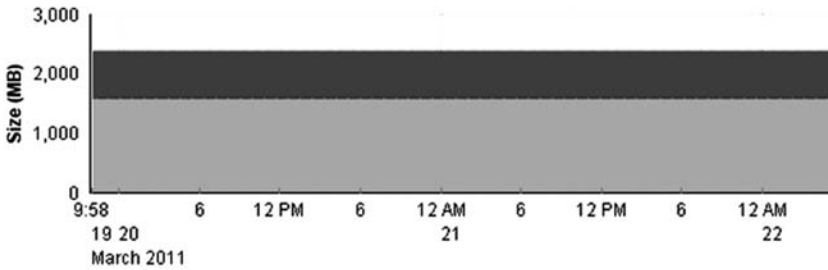
- If AMM is enabled, this feature can give you advice on setting the desired total amount of memory for the entire Oracle database, based on the memory usage statistics collected during the past.
- If Automatic Shared Memory Management is enabled, this feature gives you advice on setting the target sizes of the SGA and PGA.

The System Global Area (SGA) is a group of shared memory structures that contains data and control information for one Oracle database. The SGA is allocated in memory when an Oracle database instance is started.



Allocation History

This chart shows the history of the components of the Memory.



SGA

PGA

The Program Global Area (PGA) is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started.

Aggregate PGA Target (B)

Current PGA Allocated (KB) **112851**

Maximum PGA Allocated (KB) **228246**
(since startup)

Cache Hit Percentage (%) **99.93**

PGA Memory Usage Details

☒ **TIP** The sum of PGA and SGA should be less than the total system memory minus memory required by the operating system and other applications.

SGA

PGA

☐ Apply changes to SPFILE only. Otherwise the changes are made to both the SPFILE and the running instance which requires that you restart the database to invoke static parameters.

☒ **TIP** * indicates controls, if changed, must restart database to invoke.

(c)

Figure 19.3 (Continued)

Share Pool, Buffer Cache, Large Pool, and Java Pool. These concepts have been explained in Chapters 6 and 7, and therefore are not repeated here.

Under the SGA Allocation History chart shows the Current Allocation for the SGA. It indicates whether SGA automatic management is enabled. It also shows the current SGA allocation both in a tabular form and a pie chart. At the bottom, there is a check box to enable “Apply changes” to SPFILE only. By default, the changes are made both in memory and SPFILE, as indicated by the text below that check box.

Figure 19.3 (c) shows information related to PGA such as: (1) Aggregated PGA Target (note that a value of “0” means that PGA is auto-managed), (2) Current PGA Allocated, (3) Maximum PGA Allocated since instance startup, and (4) Cache Hit Percentage. You can also click the PGA Memory Usage Details button and see how the PGA memory is used in terms of work-area sizes. Also note the “Apply

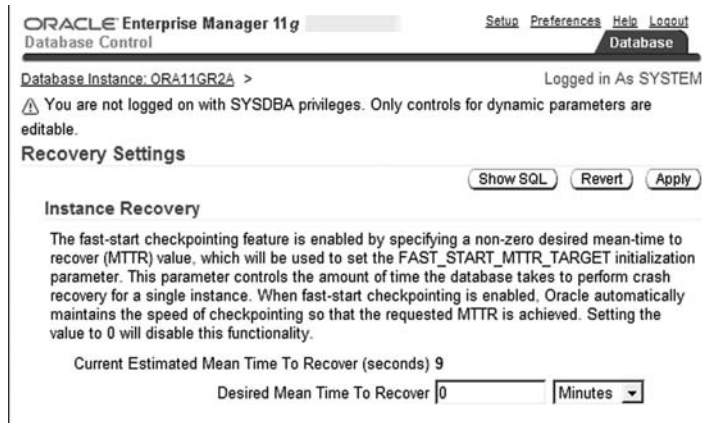


Figure 19.4 Oracle MTTR Advisor showing a current MTTR of 9 seconds.

change . . .” check box and the Tip about restarting database to invoke changes made to static parameters.

19.5 MTTR ADVISOR

The MTTR Advisor enables you to recover an Oracle instance as quickly as possible to reduce the Mean Time To Recovery (MTTR). It sets the target recovery time for recovering from a crash and then writes dirty buffers to disk, with the number of dirty buffers to be written to disk determined according to the algorithms designed to maximize performance while meeting the recovery time objective. For example, Figure 19.4 shows Recovery Settings for Instance Recovery, taken from an Oracle 11g setup for illustration purposes. Note that the sections for Media Recovery and Flash Recovery are omitted here on the MTTR Advisor page. It is seen that Current Estimated MTTR is set to 9 seconds, which is configurable with the text box there labeled Desired Mean Time To Recovery. Setting a smaller target MTTR requires too many dirty buffers to be written to disk and thus affecting performance, whereas setting a larger MTTR increases the time to recover in case a crash occurs. A balance between the two extremes is needed in a real world scenario.

19.6 SEGMENT ADVISOR

The Segment Advisor determines whether objects have unused space that can be released. It also estimates future space, based on historical trends. You can get advice on shrinking segments either for individual schema objects such as tables and indexes or an entire tablespace. This is a less commonly used feature, so consult the relevant Oracle product document if you need more information about it.

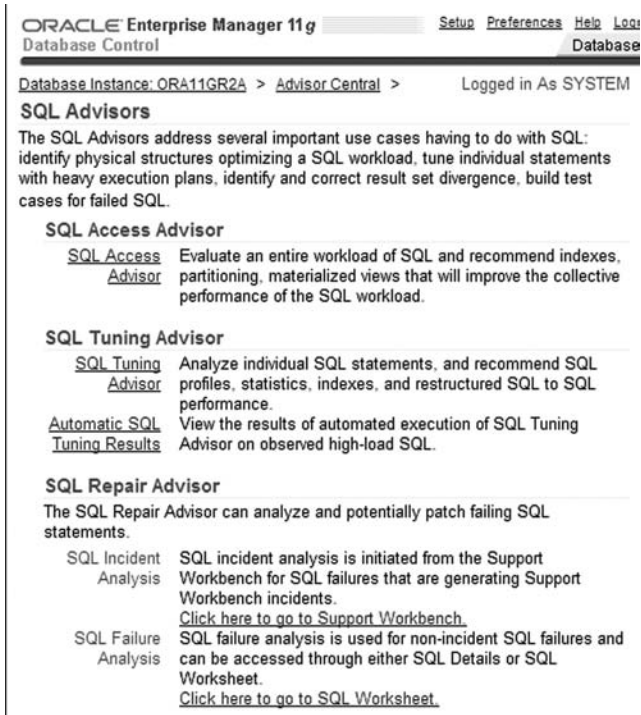


Figure 19.5 Oracle SQL Advisors consisting of SQL Access Advisor, SQL Tuning Advisor, and SQL Repair Advisor.

19.7 SQL ADVISORS

The SQL Advisors consist of three separate SQL advisors. As shown in Figure 19.5, each of those three separate SQL advisors solves a different class of SQL-related issues:

- *SQL Access Advisor.* This SQL advisor works on a workload basis to suggest indexes, partitioning, materialized views, and so on, to help improve the performance of the workload associated with your application. However, keep in mind that creation and maintenance of recommended objects might be time-consuming and may require significant amount of additional space. For instance, partitioning an un-partitioned base table may require careful planning due to its complexity.
- *SQL Tuning Advisor.* This SQL advisor works on individual SQL statements by suggesting indexes, SQL profiles, restructured SQL, and statistics to help improve the performance of the SQL statements.
- *SQL Repair Advisor.* This SQL advisor is used to repair failing SQL statements through two sub-features: SQL Incident Analysis and SQL Failure Analysis.

ORACLE Enterprise Manager 11g
Database Control

Setup Preferences Help Logout
Database

Database Instance: ORA11GR2A > Logged in As SYSTEM

Schedule SQL Tuning Advisor

Cancel Submit

Specify the following parameters to schedule a job to run the SQL Tuning Advisor.

* Name

SQL_TUNING_1299386992006

Description

* SQL Tuning Set

SQL Tuning Set Description

SQL Statements 0

Counts

Overview

The SQL Tuning Advisor analyzes individual SQL statements, and suggests indexes, SQL profiles, restructured SQL, and statistics that improve the performance of the SQL statements.

The SQL Tuning Advisor operates on a collection of SQL. You can choose a SQL Tuning Set to run the advisor. If you do not have a SQL Tuning Set with the desired SQL for running the advisor, you can create a new one.

You can click on one of the following sources, which will lead you to a data source where you can tune SQL statements using the SQL Tuning Advisor.

Top Activity

Historical SQL (AWR)

SQL Tuning Sets

► SQL Statements

Scope

Total Time Limit (minutes)

30

Scope of Analysis

☐ Limited

☒ Comprehensive

This analysis is done without SQL Profile recommendation and takes about 1 second per statement.

This analysis includes SQL Profile recommendation, but may take a long time.

Time Limit per Statement (minutes)

5

Figure 19.6 Oracle SQL Tuning Advisor supported with the concept of a SQL tuning set.

In the remainder of this section, we'll focus on the SQL Tuning Advisor. As shown in Figure 19.6, a SQL tuning advisor is scheduled to run with a given SQL Tuning Set, which is a central concept not only for using the SQL Tuning Advisor but also for using other advisors such as the ADDM and SQL Access Advisor. Let's next clarify what a SQL tuning set is about.

A SQL tuning set is a database object that is composed of the following entities:

- *A Set of SQL Statements.* You can load SQL statements into a SQL tuning set from other features, such as the AWR, the cursor cache, or input from the user.
- *Associated Execution Context.* The context information includes user schema, application module name and action, list of bind variables, and the cursor compilation environment.

- *Associated Execution Statistics.* The execution statistics include elapsed time, CPU time, buffer gets, disk reads, rows processed, cursor fetches, the number of executions, optimizer cost, and the command type.
- *Associated Execution Plans.* This part is optional.

A very important feature with a SQL tuning set is that SQL tuning sets are transportable across databases. This enables the problematic SQLs in a production database to be exported into a SQL tuning set, and then transported into a test database for investigation.

Although one can manage a SQL tuning set with SQL tuning set APIs programmatically, the recommended approach is to use the Oracle EM DBConsole. As you may have noticed in Figure 19.6, the bottom right corner displays the link of Top Activity, which provides easier accesses to tuning SQLs using SQL tuning advisor. We'll not go into the details of how to use SQL tuning advisor to tune SQLs, etc. If you need to use this feature at some point with your development of an Oracle-based application, consult Oracle's documentation for more information.

19.8 SQL PERFORMANCE ANALYZER

As is shown in Figure 19.7, the SQL Performance Analyzer (SPA) is used for evaluating the effects of environment changes on the execution performance of the

ORACLE Enterprise Manager 11g Setup Preferences Help Logout
Database Control Database

Database Instance: ORA11GR2A > Advisor Central > Logged in As SYSTEM

SQL Performance Analyzer

Page Mar 5, 2011 8:34:12 PM Refresh View
Refreshed PST Data Real Time: 15 Second Refresh

SQL Performance Analyzer allows you to test and to analyze the effects of changes on the execution performance of SQL contained in a SQL Tuning Set.

SQL Performance Analyzer Workflows

Create and execute SQL Performance Analyzer Task experiments of different types using the following links.

- [Upgrade from 9i or 10.1](#) Test and analyze the effects of database upgrade from 9i or 10.1 on SQL Tuning Set performance.
- [Upgrade from 10.2 or 11g](#) Test and analyze the effects of database upgrade from 10.2 or 11g on SQL Tuning Set performance.
- [Parameter Change](#) Test and compare an initialization parameter change on SQL Tuning Set performance.
- [Exadata Simulation](#) Simulate the effects of a Exadata Storage Server installation on SQL Tuning Set performance.
- [Guided Workflow](#) Create a SQL Performance Analyzer Task and execute custom experiments using manually created SQL trials.

SQL Performance Analyzer Tasks

Select Name	Last Owner Modified	Current Step Name	Type	Status	SQLs Processed	Steps Completed
No SQL Performance Analyzer Tasks available.						

✓ TIP For an explanation of the icons and symbols used in the following table, see the [Icon Key](#)

Figure 19.7 Oracle SQL Performance Analyzer (SPA).

SQLs contained in a SQL tuning set, for example, due to changes either in version or initialization parameters. On the SQL Performance Analyzer page, there are two subcategories listed: SQL Performance Analyzer Workflows and SQL Performance Analyzer Tasks. The SQL Performance Analyzer Workflows consists of the following three options:

- *Optimizer Upgrade Simulation*. This option allows a user to test potential effects on the performance of the SQLs defined in a SQL Tuning Set due to version changes, for instance, from Oracle 10g to Oracle 11g.
- *Parameter Change*. This option answers the question of how the performance of the SQLs defined in a SQL Tuning Set would change if an initialization parameter were changed to a value different from its base value.
- *Guided Workflow*. This option provides a procedure for guiding the execution of a two-trial SQL performance analyzer test. It contains the following steps:
 - Create a SQL performance analyzer task based on a SQL tuning set
 - Replay the SQL tuning set in the initial environment
 - Replay the SQL tuning set in the changed environment.
 - Compare the above two replays
 - View the trial comparison report.

This is a useful feature for experimenting with the impacts of potential changes without actually committing the changes. However, this is more of an experimental tool than an advisor, as it doesn't run its own tests and then tell a user the optimal setting for a given initialization parameter. Therefore, its usefulness might be limited.

19.9 SUMMARY

This chapter introduced various Oracle tuning advisors available in the latest versions of Oracle 11g as of this writing. We explained conceptually what each advisor is for. From performance and scalability perspectives, the Automatic Database Diagnostic Monitor, the Memory Advisors, and the SQL Tuning Advisor are most relevant.

From this point on, we'll focus on presenting several quality case studies to demonstrate how Oracle meets real world performance and scalability challenges, all based on the real products I worked on. The intention is to help enhance your understanding of some of the major performance and scalability features built-into Oracle from release to release. Also, if you are working on developing an Oracle-based enterprise application, any of the optimization and tuning techniques associated with these case studies may help you achieve immediate, significant, measurable improvements on the performance and scalability of your application if applicable.

RECOMMENDED READING

The following Oracle product document covers auto-tuning features in Oracle 11g in detail:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 19.1** Describe how memory management has evolved from Oracle 9i to 10g and 11g.
- 19.2** How can you invoke the Automatic Database Diagnostic Monitor?
- 19.3** What's the difference between the SQL Tuning Advisor and the SQL Access Advisor?
- 19.4** What's a SQL tuning set? Why is it an important concept for auto-tuning the performance and scalability of an Oracle database?
- 19.5** If you are working on an Oracle-based application, use the Top Activity feature to initiate a SQL tuning experience.

Part Four

Case Studies: Oracle Meeting Real World Performance and Scalability Challenges

*Art is the triumph over chaos.
—John Cheever*

In the previous parts, we first demonstrated how to set up a working Oracle environment and how to get around it. Then we explored various basic concepts with a quick tour of an Oracle Server. We introduced Oracle architectural features driven by performance and scalability. We further explored Oracle storage structure as well as Oracle memory management. We introduced Oracle Wait Interface (OWI), which is a very powerful Oracle performance troubleshooting tool. We emphasized the importance of writing efficient SQLs in achieving overall performance and scalability for Oracle-based enterprise applications. We explained how the Oracle cost-based optimizer works hard behind the scene to ensure that every SQL statement is executed optimally.

Although some quantitative case studies are included in the previous parts, they are typically less sophisticated with very limited scopes. The context of a case study presented previously is closely related to the subject of a specific section in which

it is contained. In this part, we present more quantitative case studies that are differentiated from the previous case studies in the following aspects:

- *Single Variable Based.* I always advocate that software performance and scalability tests should be conducted rigorously like scientific experiments. In this regard, it's imperative to stick to the single variable principle, namely, only one parameter is changed at a time rather than changing multiple parameters simultaneously. The reason is simple: if multiple changes were made for a single test run, then it's difficult to separate the positive and negative effects from each individual parameter. Another reason is that if you are dealing with a customer performance escalation, keep in mind that a customer's production environment is controlled much more tightly than an internal test environment that every change must be thoroughly reviewed and approved before being applied. To me, it's inappropriate to recommend either untested changes or changes that don't matter much at all. The way to sift through multiple factors is to rigorously follow the single variable principle that only one parameter is changed at a time. As you will find, all my case studies presented throughout the remainder of this text satisfies this single variable principle. In all these case studies, a single variable turned out to be a silver bullet for resolving the Oracle performance and scalability issue involved, which also is why they are chosen here.
- *Real Product Based.* Every case study in this part is based on my own real experiences in coping with Oracle performance and scalability issues associated with various real enterprise applications I have worked on during the past decade. In general, I believe the performance and scalability best practices based on real products are more trustworthy and valuable than those obtained with simple samples. Likewise, I would feel more compelling if the case studies presented by others were from real products as well.
- *Original.* Every case study in this part is presented the way it was tackled at the time when I was working on it. None of them were tailored for the sake of presenting a case study or for illustrating certain specific feature in the context of Oracle performance and scalability. These case studies represent the facts rather than opinions about how to optimize and tune Oracle performance and scalability. I value more on the *facts* exhibited with real product based case studies than on wild guesses which are not dependable most of the time in helping resolve real performance issues.
- *Practicality.* Since these case studies are based on real products and original, they are far more practical than academic exercises composed of simple samples, although I share the merits of samples for being flexible and convenient in demonstrating the performance and scalability characteristics of Oracle. In fact, I have used simple samples like the Scott schema as well in proper contexts in the previous parts, but not at all throughout this last part of the book.

With the high standards described above, this part presents the following quantitative case studies on Oracle performance and scalability:

- Chapter 20, “Case Study: Achieving High Throughput with Array Processing,” demonstrates how one can significantly improve the throughput of an Oracle-based enterprise application by adopting the array processing technique that has been a very common practice with Oracle for quite some time.
- Chapter 21, “Case Study: Performance Comparison of Index-Organized Versus Heap-Organized Tables,” illustrates the huge performance advantage of an index-organized table compared with the heap-organized table with the conditions of the same table and same query.
- Chapter 22, “Case Study: SQL Tuning: ‘IN’ Versus ‘OR’ Versus Global Temporary Table,” reveals the subtleties with various SQL constructs that the same query composed of different types of SQL statements can result in different performance and scalability characteristics for the underlying application.
- Chapter 23, “Case Study: Data Access Paths,” disproves the conventional belief of *caching can always improve performance* with a data double buffering case study. This also serves as a good example of how one can have poor performance and scalability with very expensive hardware if a software product is not configured properly.
- Chapter 24, “Case Study: Covering Index,” illustrates how proper indexes can determine the performance and scalability of an Oracle-based enterprise application with everything else undisturbed.
- Chapter 25, “Case Study: Cursor Sharing,” illustrates how a single change of an Oracle initialization parameter can make a huge difference in the performance and scalability of an Oracle-based enterprise application.
- Chapter 26, “Case Study: Bulk Transactions,” illustrates how one can benefit in performance and scalability if multiple transactions are committed in bulk. This is another way to save the overhead of a task that Oracle executes.
- Chapter 27, “Case Study: Missing Statistics,” illustrates the importance of having up-to-date statistics that we have emphasized throughout this text.
- Chapter 28, “Case Study: Misconfigured SAN Storage,” once again demonstrates how one could get poor performance and scalability with a good piece of hardware if the hardware were improperly configured.

Without delaying further, let’s start with the first case study of achieving high throughput with the array processing optimization technique in the next chapter.

20

Case Study: Achieving High Throughput with Array Processing

Art is not a thing; it is a way.
—**Elbert Hubbard**

Array processing helps reduce the number of round trips between an application server and its backend database server by packing multiple requests into one load to send across the network. On the receiving side, the requests are unpacked and processed one by one. This is a well-known technique that can help improve the performance and scalability of a client/server type of application significantly whether a database is involved or not.

With a specific application and the setup, one variable with the array processing technique is the array size, namely, the number of requests to be sent across the network in one load. In this case study, the tests using an array size of ten with two batch jobs of a real product showed that the SQL insert time was improved by as much as eight times and the overall processing time reduced by half. It's highly recommended to adopt array processing with every Oracle-based enterprise application as one of the most effective Oracle performance and scalability best practices.

The structural elements of this case study include the following:

- Context
- Performance Model
- Tests
- Solution
- Effects of Array Processing

Let's first describe the context where an implementation of the array processing optimization technique is most applicable.

20.1 CONTEXT

Large-scale enterprise applications require careful use of various performance and scalability oriented implementation patterns at the component level to achieve the highest possible performance and scalability at the system level. In this regard, it has been well known that the performance and scalability of a software application can be improved significantly by minimizing the number of round-trips between two systems or more. The associated technique is termed *array processing*, which can be mathematically formulated by the following performance law:

$$D_i = V_i \times S_i, \quad (1)$$

where, in the context of queuing theory, V_i is the average number of visits to queue i associated with a queuing node (or component) labeled with the subscript " i ", S_i is the average service time of a request at queue i per visit to queue i , and D_i is the service demand, which measures the total residence time of processing a request at the associated queuing node or component. For a more thorough coverage of queuing theory, refer to my other text (Liu, 2009) or similar texts from other sources. But it is sufficient to state that by reducing the number of round-trips between two components, it would take less time to complete the same amount of work, thus improving the overall system throughput.

The simple formula, Eq. (1), implies abundant opportunities for improving the performance and scalability of a software application. In addition to array processing, which aims at reducing the number of round-trips, one can also improve the performance and scalability by decreasing the service time through shredding unnecessary business logic or using faster hardware. Once again, these topics are beyond the scope of this text, and readers can refer to my other text (Liu, 2009) or similar texts from other sources.

Specific to database-centric enterprise applications, array processing applies to both array fetch and array insert/update. Array processing has been well-supported in Oracle through the JDBC (Java Database Connectivity) technology since Oracle 8 when Java became one of the most popular languages for developing Internet-based applications. However, array processing is a generic, programming language agnostic

performance pattern, which can be implemented in any language and in any application as long as the problem domain is contextually relevant.

20.2 PERFORMANCE MODEL

While working on performance testing and optimizing a large-scale enterprise application, which was based on Oracle 9i and WebLogic 8.0, I first established a model to reflect what parts were tested from the performance and scalability perspective. As shown in Figure 20.1, the following components were involved in the test:

- *A RequestInterfaceEJB.* EJB (Enterprise Java Bean) is a Java-based, server-side, modular component architecture for building distributed enterprise applications. The life cycle of an EJB object is managed by an EJB container. An EJB is a complex concept, but suffice it to say that an EJB is just a software component that fulfills its duties according to its contracts. The RequestInterfaceEJB shown in Figure 20.1 handles the requests from a client.
- *A ReqJMSQueue.* This is a JMS (Java Messaging Service) queue that stores incoming requests. The Java Message Service (JMS) API is a messaging standard that allows Java-based application components to create, send, receive, and read messages. It enables loosely coupled, distributed systems to communicate reliably and asynchronously.
- *An ProvisioningAgentEJB.* This is another EJB that is responsible for sending requests to a central server and forwarding responses from the central server to a response JMS queue, which is described below.
- *A RespJMSQueue.* This is similar to the ReqJMSQueue except that it stores responses instead of requests.

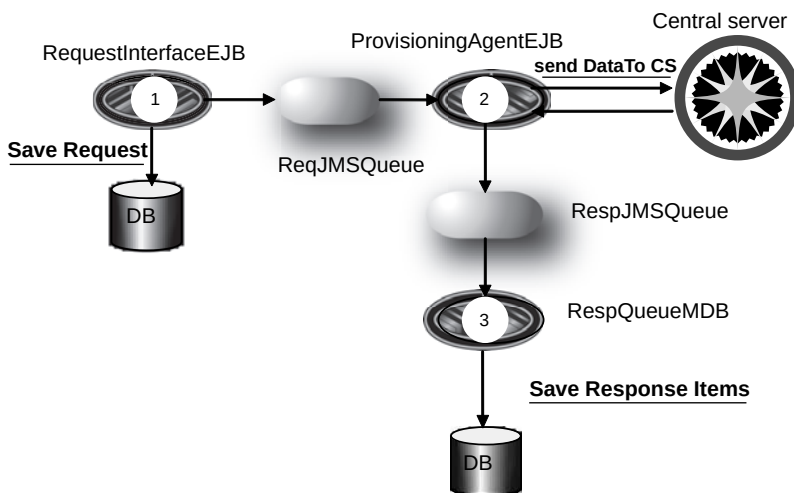


Figure 20.1 Performance model for an Oracle-based enterprise application.

- *A RespQueueMDB*. This is a Java MDB (Message Driven Bean) that acts as a JMS message listener. A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. Conceptually, it is similar to an event listener except that it receives messages instead of events. The messages that an MDB receives may come from any software component, for example, an application client, another enterprise bean, a Web component, a JMS application, or a system that is not based on Java technology.
- *A Central Server*. The central server processes requests it receives from the provisioning agent EJB and then send responses back to the provisioning agent EJB.
- *A Database*. The Oracle database stores both requests and responses.

In Figure 20.1, two APIs of `saveRequest` and `saveResponseItems` are shown as well in addition to the components mentioned above. These two APIs were associated with the `RequestInterfaceEJB` and `RespQueueMDB`, respectively. The next section discusses the tests conducted with the reference model established above. It is recommended that whenever you start a performance and scalability test, establish a model similar to what is shown in Figure 20.1 to get a clear idea about what is under test.

20.3 TESTS

The tests conducted were defined by two batch jobs, which would have to run on a nightly basis. One batch job was designed to process zip codes and the other to delete services. For convenience, we'll refer those two batch jobs as the `zipcode` batch job and `deleteServices` batch job, respectively. Note that what business functions these two batch jobs actually fulfilled in that enterprise application does not matter much, and thus are not elaborated here.

The approach taken to testing those two batch jobs was based on the concept of a *performance map*, which was discussed in detail in my other text (Liu, 2009). Based on the detailed information shown on the performance maps for the two respective batch jobs, it was identified that the two APIs of `saveRequest` and `saveResponseItems` were the bottlenecks. The call count information from the execution of each API also indicated that the relevant insert and update SQLs were executed one by one on Oracle without using array processing.

The next section illustrates how the performance and scalability issues identified with those two APIs were solved by using array processing.

20.4 SOLUTION

Since array processing is an implementation issue rather than a design issue, in general, one can just replace the part of the code that is involved in issuing the SQL statements to Oracle. The following code snippet in Java shows fetching multiple

rows in one round trip between the application server and the database server with array processing:

```
Statement stmt = DefaultConnection.createStatement ();
stmt.setFetchSize (ARRAY_SIZE); // read in from an external file
ResultSet rset = stmt.executeQuery ("SELECT * " +
    " FROM EMPLOYEES ");
rset.next ();
while ( rset.next () ) {
    /* process the rows... */
}
```

Note the statement of “`stmt.setFetchSize (ARRAY_SIZE);`” that sets the array size to a value explicitly specified in an external properties file.

The following code snippet shows how array processing can be implemented for DML SQL statements, in this case, an insert SQL statement. A `Statement` object supports the `addBatch` and `executeBatch` methods, allowing the program to construct and submit a batch of rows for inserting into the database. It’s up to the program to keep track of the number of rows in each batch and to call `executeBatch` when the expected number of rows has been added. The following code snippet shows inserting multiple rows in one batch with array processing:

```
PreparedStatement insertStmt = conn.prepareStatement
    ("INSERT INTO CUSTOMERS VALUES (?, ?, ?, ?)");

// assuming customer info is retrieved into a result set
// from another table
int insertCount = 0;
while ( rset.next () ) {
    insertStmt.setString (1, rset.getString ("CUST_FIRST_NAM"));
    insertStmt.setString (2, rset.getString ("CUST_LAST_NAME"));
    insertStmt.setString (3, rset.getString ("CUST_ADDRESS"));
    insertStmt.setString (4, rset.getString ("CUST_EMAIL"));

    insertStmt.addBatch ();
    insertCount++;

    if (insertCount==ARRAY_SIZE) {
        insertCount = 0;
        insertStmt.executeBatch ();
    }
} // end while

int[] insertCounts = insertStmt.executeBatch ();
```

For detailed information about how to use JDBC with latest versions of Oracle, refer to the sources recommended at the end of this chapter.

The next section shows the effects of array processing on the performance of the `zipcode` and `deleteServices` batch jobs.

20.5 EFFECTS OF ARRAY PROCESSING

Figure 20.2 shows the effects of array processing on the performance of the two APIs of `saveRequest` and `saveResponseItems` associated with the `zipcode` batch job. This batch job processed 490 `zipcode` entries. It's seen that the total elapsed times with the two APIs had been reduced from 46 seconds to 5.8 seconds with the `saveRequest` API and from 42 seconds to 10 seconds with the `saveResponseItems` API, respectively. Note that the performance gains of about eight times and four times were achieved for those two APIs, respectively, with the implementation of array processing. The batch job processing time was reduced from 31 seconds to 15 seconds before and after array processing was implemented.

Figure 20.3 shows the effects of array processing on the performance of the two APIs of `saveRequest` and `saveResponseItems` associated with the `deleteServices` batch job. This batch job processed 591 service entries. It's seen that the total elapsed times with the two APIs had been reduced from 21 seconds to 8 seconds with the `saveRequest` API and from 92 seconds to 13 seconds with the `saveResponseItems` API, respectively. Note that with the implementation of array processing, similarly impressive performance gains of about three times and seven times were achieved for those two APIs, respectively. The batch job processing time was reduced from 44 seconds to 25 seconds before and after array processing was implemented.

The tests associated with the results shown in Figures 20.2 and 20.3 were conducted with an array size of 10. To investigate the effects of varying array size on the performance of this application, some additional tests were run with the `ARRAY_SIZE` parameter varied from 20 to 100. The results are shown in Figure 20.4. It is seen that the throughput for each batch job had been improved sharply by as much as 200% with an array size of 10, compared with no array

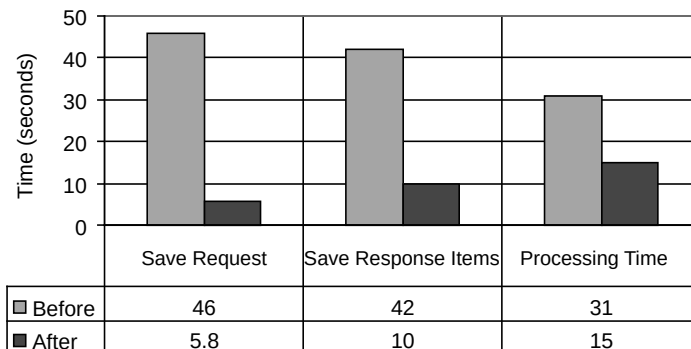


Figure 20.2 Effects of array processing on the `zipcode` batch job of an Oracle-based enterprise application.

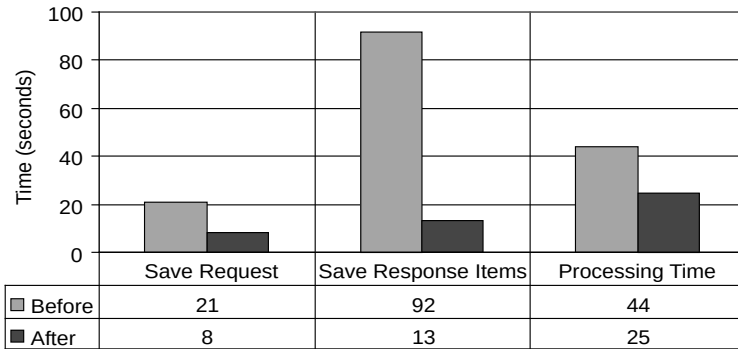


Figure 20.3 Effects of array processing on the deleteServices batch job of an Oracle-based enterprise application.

processing or an array size of one. Large array sizes continued the performance improvement trend, but additional gains relative to the array sizes varied from 1 to 10 had been much less drastic.

As a side note, I have heard people claiming that they use array sizes as large as thousands with their Oracle applications. However, my experiences with many different types of enterprise applications in different contexts appear to prove that additional gains with array sizes larger than 10 are only marginal, as is the case with this application. A side effect of using an extremely large array size is that the locks put on the rows involved might be held too long. Therefore, it's better not to over-use it if the additional performance gains are only marginal.

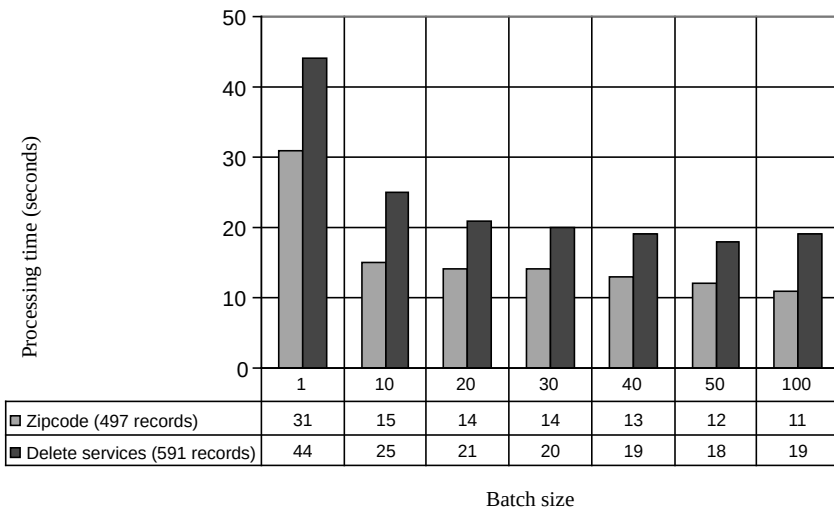


Figure 20.4 Effects of varying array size on the processing times of the zipcode batch job and of the deleteServices batch job of an Oracle-based enterprise application.

20.6 SUMMARY

In this chapter, we demonstrated with a quantitative case study that array processing associated with various types of SQL operations could be very effective in terms of improving the performance and scalability of an Oracle-based enterprise application. We also demonstrated that using an array size larger than ten resulted in marginal performance gains only.

For concrete array processing implementation on Oracle via JDBC, refer to the documents and texts recommended below.

RECOMMENDED READING

Chapter 23, “Performance Extensions,” of the following Oracle document describes in detail about how to implement array processing on Oracle with JDBC:

Oracle Corp, *Oracle Database JDBC Developer's Guide and References*, 11g Release 1 (11.1) B31224-04 (508 pages), July 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/java.111/b31224.pdf

The following text also briefly covers array processing in Oracle:

Guy Harrison, *Oracle Performance Survival Guide: A Systematic Approach to Database Optimization*, Prentice Hall, Boston, 2009.

The following text discusses array processing as a generic performance and scalability pattern with additional case studies:

Henry H. Liu, *Software Performance and Scalability: A Quantitative Approach*, Wiley, Hoboken, 2009.

EXERCISES

- 20.1** Describe the advantage of using a performance model to help guide performance and scalability tests in general.
- 20.2** What is array processing? Is it limited to database-centric enterprise applications only?
- 20.3** If you are working on a real product, how do you verify if array processing is implemented with your application? If not, do you feel that you are obligated to introduce this effective performance and scalability pattern to your team and get it considered and implemented?
- 20.4** If array processing is implemented with your application, what's the optimal array size in your case?
- 20.5** Refer to Figures 20.2 and 20.3. It is seen that the processing time of a batch job could be smaller than its API times. Under what circumstances could something like this happen, namely, a batch job has ended, but its APIs are still running in the background?

21

Case Study: Performance Comparison of Heap-Organized Versus Index-Organized Tables

*Let no one enter here who does not have faith.
—Inscription over the door of Max Plank's Laboratory*

In Oracle, tables are created by default as heap-organized tables (HOT). This kind of data organization is inefficient when hundreds of rows are returned with the data access method of indexed range scan (note: various data access methods were discussed in Chapter 16, including indexed range scan). That's because table data blocks are stored randomly, and it's hard to take advantage of the more efficient data retrieval method of *array fetching*. In order to improve data fetching efficiency, Oracle has implemented index-organized tables (IOT) as was introduced in Chapter 4. In this chapter, using a specific query out of a real product, we compare the performance of the two different table storage structures, index-organized versus heap-organized, for the same table to demonstrate how much performance improvement could be obtained if the table were converted from heap-organized to

index-organized. The product was a product catalog component as part of a large-scale enterprise application targeted to support tens of millions of users,

The IOT has its pros and cons. Caution must be taken when converting a heap-organized table into an index-organized table. This is especially true if the table in question is anticipated to have intensive insert/update operations. A thorough evaluation is required whenever a heap-organized table is being converted into an index-organized table, with all consequences on the DUI (DELETE, UPDATE, and INSERT) operations considered.

This case study is presented with the following sections:

- Context
- Conversion from Heap-Organized to Index-Organized
- Creating Indexes
- Creating Constraints
- EXPLAIN PLANs
- Oracle SQL Traces

In these sections, we cover in detail about how a heap-organized table was converted into an index-organized table and achieved significant performance improvement with a frequently executed query. We first begin with the context of this case study in the next section.

21.1 CONTEXT

We were motivated to convert a heap-organized table to an index-organized table with a query that was frequently executed and slow in performance. The query in question resulted from the context of a product catalog component, which is shown as follows:

```
SELECT ri.prodct_id, p.manual_product_id, ri.is_complete_flg,
p.order_begin_dt, p.order_end_dt, p.product_name, p.is_package_flg,
p.prdtype_id FROM pc.retailer_item ri, pc.product p WHERE
ri.pc_ret_id=252 AND ri.prodct_id=p.prodct_id AND
ri.is_complete_flg=1;
```

The two tables, RETAILER_ITEM and PRODUCT, were created as heap-organized tables during the initial development stage. The performance tests with customer data revealed that this query typically returned hundreds of rows and was slow in performance. We were interested in converting one of the tables, the PRODUCT table, into an index-organized table. This table was fairly static, which means it was a good candidate for being converted into an index-organized table. The next section shows how this table was converted from a heap-organized table into an index-organized table.

21.2 CONVERSION FROM HEAP-ORGANIZED TO INDEX-ORGANIZED

Using the following script named `product_iot_create.sql`, the original heap-organized table was converted into an index-organized table named `product_iot`:

```
CREATE TABLE pc.product_iot (
    prodct_id, product_name, prdtyp_id, is_package_flg,
    created_by, created_dt, modified_by, modified_dt,
    is_duplicate_allowed_flg, manual_product_id, order_begin_dt,
    order_end_dt, effective_begin_dt, effective_end_dt,
    grandfather_period, grandfather_uommes_id, prnrtg_id,
    revenue_accru_begin_dt, revenue_accru_end_dt,
    CONSTRAINT spe_product_iot_pk PRIMARY KEY (prodct_id)
)
ORGANIZATION INDEX
NOCOMPRESS
AS
SELECT prodct_id, product_name, prdtyp_id, is_package_flg,
    created_by, created_dt, modified_by, modified_dt,
    is_duplicate_allowed_flg, manual_product_id, order_begin_dt,
    order_end_dt, effective_begin_dt, effective_end_dt,
    grandfather_period, grandfather_uommes_id, prnrtg_id,
    revenue_accru_begin_dt, revenue_accru_end_dt FROM pc.product
ORDER BY prodct_id, product_name, prdtyp_id, is_package_flg,
    created_by, created_dt, modified_by, modified_dt,
    is_duplicate_allowed_flg, manual_product_id, order_begin_dt,
    order_end_dt, effective_begin_dt, effective_end_dt,
    grandfather_period, grandfather_uommes_id, prnrtg_id,
    revenue_accru_begin_dt, revenue_accru_end_dt;
```

Then, the indexes were created on the converted index-organized table, as is shown in the next section.

21.3 CREATING INDEXES

The following script named `product_iot_create_index.sql` was used to create the corresponding indexes on the index-organized table:

```
CREATE INDEX pc.prodct_iot_sk01_idx on pc.product_iot
    (upper(product_name)) tablespace pc_data pctfree 20 initrans 60
    maxtrans 100 storage ( initial 128k next 0k minextents 1 maxextents
    2147483645 pctincrease 0 freelists 1 freelist groups 1) logging;

create unique index pc.prodct_iot_uk on pc.product_iot
    (manual_product_id) tablespace pc_data pctfree 10 initrans 60 maxtrans
```

```

255 storage ( initial 64k next 0k minextents 1 maxextents 2147483645
pctincrease 0 freelists 1 freelist groups 1) logging;

CREATE INDEX pc.productpackflg_iot_idx on pc.product_iot
(is_package_flg) tablespace pc_data pctfree 10 initrans 60 maxtrans 255
storage ( initial 64k next 0k minextents 1 maxextents 2147483645
pctincrease 0 freelists 1 freelist groups 1) logging;

CREATE INDEX pc.ptprdtyp_iot_idx on pc.product_iot (prdtyp_id)
tablespace pc_data pctfree 10 initrans 60 maxtrans 255 storage (
initial 64k next 0k minextents 1 maxextents 2147483645 pctincrease 0
freelists 1 freelist groups 1) logging;

```

Next, constraints were created on the index-organized table.

21.4 CREATING CONSTRAINTS

The following script named `product_iot_add_constraints.sql` was used to create the corresponding constraints on the index-organized table:

```

ALTER TABLE pc.product_iot add constraint duplicateflag_iot_chk
check(is_duplicate_allowed_flg in (0, 1));

ALTER TABLE pc.product_iot add constraint packageflag_chk_iot
check(is_package_flg in (0, 1));

ALTER TABLE pc.product_iot add constraint prodct_prnrtg_iot_fk
foreign key(prnrtg_id)
references pc.parental_rating(prnrtg_id);

ALTER TABLE pc.product_iot add constraint prodct_iot_uk
unique(manual_product_id);

ALTER TABLE pc.product_iot add constraint prodct_uommes_iot_fk
foreign key(grandfather_uommes_id) references
pc.unit_of_measure(uommes_id);

```

At this point, the `PRODUCT` table was analyzed to have the table statistics updated for the index-organized table. Next, the `EXPLAIN PLANS` and `SQL` traces were obtained to compare the performance of the original heap-organized table to that of the index-organized table.

21.5 EXPLAIN PLANS

Figure 21.1 shows the three `EXPLAIN PLANS` (from top to bottom) for the same query executed under the following three different conditions, respectively:

IOT EXPLAIN PLAN	Cost	Rows	Bytes	Pos	Sort	IO
1. SELECT STATEMENT on	2	303	18180	0		2
2. NESTED LOOPS on	2	303	18180	1		2
3. INDEX RANGE SCAN on SPE_RI_TEST1	2	303	3636	1		2
3. INDEX UNIQUE SCAN on SPE_PRODUCT_IOT_PK		1	48	2		
HOT EXPLAIN PLAN	Cost	Rows	Bytes	Pos	Sort	IO
1. SELECT STATEMENT on	40	303	17271	0		40
2. HASH JOIN on	40	303	17271	1		40
3. INDEX RANGE SCAN on SPE_RI_TEST1	2	303	3636	1		2
3. TABLE ACCESS FULL on PRODUCT	37	17966	808470	2		37
HOT with Index hint on Product_ID_PK	Cost	Rows	Bytes	Pos	Sort	IO
1. SELECT STATEMENT on	305	303	17271	0		305
2. NESTED LOOPS on	305	303	17271	1		305
3. INDEX RANGE SCAN on SPE_RI_TEST1	2	303	3636	1		2
3. TABLE ACCESS BY INDEX ROWID on PRODUCT	1	1	45	2		1
4. INDEX UNIQUE SCAN on PRODCT_PK		1		1		

Figure 21.1 EXPLAIN PLANs of the same query under three different conditions: top—index-organized table, middle—heap-organized table without an index hint, and bottom—heap-organized table with an index hint.

1. The converted index-organized PRODUCT table
2. The original heap-organized PRODUCT table without the index hint on PRODUCT_ID_PK
3. The original heap-organized PRODUCT table with an index hint on PRODUCT_ID_PK

It is seen that the costs were 2, 40, and 305 with those three different queries, respectively. The first query was based on the index-organized table, whereas the second and third queries were with the original heap-organized table without and with a hint passed to the CBO, respectively. It's clear that the index-organized table had reduced the cost of the query significantly. The differences in total execution times are presented next with the SQL traces captured for each test case with the corresponding SQL query as mentioned above.

21.6 ORACLE SQL TRACES

For these tests, a Java test program with JDBC was used to issue the queries alternately against the product catalog database, with the same table built twice, respectively, once heap-organized and once index-organized. Oracle tracing was turned on and off inside the Java program before and after the queries were executed. The product catalog database was bounced prior to each test to clear up the Oracle buffer cache.

Figure 21.2 shows the tkprof report from the SQL trace obtained with the HOT-based PRODUCT table query (note: tkprof is a script that can be used to turn raw SQL trace data into a report). To be clear, this was the HOT-based query that had no index hint applied. It is seen that this query was executed as a HASH JOIN, which took about 385 milliseconds. It is also seen that 375 disk blocks were fetched for executing this query.

Figure 21.3 shows the tkprof report from the SQL trace obtained with the IOT-based PRODUCT table query. It is seen that this query was executed as a NESTED

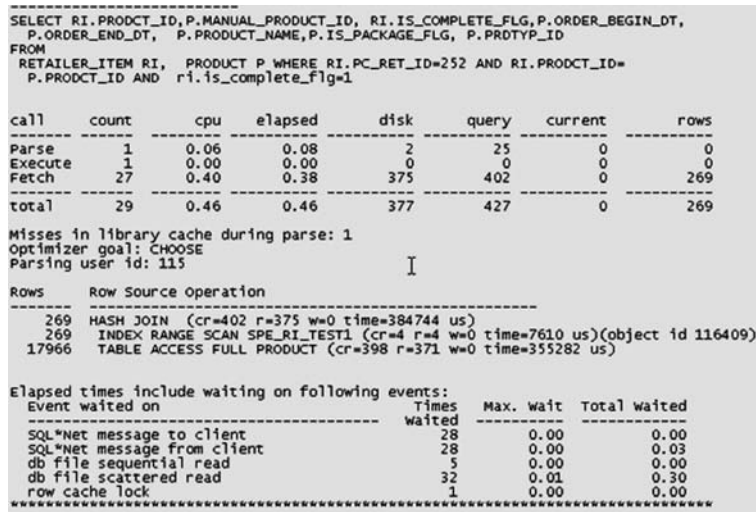


Figure 21.2 The tkprof report from the SQL trace obtained with the HOT-based PRODUCT table query with no index hint applied.

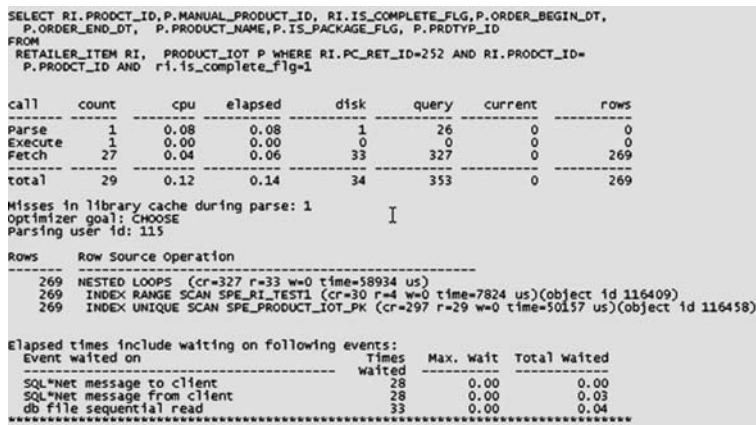


Figure 21.3 The tkprof report from the SQL trace obtained with the IOT-based PRODUCT table query.

LOOPS JOIN, which took 59 milliseconds. It is also seen that 33 disk blocks were fetched for executing this query.

21.7 SUMMARY

From the test results shown in the previous section, it is clear that an index-organized table can help reduce the number of disk blocks significantly, compared with a heap-organized table with the same table data. That orders-of-magnitude reduction in disk I/O blocks fetched resulted from the fact that data was organized according to the

Primary Key with an IOT rather than stored randomly as with a HOT. The overall response time was improved by as much as 6.5 times, which was quite impressive.

There is a parameter named `OVERFLOW` that is relevant to index-organized tables but not used in this case study. The `OVERFLOW` option allows non-key columns to be stored in a separate data segment. If an index-organized table is built once and read frequently, overflow is less an issue than with frequently modified index-organized tables. This condition was satisfied in this case study, as a product catalog is fairly static in general. Refer to the references listed below for more discussions on this issue.

Index-organized tables may also cause maintenance problems. A local primary key index on a heap table partition can be rebuilt `ONLINE` whereas an IOT partition cannot be. Perhaps a product catalog is a special situation where many tables are neither frequently modified nor partitioned. However, any performance recommendation should be implemented only when all pros and cons associated with it have been fully explored and understood. Operational maintenance is an important area that needs to be considered whenever a performance improvement method is put on the table for resolving a performance issue.

RECOMMENDED READING

Chapter 18, “Managing Tables,” of the following Oracle document has a section about managing index-organized tables that provides more information about this subject:

Oracle Corp, *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.

The following text also covers index-organized tables in detail:

Tom Kyte, *Expert Oracle Database Architecture, 2nd Edition*, A Press, New York, 2010.

EXERCISES

- 21.1 Describe the pros and cons of an index-organized table versus a heap-organized table in general.
- 21.2 Refer to the `EXPLAIN PLANS` presented in this section. Identify the following objects that were introduced conceptually in the previous parts of this text:
 - Types of indexes
 - Data access methods
 - Types of wait events
 - Stages associated with a SQL execution
- 21.3 How do you reconcile the various time elements displayed with a `tkprof` report?
- 21.4 What heap-organized tables are better candidates to be converted into index-organized tables?

22

Case Study: SQL Tuning: “IN” Versus “OR” Versus Global Temporary Table

Nicht Kunst und Wissenschaft allein, Geduld will be idem Werke sein. (Not art and science only, but patience will be required for the work.)

—Johann Wolfgang von Goethe

Using the same real product of a product catalog as used in the preceding two case study chapters, this chapter discusses various options in coding SQLs to achieve the best possible performance and scalability. Such SQL optimization activities are common with all large-scale enterprise applications. The purpose of this chapter is beyond just showing the performance numbers from each option. It serves as an example of how one should follow a rigorous test approach to determining the optimum option out of multiple choices available. Hopefully, you will follow the similar approach to optimizing the SQL queries associated with your enterprise applications.

This case study is about whether one should use `IN`, `OR`, or a temporary table in the `WHERE` clause of a SQL query from the performance perspective. Based on a specific query that returned the product availability information from the same product catalog application, it was found that the `IN` approach offered the best

performance relative to other options, although it has the limitation of at most 1000 literal values that can be placed in the `IN`-clause. The `temporary table` solution proved to be very efficient as well. It's surprising that with this specific project I worked on, the temporary table approach performed much better with no index hint than with an index hint. However, with online transaction processing (OLTP) applications, a temporary table has to be populated from the application's side. This extra overhead of populating the temporary table might compromise the effectiveness of the temporary table-based approach.

We also observed that with less than 100 literal values, the `IN` approach with `JDBC preparedStatement` performed better than the same approach with `JDBC createStatement`. For a large number of literal values, however, the time saved from pre-parsing on the database side would be offset by the time required to set the bind variables to concrete values on the application side; therefore, the `IN` approach with `JDBC preparedStatement` actually performed much worse than the same approach with `JDBC createStatement` for larger number of literal values.

This case study is presented with the following sections:

- Context
- Test Program
- Observation 1: `IN_CreateStatement` is the Best Performer
- Observation 2: Batch Insert Saves Time
- `Temptable` Performed Better without an Index Hint than with an Index Hint
- Effects of `APPEND` hint for Populating Template
- Effects of Number of Iterations
- `OR` and `IN` without an Index Hint
- Limitation on the Number of Literal Values and the Size of `OR` Statement
- Dealing with More Than 1000 Literal Values for an `IN` Based SQL Query
- Recommendations for Dealing with 1000 Literal Value Limit in an `IN` Statement

Let's begin with the qualitative context description in the next section and progress to quantitative test results that illustrate the query performance with each option in coding the same query returning the same result set.

22.1 CONTEXT

The problem context is to query seven availability-related columns from the same `availability_value` table, given an arbitrarily large number of `avlval_id` values. Then, how should one compose the query in order to get the best possible performance? There are three approaches as described as follows (note: to save space, the `IN` and `OR` lists have been abbreviated):

1. Using an IN-Clause

```
SELECT /*+ index (av, avlval_pk) */ av.avlval_id, av.avltyp_id,
av.availability_value_name, av.availability_upper_value,
av.availability_lower_value, av.availability_value_desc100
FROM pc.availability_value av WHERE av.avlval_id IN (value1,
value2,..., valuen,...);
```

2. Using an OR-Clause

```
SELECT /*+ index (av, avlval_pk) */ av.avlval_id, av.avltyp_id,
av.availability_value_name, av.availability_upper_value,
av.availability_lower_value, av.availability_value_desc100
FROM pc.availability_value av WHERE (av.avlval_id = value1 OR
av.avlval_id = value2, ..., OR av.avlval_id = valuen, ...);
```

3. Using a Global Temporary Table

```
SELECT /*+ index (av, avlval_pk) */ av.avlval_id, av.avltyp_id,
av.availability_value_name, av.availability_upper_value,
av.availability_lower_value, av.availability_value_desc100
FROM pc.availability_value av, temptable t WHERE av.avlval_id =
t.value;
```

The `availability_value` table had an index on the primary key of `avlval_id`. The Oracle global temporary table was created as follows:

```
CREATE GLOBAL TEMPORARY TABLE pc.temptatble
(value number not null)
ON COMMIT DELETE ROWS;
GRANT all on pc.temptatble to app;
CREATE SYNONYM app.temptable for pc.temptatble;
```

In order to quantitatively determine which approach resulted in the best performance, a test program was constructed to obtain quantitative performance data with each approach independently. The next section details how the test program was constructed.

22.2 TEST PROGRAM

The test program was written in Java to be consistent with how the query was executed in the real product. That Java program created its own JDBC connections to connect to the product catalog database where the table `availability_value` resided. Additional implementation details of that program include:

- Either JDBC Statement or PreparedStatement object was used to execute the query for each approach. It's the end-to-end execution time of those objects that we were concerned with. With the temporary table approach, the table population time using JDBC batch insert was reported as well, as that was how the temptable would have to be populated eventually. The IN and OR based approaches did not incur that overhead associated with the temptable approach.
- The primary performance factor for this study was the number of literal values of avlval_id column that each query used in its WHERE clause. In order to study this factor more systematically, the test program was made to be executable with a varying number of avlval_id values.
- Considerable efforts had been made to make sure that the results were repeatable and realistic. Some of the efforts include:
 - Multiple iterations were executed for each test with the first iteration excluded from averaging.
 - By using a random start number, no two runs or iterations would use the same set of avlval_id values to eliminate the possibility of fetching the same result set from the same set of avlval_id values from caching rather than disk.
 - Each query was tested both with indexing and without indexing.
- The test program measured the elapsed time from the side of the Java program, which included the elapsed time on the database side, the network latency between the application server and the database server, and the execution time incurred on the application server itself.

Besides, the test environment was an isolated environment set up for performance test only without being affected by other non-controllable factors.

The next section summarizes the quantitative test results for each approach.

22.3 OBSERVATION 1: IN_CREATESTATEMENT IS THE BEST PERFORMER

Figure 22.1 shows the test results for all query approaches. Both IN and OR approaches were tested with two different JDBC objects, one with the JDBC PreparedStatement (PS) and the other with the JDBC createStatement (CS). The data points associated with each curve in the figure show the total query execution times with query construction times added to query execution times. For the temptable (TT) approach, the total execution times included the overhead of batch insert time spent populating the temporary table as well. Each set of test results was obtained with 21 iterations while having the first iteration always excluded from averaging (note: the first iteration was considered the priming run). As stated early, each iteration used a different set of avlval_id literal values in the WHERE clause in order to avoid caching issue.

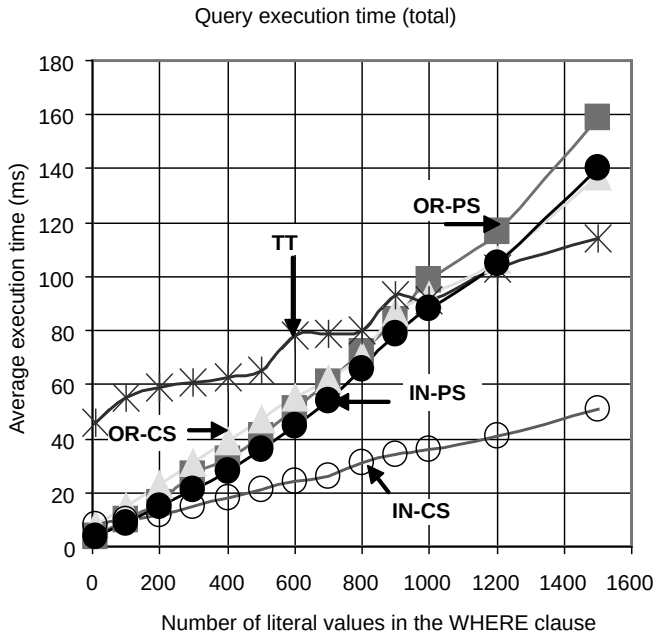


Figure 22.1 Total execution time measured within the Java test program for each query approach (note: index hint was applied in every case).

Note that the `IN` approach had a limitation of allowing 999 literal values at most in the `WHERE` clause. For those runs with more than 999 literal values, the additional literal values were concatenated with another bracket using `OR`. The other two query approaches did not have this limitation.

Table 22.1 shows the data used for generating Figure 22.1. The purpose of including this table is to show the test data in a finer granularity. It is seen that:

- Overall, the `IN` approach resulted in the best performance.
- For smaller number of literal values, `IN` and `OR` approaches performed comparably. However, for larger numbers of literal values, the `IN` approach performed significantly better than the `OR` approach, up to 2.64 times better with 1500 `av1val_id` literal values using `JDBC createStatement (CS)`.
- For a smaller number of 10 literal values, `JDBC PS` (`prepareStatement`) performed twice better than `JDBC CS` (`createStatement`). This performance gain was from pre-parsing with `JDBC prepareStatement`. However, with larger number of literal values, the benefit from pre-parsing with `JDBC prepareStatement` was offset by the overhead of setting bind variables to concrete values on the application side.
- The temptable approach performed significantly worse than the `IN_CS` approach regardless of the number of literal values.

Table 22.1 Execution Times in Milliseconds with Queries Based on TempTable (TT), OR-statement, and IN-statement*

# of literal values	Query time					Overhead				TT insert
	TT	OR		IN		OR		IN		
		PS	CS	PS	CS	PS	CS	PS	CS	
10	44	4	8	4	8	0	0	0	0	2
100	49	9	15	9	10	1	0	0	0	6
200	49	14	23	14	12	2	0	1	0	10
300	47	21	31	19	15	6	0	2	0	14
400	44	27	39	26	18	5	0	2	0	19
500	42	35	46	33	21	6	1	3	0	23
600	52	44	54	41	24	7	1	4	0	26
700	47	53	61	50	26	8	1	4	0	32
800	45	62	70	60	30	10	1	6	1	35
900	54	73	80	73	33	11	7	6	1	39
1000	47	84	91	81	35	15	2	7	1	44
1200	49	102	104	97	40	15	2	8	1	54
1500	51	136	132	130	50	23	5	10	1	63

*Note that query execution time and overhead prior to query execution are separated.

22.4 OBSERVATION 2: BATCH INSERT SAVES TIME

Figure 22.2 shows the temptable population time with batch insert and non-batch insert. With batch insert, the batch size was set equal to the number of literal values. It’s clear that batch insert was a lot faster than non-batch insert, up to 10 times faster with 1500 avlval_id literal values.

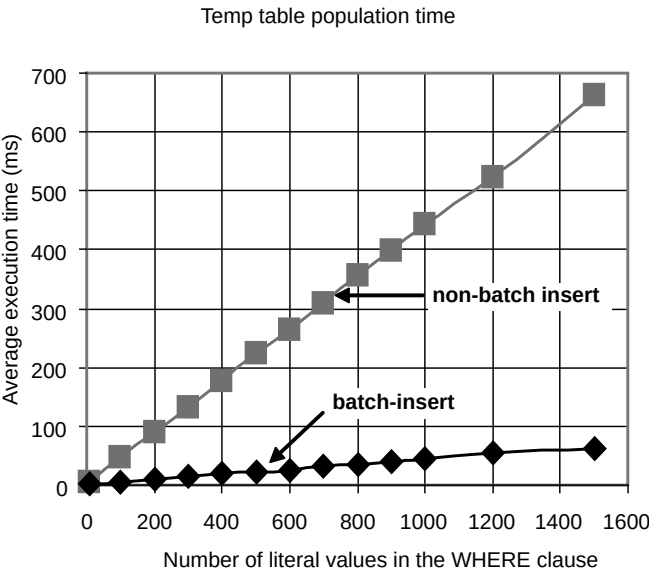


Figure 22.2 TempTable population time measured within the Java test program with batch insert and non-batch insert. JDBC PreparedStatement object was used to issue the insert statement.

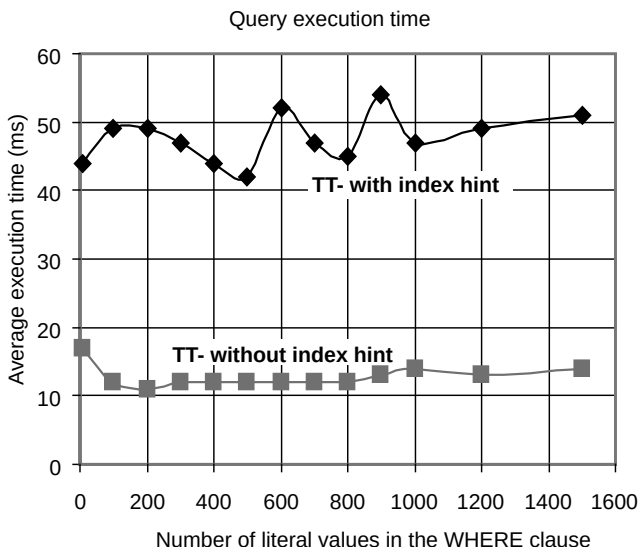


Figure 22.3 Query execution time measured within the Java test program with the TempTable approach with and without an index hint, respectively.

22.5 TEMPTABLE PERFORMED BETTER WITHOUT AN INDEX HINT THAN WITH AN INDEX HINT

It's interesting to see from Figure 22.3 that the temptable approach performed better without an index hint than with an index hint in the query. To understand the performance disparity shown in Figure 22.3, the `EXECUTION PLANS` for the queries with and without an index hint were obtained with 1000 literal values in both cases, as shown in Figures 22.4 and 22.5, respectively. It is seen that the total cost associated with the query was 2829 with the index hint versus 68 without the index hint. Further examination indicates that there was a huge difference in the cost of accessing the `availability_value` table, from 37 without the index hint to 2798 with the index hint. Without the index hint, the `availability_value` table was accessed with one full table scan, while with the index hint, first an `INDEX FULL SCAN` on the `AVLVAL_PK` index was performed, followed by a `TABLE ACCESS BY INDEX ROWID` operation. In the latter case, the CBO was unable to choose `INDEX RANGE`

Operation	Plan/Obj Name	Rows	Bytes	Cost
SELECT STATEMENT	----- 332464442 -----			68
HASH JOIN		8K	502K	68
TABLE ACCESS FULL	TEMPTATBLE1	8K	103K	11
TABLE ACCESS FULL	AVAILABILITY_VALUE	26K	1M	37

Figure 22.4 `EXECUTION PLAN` for the TempTable approach without index hint applied to the query.

Operation	Plan/Operation Name	Rows	Bytes	Cost
SELECT STATEMENT	----- 2991946796 -----			2829
HASH JOIN		8K	502K	2829
TABLE ACCESS FULL	TEMPTABLE1	8K	103K	11
TABLE ACCESS BY INDEX ROWID	AVAILABILITY_VALUE	26K	1M	2798
INDEX FULL SCAN	AVLVAL_PK	1		69

Figure 22.5 EXECUTION PLAN for the TempTable approach with index hint applied to the query.

SCAN, because literal values were not available from the temptable query. This explains why the temptable approach was faster without the index hint than with the index hint.

In the following sections, a few additional issues were explored to ensure that the various approaches discussed above were fully understood and the test results were solid.

22.6 EFFECTS OF APPEND HINT FOR POPULATING TEMPTABLE

The insert times of the temptable shown in Figure 22.2 were obtained with no APPEND hint applied to the insert SQL statement. In order to evaluate the benefits of APPEND hint, additional tests were conducted. By using an APPEND hint like “INSERT/* APPEND */INTO ...;,” Oracle bypasses free-lists and uses new data blocks by raising the high-water-mark to insert data into the table. The test results showed that the differences in batch insert times between with and without APPEND hint were less than 2 milliseconds, which were negligibly small. This probably was because the test environment was a very isolated environment and therefore free-list unlinking was not an issue.

22.7 EFFECTS OF NUMBER OF ITERATIONS

All the test results presented in the previous sections were obtained with 21 iterations, with the first iteration always excluded from averaging. In order to evaluate the effects of the number of iterations on the averaged execution times, we conducted tests with 101 iterations. The test results averaged over 100 iterations are listed in Table 22.2 with those over 20 iterations listed side by side as well. As is seen, no noticeable differences were observed. That proved that the data obtained with 21 iterations was trustworthy.

22.8 OR AND IN WITHOUT THE INDEX HINT

In order to evaluate the effects of index hint on OR and IN based queries, corresponding tests were conducted with index hint for those two OR and IN based queries removed. The results are listed in Table 22.3. As is seen, the query execution times

Table 22.2 Comparison of Query Execution Time in Milliseconds between 20 and 100 Iterations

# of values	TempTable		OR-query				IN-query	
	tt_20	tt_100	ps_20	ps_100	cs_20	cs_100	cs_20	cs_100
10	17	19	4	3	8	8	7	8
100	12	17	9	9	15	15	9	9
200	11	13	15	15	22	23	12	12
300	12	12	21	21	30	29	15	15
400	12	12	28	28	38	37	18	18
500	12	12	35	35	47	46	23	21
600	12	12	43	47	54	53	24	23
700	12	12	52	52	62	62	27	26
800	12	13	62	62	68	70	29	29
900	13	13	73	74	78	77	32	32
1000	14	13	85	85	87	84	35	35
1200	13	13	102	104	116	104	—	—
1500	14	13	135	136	129	127	—	—

jumped abruptly when the number of literal values exceeded 300 in the case of without an index hint. Apparently, the CBO was unable to arrive at an optimum EXECUTION PLAN when the index hint was removed. Keep in mind that the CBO is only as efficient as the intelligence built into it, and there is no guarantee that it would always pick the best execution path. This is an example showing that sometimes the CBO may need external hints to arrive at an optimum execution plan for some queries.

Table 22.3 Effects of Index Hint on OR and IN Queries*

# of values	OR-query				IN-query	
	ps w/hint	ps w/o hint	cs w/hint)	cs w/o hint	ps w/hint	ps w/o hint
10	4	3	8	8	7	8
100	9	9	15	19	9	14
200	15	15	22	32	12	22
300	21	21	30	46	15	32
400	28	617	38	594	18	487
500	35	817	47	733	23	628
600	43	1103	54	1045	24	711
700	52	1123	62	1353	27	1084
800	62	1351	68	1545	29	1577
900	73	1414	78	1720	32	1119
1000	85	1426	87	1433	35	1687
1200	102	2227	116	2551	35	1547
1500	135	2573	129	2807	35	1688

*Numbers are in milliseconds. The abbreviations of ps and cs stand for PreparedStatement and CreateStatement, respectively.

22.9 LIMITATION ON THE NUMBER OF LITERAL VALUES AND THE SIZE OF OR STATEMENT

We did not find that there was a limit to the number of literal values or a limit to the textual size of the OR based statement. We tested with a large OR query with 10769 literal values and 247 kB in text length through both JDBC and SQL*Plus using the same product catalog database, and in both cases, the results were returned successfully without errors.

22.10 DEALING WITH MORE THAN 1000 LITERAL VALUES FOR AN IN BASED SQL QUERY

When the number of literal values in an IN based SQL query exceeds 1000, the following error occurs:

```
SQLException: java.sql.SQLException: ORA-01795: maximum
number of expressions in a list is 1000.
```

After some research, we found that many commercial applications actually ran into this problem with Oracle in production as well. This is a known limitation with the Oracle SQL parser, which remains true up to the latest version of 11g. Probably Oracle did not expect that any IN based SQL queries would exceed the 1000 literal value limit. The following section provides a recommendation for dealing with this issue within the application's code.

22.11 A RECOMMENDATION FOR DEALING WITH 1000 LITERAL VALUE LIMIT IN AN IN STATEMENT

Based on our tests, this limit can be overcome easily with multiple OR's as shown below:

```
SELECT /*+ index (av, avlval_pk) */ av.avlval_id, av.avltyp_id,
av.availability_value_name, av.availability_upper_value,
av.availability_lower_value, av.availability_value_desc100
FROM pc.availability_value av WHERE av.avlval_id IN (value11,
value12, ..., value1n, ...) OR av.avlval_id in (value21, value22, ...,
value2n, ...) OR (...);
```

In our test program, we used a java statement similar to the following pseudo code to break it:

```
if ( inCount % 999 == 0 ) {
    append a new IN segment with OR...
}
```

The data points with more than 1000 literal values shown in Figure 22.1 and Table 22.1 were obtained with OR’ed IN statements as described above.

22.12 SUMMARY

This chapter demonstrated that different options in coding the same SQL query to return the same result set may result in drastically different performance. With the given query and the options explored, we observed that:

- The IN based approach seems to be the far best performer. The only disadvantage with this approach is that it has a limitation of maximum 1000 literal values. However, this limitation can be overcome easily by cascading multiple IN’s with OR’s in the WHERE clause.
- With less than 100 literal values, the IN based approach with JDBC `prepareStatement` performed better than the same approach with JDBC `createStatement`. For larger number of literal values, however, the time saved from pre-parsing on the database side was offset by the time required to set the bind variables to concrete values on the application side; therefore, the IN based approach with JDBC `prepareStatement` actually performed much worse than the same approach with JDBC `createStatement` for large number of literal values.
- The IN based approach and OR based approach are not the same from the performance perspective for a large number of literal values. The test data obtained with this case study favored using IN in conjunction with JDBC `createStatement`. This probably had something to do with how each approach was optimized within Oracle’s cost-based optimizer. This might be contrary to the general belief in Oracle’s community that the IN and OR approaches were treated equally within Oracle.
- At least for this case study, the temptable approach didn’t seem to perform better than the IN and OR based approaches within the limit of 1000 literal values. Its overhead of populating the temptable was significant for a large number of literal values. However, the temptable approach seemed to be able to outperform the OR based approach beyond 1000 literal values.
- The temptable approach performed better without an index hint than with an index hint. This suggests that when comparing different approaches, it’s important to make sure that each approach was optimized in order to single out the truly best performer.

Before concluding this chapter, I would like to share the following comment from the text (p 254, Kyte, 2001) about the use of temporary tables in Oracle in general:

“I find many times people use temporary tables because they learned in other databases that joining too many tables in a single query is a ‘bad thing’. This is a practice

that must be unlearned for Oracle development. Rather than trying to out-smart the optimizer and breaking what should be a single query into three or four queries that store their sub results into temporary tables and then joining the temporary tables, you should just code a single query that answers the original question. Referencing many tables in a single query is OK; the temporary table crutch is not needed in Oracle for this purpose.”

I totally agree that this is a good guideline to observe, as it appears that my test data associated with this case study supports the notion clearly stated above.

RECOMMENDED READING

The following Oracle document covers JDBC performance and scalability in detail:

Oracle Corp, *Oracle Database JDBC Developer's Guide and References*, 11g Release 1 (11.1) B31224-04 (508 pages), July 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/java.111/b31224.pdf

Chapter 19, “Using Optimizer Hints,” of the following Oracle document discusses in more detail on Oracle optimizer hint:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

The following text has in-depth coverage of temporary tables:

T. Kyte, *Expert One-On-One: Oracle*, pp. 251–258, A Press, New York, 2001.

EXERCISES

- 22.1** Refer to Figure 22.1. Explain why the execution times associated with `IN` and `OR` based queries were more linear with the number of literal values than with the temporary table-based approach.
- 22.2** Compare Table 22.2 with Table 22.3. Explain why the temporary table-based query performed better without an index hint than with an index hint.
- 22.3** If you are working on an Oracle-based enterprise application, do you happen to have to deal with queries that have many literal values in their `WHERE` clause? If yes, which approach is taken in your application? Justify the approach taken in your application.
- 22.4** How would you deal with `IN` based queries that have more than 1000 literal values in its `WHERE` clause in order to avoid the ORA-01795 error?

23

Case Study: Data Access Paths (Double Buffering)

There is a theory which states that if ever for any reason anyone discovers what exactly the Universe is for and why it is here it will instantly disappear and be replaced by something even more bizarre and inexplicable. There is another that states that this has already happened.

—Douglas Adams

This case study illustrates the importance of a data access path to the performance and scalability of an Oracle-based enterprise application. When an Oracle database moves data to and from physical disks in order to fulfill user requests, a certain data access path is followed. The efficiency of a data access path determines how fast the Oracle database server can serve its users. Typically, it depends on the underlying data storage structure: whether it uses a file system or raw devices configured as raw partitions or raw logical volumes. Each of these two approaches has its pros and cons. A file system is simpler but may have significant overheads that may affect the performance and scalability of an Oracle database severely if not dealt with properly. Raw partitions or raw logical volumes are faster but incur significant complexities that are harder to cope with.

The purpose of this case study is not to prove or disprove one data access path or another. Instead, it illustrates how one can improve the performance and scalability of an Oracle database just by tweaking some external, configurable parameters of a given

file system used as the underlying data storage. By doing so, Oracle data movement is directed to a more efficient data access path which results in better performance and scalability.

Specifically, this case study demonstrates quantitatively how the efficiency of a file system-based data access path can be improved significantly if the double-buffering issue is circumvented properly. This double-buffering issue typically exists out-of-the-box with Oracle on various flavors of UNIX and Linux operating systems, and it's the first issue that you have to look out when you put up your Oracle-based application on a UNIX or Linux system. Since the test cases presented in this case study originated from my engagement with a real customer environment, this case study is both pedagogically and practically interesting.

To put it into perspective, this chapter begins with a discussion on data access paths in general. Then it describes three test environments: Solaris on Veritas, Solaris on UFS, and Windows on NTFS, followed by the quantitative test results obtained with each test environment. Finally, the moral of the case study is summarized to help you harvest the key lessons learned out of this case study.

This chapter consists of the following sections:

- Data Access Paths in General
- Test Environments
- Test Results with Solaris on Veritas
- Test Results with Solaris on UFS
- Test Results with Windows on NTFS
- Moral of the Case Study

Let's begin with the data access paths in general in the next section.

23.1 DATA ACCESS PATHS IN GENERAL

Before getting into the details of a data access path, let's review a very important concept: the concept of a logical volume. Whether using a raw device or a file system, in production, Oracle rarely deals with physical disks or volumes. Instead, Oracle deals with logical volumes: raw logical volumes or file system based logical volumes.

Figure 23.1 shows schematically how Oracle interacts with a UNIX or Linux kernel and how the kernel presents the underlying data storage to Oracle. The access hierarchy from the kernel to the physical drives or physical array is described as follows:

- Physical disk drives or disk array. These physical disks cannot be used directly. They have to be controlled and managed by the disk driver or RAID adapter software to form physical volumes. A physical volume is a single entity that is composed of one or more disk drives.

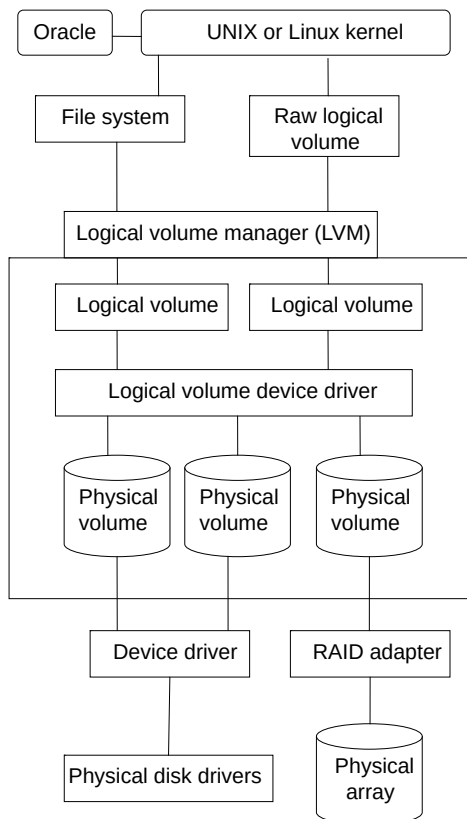


Figure 23.1 The concept of a logical volume.

- Logical volumes or partitions (note that the terms of partition and volume are interchangeable). Physical volumes are formed into logical volumes with the use of the logical volume device driver software. A logical volume is composed of one or more physical volumes.
- Logical volume manager (LVM). The LVM plays an indispensable role here: it presents the logical volumes either to the file system or to the kernel directly as raw logical volumes. Whether Oracle uses a file system or raw logical volumes is configured *a priori* at the Oracle level.

As was described previously, a minimum number of I/O waits would occur if an Oracle database uses raw devices in the format of raw partitions or raw logical volumes configured on a UNIX or Linux system. By using raw devices, all overheads and delays associated with a file system could be saved, thus alleviating potential I/O contentions and resulting in better performance and scalability. However, these benefits from using raw devices come at the cost of the complexities imposed on the database side. The ideal case is that one could achieve nearly raw device

performance while still using a file system to handle various types of I/O operations on Oracle's behalf. To help understand how such a goal can be achieved, in this section, we explain the following issues commonly shared among various flavors of UNIX and Linux file systems when used to support various data storage needs:

- Data buffering
- Inode locking
- Write-sync daemon

Let's begin with data buffering next.

23.1.1 Data Buffering

The concept of data buffering is best explained in Figure 23.2. In order to speed up the I/O operations for a calling application, all flavors of UNIX and Linux systems, except Windows, maintain data buffers in main memory. When a calling application issues a read request, the UNIX or Linux OS first checks its data buffer cache to see if data is already there. If it is, it's called a buffer hit and a physical read from the disk is saved. Otherwise, it's called a buffer miss and the data is read off the disk, stored into the file system buffer cache for future reuse, and then served to the calling application. As is seen, in both cases, the same data would be saved twice: once in the file system buffer cache and once in the database buffer cache. This is called double buffering.

In the case of a calling application issuing a write request, the logic is slightly more complicated than a read request. As is seen in Figure 23.3, first the OS kernel brings the data from the calling application's buffer cache to the file system's buffer cache. Typically, the calling application's write request is asynchronous, which means that the application will continue to execute without waiting for the data to be flushed from the file system's data buffer cache to disk. The OS kernel periodically flushes data in its buffer cache to disk in batches through a sync daemon that will be discussed shortly.

So why is it bad to have this data buffering at the OS file system level? Isn't it true that everyone has been taught about how caching could help performance? The problem is that the calling application or an Oracle database already has its own data buffer cache in its main memory, and this extra level of data buffering at the OS file system level can sometimes work against its purpose. Whether data buffering at the OS file system level is beneficial depends on the characteristics of the workload. Based on the data access pattern being sequential or random and the application being read or write intensive, a few potential scenarios are classified as follows:

- *Sequential, Read-intensive.* Sequential reads imply that data blocks are read from contiguous areas in the data buffer cache, which means that higher buffer hit ratios are more likely. This scenario favors data buffering at the OS file system level. Specific to Oracle, sequential reads typically occur with index fetching by ROWID's, and the associated wait event is *db file sequential read*.

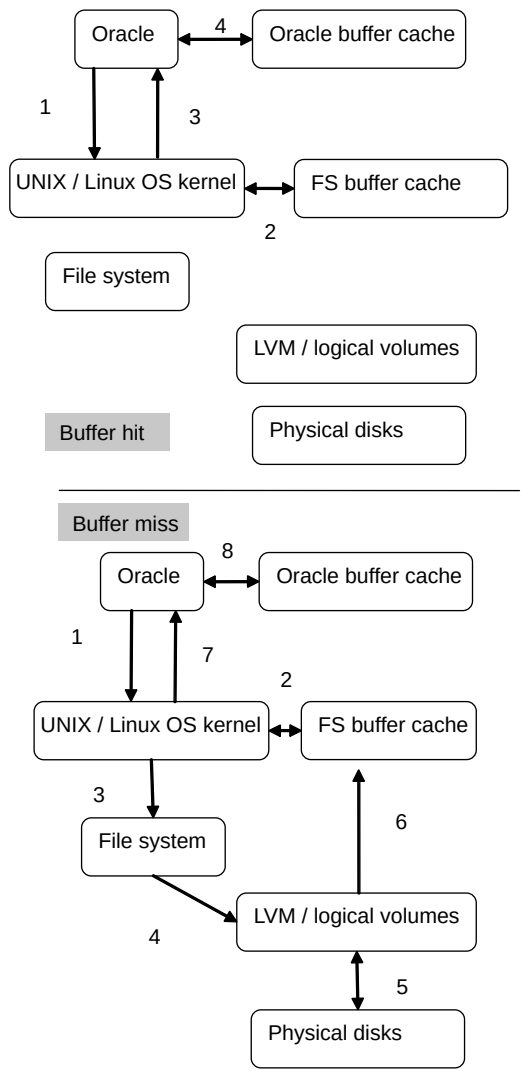


Figure 23.2 File system buffer hit versus buffer miss.

- *Random, Read-intensive.* Random reads imply that data blocks are read from noncontiguous areas in the data buffer cache, which means that higher buffer miss ratios are more likely. In this scenario, data buffering at the OS file system level causes an undesirable overhead, because eventually data items have to be fetched from physical disks.
- *Write or Update Intensive.* This is the worst scenario for data buffering at the OS file system level. In this case, a data block has to be written three times: once to the application or Oracle’s buffer cache, once to the file system’s cache, and once to the disk. Apparently, it’s impossible to avoid all these three writes for a same

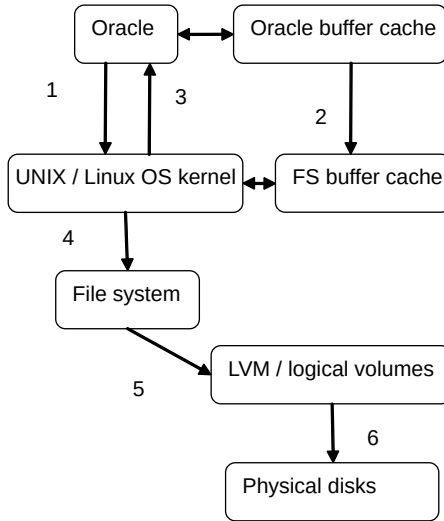


Figure 23.3 Asynchronous write.

data block, because the data has to be persisted to the physical disk. However, an option is available for circumventing this problem at the OS file system level: bypassing data buffering at the file system level so that a double-buffering sequence (once in Oracle and once in the file system) can be reduced to a single-buffering action (in Oracle only). In this case, step 2 in Figures 23.2 and 23.3 can be saved.

All flavors of UNIX and Linux operating systems support this option as a parameter in a file mounting command. This option is known as **DIRECT I/O (DIO)**, and it must be configured properly both with Oracle and with the OS-specific mounting command at the file system level.

Although this case study is about the data double-buffering issue on Solaris, it is helpful to extend our discussion to the other two factors that have a lot to do with the performance and scalability of an Oracle-based enterprise application as long as a file system gets involved. One of these factors is about inode locking, and the other is about the fast path that can avoid problems associated with inode locking. These are the subjects of the next two sections.

23.1.2 Inode Locking

What is an inode? This should not be an unfamiliar concept to those who understand a UNIX file system fairly well. With a UNIX file system, an inode is simply a data structure that maintains all the information for a file. When you query about a file using the `ls` command with proper options, you could get a number of attributes about the file queried, for example, the owner of the file, access permissions, size, the time of the last access or modification, etc. All this information comes from the inode stored

internally at the OS file system level. Besides, an inode also contains the information such as the access address of the file on disk.

We would care less if an inode is no more than an information container for a file. The problem is that whenever either a new data block needs to be written to a file, or a data block in the file needs to be updated, or an attribute of the inode needs to be modified, a lock has to be put on the inode of the file so that only one thread has the exclusive access to enforce data consistency. This is exactly the problem with inode locking: when one thread is updating an inode, the other threads that need to update the same inode must wait. This is especially relevant when a workload is write or update intensive. This is a separate issue from double-buffering discussed in the previous section.

Certainly, every flavor of UNIX operating system has its own way of coping with the inode locking issue. However, similar to the double buffering issue discussed in the previous section, there could be a double-locking issue when the calling application, Oracle in this case, has already implemented its own write-serialization model so that the write requests issued to a file system are guaranteed to be exclusive and no further such write-serialization measures are needed at the file system level. On the latest AIX operating systems, the combating strategy is called concurrent I/O or CIO. It can be enabled by a command which can be looked up from the relevant product documentation.

Note that the concurrent I/O scheme on AIX is implemented with multiple kernel threads, which are termed as asynchronous I/O (AIO) servers. The number of AIO servers is configurable externally at the command line. However, this exposes another potential performance and scalability problem: The context switches among multiple threads handling multiple asynchronous I/O requests can be a significant overhead for update/write intensive applications. The latest releases of the IBM AIX provide a “fastpath” option to circumvent the context switches associated with the kernel-threaded AIO servers. When the fastpath option is turned on, AIO servers are bypassed and AIO requests are routed to the logical volume manager (LVM) directly by the kernel even for Oracle databases that use file system based files rather than raw devices. Refer to Figure 23.1 about how the file system can be bypassed in this scenario. Of course, if raw partitions or raw logical volumes are used, there is no file system to bypass and therefore there is no inode locking issue. In this case, the fastpath is enabled by default as the only option. Once again, refer to Figure 23.1 to understand how a logical volume manager fits between the physical volumes and raw logical volumes.

In the next section, we discuss the third performance and scalability factor associated with a UNIX or Linux file system, which has something to do with the write-sync daemon mentioned previously.

23.1.3 Write-Sync Daemon

As was explained previously, the dirty pages in the file system’s buffer cache must be flushed out to the physical disks periodically with asynchronous I/Os (AIOs). This task is assigned to a write-sync daemon, which runs in the background. This extra

layer of overhead caused by looking up dirty pages in the file system's buffer cache is saved if the direct IO option is enabled. This is an extra benefit when the double-buffering issue is avoided.

In the next section, the various test environments associated with this case study are introduced. I have to mention that the purpose for this case study is to help understand the data buffering issue which could be a significant performance and scalability factor for Oracle databases running on UNIX/Linux platforms. There is no intention here to hint which platform or vendor is better or worse.

23.2 TEST ENVIRONMENTS

In this section, each test environment associated with this case study is described so that one can understand better about the double-buffering issue on different platforms and environments. Although the double-buffering issue occurs on UNIX/Linux platforms only, a Windows test environment is included as well as a reference to show the IO characteristics in an environment where double-buffering is not an issue at all.

Next, let's begin with the test environment of Solaris on Veritas first.

23.2.1 Solaris on Veritas

This was a customer environment, with which I actively engaged in order to help resolve a performance escalation. The environment was simple: an application server on Windows 2003, and an Oracle 10g database on a Solaris10 system with the Veritas vxfs file system for storing data. Figure 23.4 shows all the subsystems as well as the relevant specs for each subsystem.

In case you are less familiar with Veritas, a brief background introduction to Veritas is in order here. Veritas was claimed to be the first journaling file system (JFS). What is a JFS then? A JFS is a file system that uses a special area of the file system to keep track of the changes that it intends to make before they are committed permanently. The way it works is similar to using a regular journal to keep track of the tasks that are about to carry out and hence the term *journaling* for the name of JFS. The advantage with journaling is obvious that in case the system crashes, it can be brought back and continue to function from where it left off. However, journaling is neither an issue nor a required functionality for this case study. We are merely stating that a file system instead of a raw device was used in this test environment.

Next, the test environment with Solaris on UFS is introduced.

23.2.2 Solaris on UFS

This second test environment was from an internal lab test. In this setup, both the application server and the Oracle database server were on Solaris. The UFS was used for Oracle data storage. The term UFS stands for UNIX File System. The UFS is a file

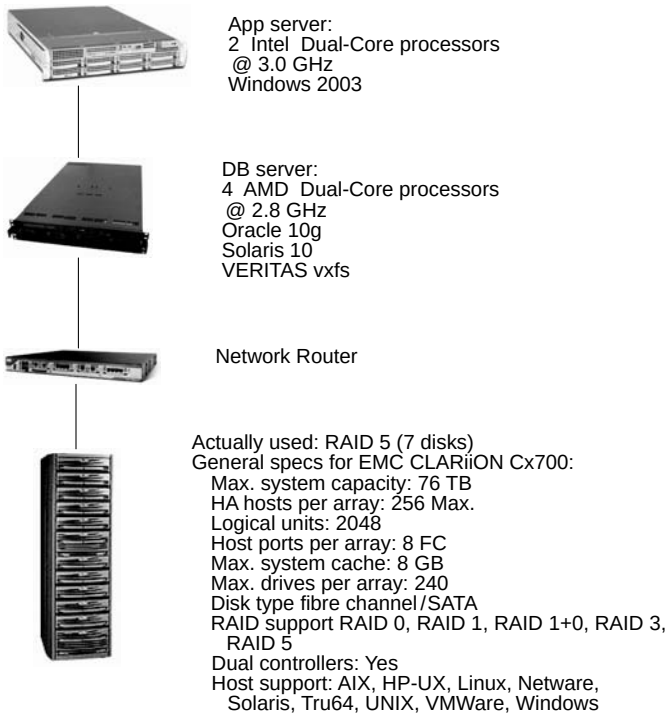


Figure 23.4 Test environment with Solaris on Veritas vxfs file system.

system used by many flavors of UNIX and Linux operating systems. It's also called the Berkley Fast File System or FFS. See Figure 23.5 for the specs of each subsystem involved in this test environment.

Next, a test environment based on Windows operating system is introduced.

23.2.3 Windows on NTFS

Figure 23.6 shows the third test environment with all systems on Windows. This is a very special test environment in the sense that a different file system, namely, the NTFS, was used for Oracle data storage. The term NTFS stands for New Technology File System. It's the standard file system on Windows platform, from Windows NT through Windows 2000, Windows XP, Windows 2003, Windows 2008, Windows Vista, and up to Windows 7. Its predecessors include the FAT (File Allocation Table) and HPFS (High Performance File System).

Since this case study is about Oracle performance on different file systems, it might be warranted to explore a little bit deeper into NTFS. In contrast to a UNIX file system, which depends on the two separate structures (inode and the actual file) to manage files, the NTFS uses a single data structure named Master File Table (MFT) to manage files. All file data (file name, creation date, access permissions, and contents) are stored as metadata in the MFT. The performance and scalability

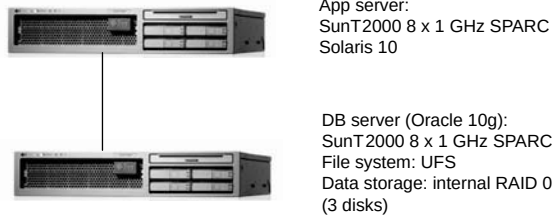


Figure 23.5 Test environment with Solaris on UFS.

of the NTFS have been improved drastically over time with some of the measures as introduced below:

- The MFT's structure supports algorithms specifically designed to minimize disk fragmentation. It has an entry consisting of a filename and a "file ID," which is the record number representing the file in the Master File Table. The file ID contains a reuse count to detect stale references. Stale references are cleaned regularly, which is similar to the garbage collection mechanism implemented with modern high-level programming languages such as Java and C#.
- NTFS uses B+ trees to index file system data. This allows an efficient, index-based, faster file look up, which is crucial not only for the operating system itself but also for all types of applications such as database systems that use NTFS for their data storage.
- NTFS supports journaling as well to guarantee the integrity of the file system metadata. This feature is similar to what is adopted in most major UNIX file systems.

In the next few sections, the test results from each test environment are presented. The approach here is purely data and fact based rather than opinion or wild guessing based. In order to show you the veracity of the case study presented in this chapter, the

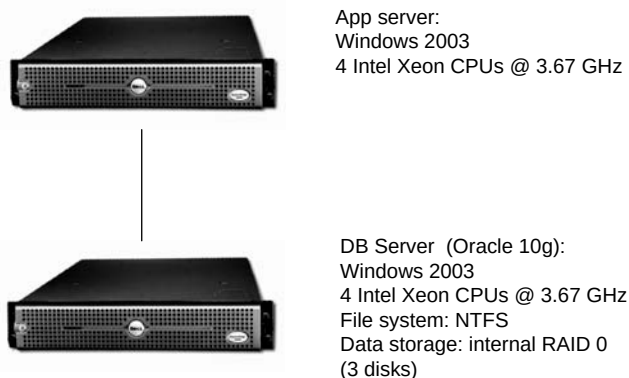


Figure 23.6 Test environment with Windows 2003 on NTFS.

relevant portions of the *original* AWR reports are given as they were. Including an entire AWR report wastes too much space, but just showing a few statistics doesn't serve the purpose well either.

However, if you are reviewing a whole AWR report given separately as an HTML file, here is what I would recommend about how to read and make the most of an AWR report:

1. Take a cursory look at all sections and identify the symptomatic areas first.
 2. Concentrate on the areas that have caught your attention from the previous step. Make a mental list of the problems that are most likely to be the potential bottlenecks.
 3. Do your drill-down exercise and nail down the problems further.
 4. Derive an attack strategy, implement your optimization or tuning (one at a time), run your test again and see if it helps. At this point, you can generate another AWR report and compare how those symptomatic areas have changed.
- This iterative approach has worked very well with me for many years.

Showing a selected portion of the AWR report rather than just listing a few specific statistics for each test configuration has some extra merit. You will see not only some directly related statistics such as disk read and write times but also how these metrics manifest themselves in the other sections of an AWR report such as top five timed events, and so on. If you feel that it's some sort of overwhelming to look at too much data or if you are not interested in exploring the other sections of an AWR report, you can jump to the IO stats section of the AWR report and digest the information presented there only.

Next, let's begin with the test results of the Solaris on Veritas test environment first.

23.3 TEST RESULTS WITH SOLARIS ON VERITAS

Without further delay, the portions of the original AWR reports obtained with the tests run on this test environment are listed below on a run-by-run basis. The first three runs were with data double-buffering on, resulting in large average read times of 145 milliseconds, 401 milliseconds, and 261 milliseconds, respectively. The last run was with double-buffering off, which resulted in an average disk read time of less than one millisecond. A brief analysis of all test results for all four test runs is given following the test results of the last test run.

23.3.1 Test Run #1—145 ms Average Read Time

This section presents the portions of the AWR report obtained from the first run with the Solaris on Veritas test environment while a complicated batch job was running. If you have not had a chance to troubleshoot Oracle performance and scalability issues using AWR reports, you can review Chapter 11 for a general coverage of what each section of an AWR report is about.

The total elapsed time and DB time for this AWR report were 539.88 minutes and 5,694.74 minutes, respectively. The following AWR sections are listed for this test run:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats
- Init.ora Parameters (this portion is redundant for all remaining three test runs and will not be repeated)

For this test run, note that the File IO Stats section shows that the average read time was 145 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
enq: HW - contention	88,735	54,262	612	15.9	Configuration
CPU time		52,023		15.2	
direct path read	213,178	31,308	147	9.2	User I/O
db file sequential read	251,978	29,360	117	8.6	User I/O
latch: cache buffers chains	93,325	17,193	184	5.0	Concurrency

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	101,995.46	29.85
DB CPU	52,022.74	15.23
parse time elapsed	7,027.85	2.06
hard parse elapsed time	1,676.05	0.49
PL/SQL execution elapsed time	1,167.24	0.34

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
User I/O	1,371,099	0.00	61,327	45	2.15
Configuration	92,636	2.04	54,464	588	0.15
Other	906,247	2.95	25,184	28	1.42
Concurrency	33,944,451	0.44	24,104	1	53.26
System I/O	1,121,296	0.00	8,534	8	1.76
Commit	639,441	0.00	5,587	9	1.00
Network	14,583,552	0.00	88	0	22.88
Application	41	0.00	6	137	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/oradir/ app.dbf	422,733	13	145.24	1.07

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/oradir/ app.dbf	1,784,965	55	91,058	21.55

init.ora Parameters

Parameter Name	Begin value	End value (if different)
cursor_sharing	SIMILAR	
db_block_size	8192	
db_file_multiblock_read_count	32	
db_files	255	
job_queue_processes	3	
log_buffer	2097152	
log_checkpoint_interval	10000	
open_cursors	255	
pga_aggregate_target	52428800	
processes	200	
query_rewrite_enabled	true	
query_rewrite_integrity	trusted	
resource_limit	TRUE	
sga_target	4194304000	
timed_statistics	TRUE	
transactions_per_rollback_segment	1	
undo_management	AUTO	

23.3.2 Test Run #2—401 ms Average Read Time

This section presents the portions of the AWR report obtained from the second run with the Solaris on Veritas test environment while the same batch job was repeated. This report was taken at a different time than the previous run with all same test conditions. The total elapsed time and DB time for this report were 720.57 minutes and 4,073.63 minutes, respectively. The following four AWR sections are listed:

- Top Five Timed Events
- Time Model Statistics

- Wait Class
- File IO Stats

For this test run, the File IO Stats section shows that the average read time was 401 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
direct path read	106,825	42,526	398	17.4	User I/O
enq: HW - contention	34,929	40,803	1,168	16.7	Configuration
CPU time		37,559		15.4	
db file sequential read	77,561	27,598	356	11.3	User I/O
db file parallel write	266,507	13,634	51	5.6	System I/O

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
DB CPU	37,559.21	15.37
sql execute elapsed time	4,104.13	1.68
parse time elapsed	417.26	0.17
PL/SQL execution elapsed time	345.35	0.14
hard parse elapsed time	108.18	0.04

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
User I/O	588,788	0.00	70,283	119	2.33
Configuration	35,989	17.28	40,867	1136	0.14
System I/O	545,469	0.00	14,195	26	2.16
Concurrency	41,001	1.30	1,267	31	0.16
Commit	252,577	0.04	698	3	1.00
Other	30,255	4.97	410	14	0.12
Network	5,877,583	0.00	24	0	23.29
Application	37	0.00	0	0	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/oradir/app.dbf	177,312	4	400.97	1.00

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/oradir/app.dbf	669,064	15	28,592	12.94

23.3.3 Test Run #3—261 ms Average Read Time

This section presents the portions of the AWR report obtained from the third run with the Solaris on Veritas test environment—once again while the same batch job was running. This report was taken at a different time than the previous two runs with all same test conditions. The total elapsed time and DB time for this report were 60.20 minutes and 394.47 minutes, respectively. The following four AWR sections are listed:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats

For this test run, the File IO Stats section shows that the average read time was 261 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		3,894		16.5	
direct path read	14,910	3,724	250	15.7	User I/O
db file parallel write	36,100	924	26	3.9	System I/O
latch: cache buffers chains	2,021	391	193	1.7	Concurrency
log file sync	53,995	138	3	.6	Commit

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
DB CPU	3,893.79	16.45
sql execute elapsed time	693.21	2.93
parse time elapsed	51.71	0.22
hard parse elapsed time	1.16	0.00
connection management call elapsed time	0.32	0.00

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
User I/O	109,430	0.00	3,749	34	2.04
System I/O	86,057	0.00	1,020	12	1.60
Concurrency	14,370	0.44	452	31	0.27
Commit	53,995	0.00	138	3	1.01
Other	8,821	7.90	129	15	0.16
Network	961,557	0.00	5	0	17.92
Configuration	152	0.00	1	7	0.00
Application	6	0.00	0	0	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/oradir/app.dbf	14,686	4	260.98	1.00

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/oradir/app.dbf	100,910	28	10,363	5.58

23.3.4 Test Run #4—0.98 ms Average Read Time

This section presents the portions of the AWR report obtained from the last run with the Solaris on Veritas test environment while the same batch job was running. In this case, double-buffering was turned off by configuring Oracle and Veritas vxfs properly. All other test conditions remained the same. The total elapsed time and DB time for this report were 179.62 minutes and 705.85 minutes, respectively. The following four AWR sections are listed:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats

For this test run, the File IO Stats section shows that the average read time was 0.98 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
latch: cache buffers chains	102,054	24,780	243	58.5	Concurrency
wait list latch free	322,171	6,176	19	14.6	Other
buffer busy waits	52,813	6,026	114	14.2	Concurrency
CPU time		3,436		8.1	
log file sync	446,566	548	1	1.3	Commit

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	37,348.45	88.19
DB CPU	3,435.74	8.11
parse time elapsed	877.68	2.07
hard parse elapsed time	39.80	0.09
hard parse (sharing criteria) elapsed time	6.03	0.01

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Concurrency	2,799,799	0.22	31,197	11	6.28
Other	343,144	2.54	6,346	18	0.77
Commit	446,566	0.00	548	1	1.00
System I/O	403,920	0.00	356	1	0.91
User I/O	494,780	0.00	103	0	1.11
Configuration	7,467	3.00	66	9	0.02
Network	10,924,986	0.00	63	0	24.52
Application	790	0.00	0	0	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/oradir/app.dbf	184,211	17	0.98	1.02

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/oradir/app.dbf	964,148	89	45,338	130.72

23.3.5 Analysis

This section analyzes the test results with the runs from the customer's Solaris on Veritas environment. First, we have to quantify what performance problem was escalated by the customer. The escalation was about a batch job that ran too slowly—an average throughput of only about 2.8 objects/second, which was far below the expected mid to high 20s.

Assuming we didn't know that the bottleneck was with the Veritas vxfs or double-buffering, how could one determine what was causing the problem just based on the first three runs with the three AWR reports provided by the customer? Here is a replay of the sequence of analyzing steps I went through:

1. I always start with the top five timed events and with these, I always look out the CPU time first. If the CPU time in terms of % Total Call Time is low, I can eliminate the possibility that database server CPU power is an issue. That's the case with this customer's environment: The CPU time was 15.2%, 15.4%, and 16.5% for those first three runs, respectively. However, a high CPU utilization could be either a good thing or a bad thing. We'll discuss more about this in the next chapter with another case study.
2. If the database server CPU power doesn't seem to be an issue, I look to the next potential bottleneck based on the average wait time data provided by an AWR report. Before I check everything else, I jump to the IO stats section to see what the average disk time looks like. Ideally, it should be in the range of 5 to 10 milliseconds or even much better if a high-end enterprise-class storage device is used. But even if I see 20 to 30 millisecond average disk access times, I would not bark immediately. I would check a few other things instead. However, in this case, hundreds of millisecond average disk access times shown from the first three AWR reports were sufficiently alarming for me to raise the storage issue to the customer, which I indeed did. In order to convince the customer that the problem was not with our product, I even showed the customer the typical average disk access times I got from my benchmarking tests with the comparable workload. That's with the Windows test environment to be presented shortly.

After working with the customer, the double-buffering issue was identified and resolved. When double-buffering was disabled in the same Solaris on Veritas test environment, the customer got an improved throughput of 23 objects/second with the same batch job they had run multiple times previously. That's an over 8X improvement relative to the previous low throughput of 2.8 objects/second. I requested the customer to send me the new AWR report associated with that run resulting in that high throughput of 23 objects/second with double-buffering disabled. That's the AWR report labeled test run #4 presented to you in the previous section, which verified a much improved average disk access time of 0.98 milliseconds.

In fact, I had a sizing formula for that type of batch job based on my long-time benchmarking tests, which stated that the expected throughput would be

approximately equal to 2 objects per second multiplied by the total CPU GHz power of the application server, which is the product of the # of CPUs and the CPU frequency. Based on that formula and according to the customer hardware specs as shown in Figure 23.4, I conveyed my predicted throughput of $2 \times 4 \text{ (CPUs)} \times 3 \text{ (GHz)} = 24$ objects/second to the customer prior to their run with double-buffering disabled. After their new throughput number of 23 objects/second with double-buffering disabled came out, the customer was very impressed and satisfied.

I have to explain that in general I expected my sizing formula would be reliable within $\pm 15\%$, or to be more precise, that customer could get a throughput number in the range of 20 to 28 objects/second. So, that extremely high accuracy of predicted 24 versus measured 23 could be a little bit luckier than I thought. However, this experience proved that customers would be able to recognize it if you treat them with honesty deeply rooted in a data-and-fact-based rather than opinion-based performance engineering approach. This approach is all this book is about and also all my software performance work has been about.

However, we did not present the other sections of the AWR reports, such as Top Five Timed Events, Time Model Statistics, and Wait Class, in vain. One can actually correlate the double-buffering problem with the categorical events listed in those sections. How this can be done is left as an exercise at the end of this chapter.

23.4 TEST RESULTS WITH SOLARIS ON UFS

This was a separate, internal lab test environment. As is shown in Figure 23.5, two Solaris systems were used for the application server and Oracle database server. The data storage was an internal RAID 0 configured with three high-performance physical disks. As an internal performance and scalability benchmarking effort on a real product, I carried out the end-to-end tests including configuring all components and collecting all AWR reports. The first run was with double-buffering turned on which was an out-of-the-box setting on the Solaris database server. The second run was with double-buffering turned off. A major difference between this test environment and the previous test environment was that the UFS instead of Veritas vxfs was used for Oracle data storage. However, it exhibited the similar double-buffering impact on the average disk access time, as is shown next.

23.4.1 Test Run #1—447 ms Average Read Time

The total elapsed time and DB time for this report were 294.94 minutes and 3,284.24 minutes, respectively. The following four AWR sections are listed:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats

- Init.ora Parameters (this portion is redundant for the second test run and will not be repeated)

For this test run, the File IO Stats section shows that the average disk read time was 447 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
db file parallel write	15,904	68,201	4,288	34.6	System I/O
buffer busy waits	307,325	57,558	187	29.2	Concurrency
enq: TX - index contention	37,399	31,234	835	15.9	Concurrency
free buffer waits	803,661	19,656	24	10.0	Configuration
log file switch (checkpoint incomplete)	20,752	18,875	910	9.6	Configuration

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	187,802.16	95.30
DB CPU	11,232.34	5.70
parse time elapsed	3,101.93	1.57
PL/SQL execution elapsed time	587.03	0.30
hard parse elapsed time	227.60	0.12

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Concurrency	353,321	18.60	96,364	273	0.13
System I/O	128,979	0.00	78,425	608	0.05
Configuration	855,071	98.70	65,651	77	0.32
User I/O	53,163	11.70	20,808	391	0.02
Other	71,575	91.73	7,366	103	0.03
Commit	404	34.16	175	434	0.00
Network	18,095,772	0.00	43	0	6.84

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/data3/app	30,784	2	446.81	3.23

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/data3/app	368,246	21	290,274	186.95

init.ora Parameters

Parameter Name	Begin value	End value (if different)
cursor_sharing	SIMILAR	
db_block_size	8192	
db_file_multiblock_read_count	16	
job_queue_processes	10	
open_cursors	500	
pga_aggregate_target	1706033152	
processes	150	
session_cached_cursors	100	
sga_target	1073741824	
undo_management	AUTO	

23.4.2 Test Run #2—10ms Average Read Time

This was the run with double-buffering turned off. The total elapsed time and DB time for this report were 121.48 minutes and 373.20 minutes, respectively. The following four AWR sections are listed:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats

For this test run, the File IO Stats section shows that the average disk read time was about 10 milliseconds, which is quite normal.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		15,804		70.6	
db file sequential read	673,433	6,747	10	30.1	User I/O
log file parallel write	346,570	6,589	19	29.4	System I/O
db file parallel write	514,344	4,685	9	20.9	System I/O
db file scattered read	38,246	599	16	2.7	User I/O

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	16,311.64	72.85
DB CPU	15,803.92	70.58
parse time elapsed	3,307.91	14.77
PL/SQL execution elapsed time	78.64	0.35
hard parse elapsed time	74.22	0.33

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
System I/O	869,011	0.00	11,375	13	0.27
User I/O	781,445	0.00	7,968	10	0.24
Concurrency	516,696	0.00	731	1	0.16
Network	22,124,925	0.00	51	0	6.83
Configuration	1,939	8.25	35	18	0.00
Other	119,586	66.00	21	0	0.04
Commit	278	0.00	10	36	0.00
Application	1	0.00	0	1	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	/data3/app	87,249	12	9.88	1.10

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/data3/app	86,111	12	214,008	1.07

23.4.3 Analysis

With this test case, I decided not to bore you with a tedious analysis. You look at the top five timed events, and it clearly says that the I/O was the number one factor. Then you directly jump to the IO stats section without looking anywhere else. You immediately spot a large average disk access time of 447 milliseconds there!

However, I do want to share with you that after double-buffering was disabled in this test environment, the throughput was improved from 49 objects/second to 145 objects/second—roughly a factor of 3 improvement, with a different type of workload than the previous customer test environment, though.

23.5 TEST RESULTS WITH WINDOWS ON NTFS

This test environment was actually set up internally to help investigate the customer performance escalation encountered in the Solaris on Veritas test environment presented first in this case study. This had also been my regular product performance and scalability benchmarking test environment. The setup used the workload comparable to what the customer applied in their environment. The intention was to verify how the product would perform with a comparable workload expect that the environment was based on Windows platform rather than Solaris platform. As a matter of fact, the very normal average read time of about 8 milliseconds from this test prompted me to request the customer to investigate why they got up to 400 millisecond average disk access time while I got only 8 milliseconds with my test. The customer’s Veritas consultant finally nailed it down to the double-buffering issue in that Solaris on Veritas test environment. As you already learned from the previous test runs specific to that customer environment, the average disk access time was reduced from hundreds of milliseconds to about one millisecond after double-buffering was turned off.

Since NTFS doesn’t implement a buffer cache at the file system level, the double-buffering issue does not exist in NTFS-based Windows test environment. Therefore, we have only one run to present in this section, as shown next.

23.5.1 Test Run—8 ms Average Read Time

The total elapsed time and DB time for this AWR report were 101.77 minutes and 457.08 minutes, respectively. The following AWR sections are listed for this test run:

- Top Five Timed Events
- Time Model Statistics
- Wait Class
- File IO Stats
- Init.ora Parameters)

For this test run, note that the File IO Stats section shows that the average disk read time was about 8 milliseconds.

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
log file sync	673,988	14,470	21	52.8	Commit
CPU time		8,702		31.7	
log file parallel write	474,878	2,214	5	8.1	System I/O
direct path read	203,103	1,321	7	4.8	User I/O
log file switch (checkpoint incomplete)	1,741	991	569	3.6	Configuration

Time Model Statistics (Top Five)

Statistic Name	Time (s)	% of DB Time
DB CPU	8,701.83	31.73
sql execute elapsed time	5,669.29	20.67
parse time elapsed	2,583.18	9.42
hard parse elapsed time	770.93	2.81
PL/SQL execution elapsed time	112.87	0.41

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Commit	673,988	0.45	14,470	21	1.02
System I/O	566,553	0.00	3,167	6	0.86
User I/O	1,117,658	0.00	2,170	2	1.69
Configuration	3,573	25.55	1,210	339	0.01
Concurrency	457,548	0.01	493	1	0.69
Other	18,493	35.01	364	20	0.03
Network	16,186,507	0.00	183	0	24.43
Application	538	0.00	1	1	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd
APP	D:\ORA\APP0.ORA	10,670	2	8.00	1.15

Tablespace	Filename	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	D:\ORA\APP0.ORA	45,785	7	3,943	5.44

init.ora Parameters

Parameter Name	Begin value	End value (if different)
compatible	10.2.0.1.0	
cursor_sharing	SIMILAR	
db_block_size	8192	
db_domain		
db_file_multiblock_read_count	16	
db_recovery_file_dest_size	2147483648	
job_queue_processes	10	
open_cursors	300	
pga_aggregate_target	243269632	
processes	150	
sga_max_size	926941184	
sga_target	767557632	
undo_management	AUTO	

23.5.2 Analysis

This test environment exhibited a very normal average disk read time of 8 milliseconds. However, if you look at the top five timed events presented there, the log file sync showed up as the top number one event. Also it had a large %Total Call Time of 52.8 with a 21 millisecond average wait time. You might ask why I left it there without doing something about it if it's already identified as the top bottleneck. This was clearly due to the limitation of the low-end, internal RAID I was using with this internal test environment. I left it for the customers to improve it with their more costly, high-end, production-class SAN storage they typically have in their environment. If you refer back to the top five timed events section of the test run #4 of the first test environment with Solaris on Veritas, you can immediately verify that the log sync average wait time was one millisecond only. That's because a 76 TB EMC CLARiiON Cx700 was working hard behind the scene!

23.6 MORAL OF THE CASE STUDY

Here is a list of the take-away's that might be valuable to you:

- Software performance work is not a wild guessing game. Anybody who is in the play needs to adhere to a data and fact-based approach, rather than an opinion-based approach. Customers will benefit more and be happier if more data-based performance best practices instead of opinions disguised as best practices are given to them.

- Even for a well-developed software product, how it performs and scales have a lot to do with the actual hardware it has to run. And even with very high-end hardware, it would perform worse or even much worse if misconfigured than optimally configured low-end hardware. This case study showed that the average disk read time was reduced from hundreds of milliseconds to about one millisecond just by disabling double-buffering on a very high-end storage system.
- Any enterprise software product would benefit tremendously if an internal performance and scalability benchmarking process is in place with sufficient rigor and precision imposed to the test environment design, workload construction, test execution, and lastly, with all optimization and tuning opportunities sought after persistently.

I hope you find this case study interesting and valuable. Several more case studies are presented in the next few chapters using the same consistent, coherent, data-and-fact-based approach.

RECOMMENDED READING

For understanding how UNIX works, my favorite is the following text by Maurice J. Bach. Although it was written some time ago, many concepts clearly explained there still apply.

Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice Hall Software Series, Prentice Hall P T R, Englewood Cliffs, 1990.

The following paper is very helpful for understanding AIX fast path in improving Oracle performance:

S. Kashyap, B. Olszewski, and R. Hendrickson, *Improving Database Performance with AIX Concurrent I/O*, IBM White Paper, 2003.

The following text is recommended for a general understanding of SAN:

Marc Farley, *Storage Networking Fundamentals: An Introduction to Storage Devices, Subsystems, Applications, Management, and File Systems (Vol 1)*, 1st Edition, Cisco Press, Indianapolis, 2004.

To be enriched more with a data-and-fact-based, rather than opinion-based approach to carrying out software performance work, refer to the author's other text listed below as well as other texts from the Wiley Series on Quantitative Software Engineering, for example, Dr. Bernstein's seminal work listed below:

Henry H. Liu, *Software Performance and Scalability: A Quantitative Approach*, Wiley, 2009.
L. Bernstein and C. M. Yuhas, *Trustworthy Systems through Quantitative Software Engineering*, Wiley, Hoboken, 2005.

EXERCISES

- 23.1** Refer to Figure 23.1. Identify the alternate pattern between hardware and software in the hierarchy from the UNIX or Linux kernel to physical disk level at the bottom.
- 23.2** Refer to Figures 23.2 and 23.3. Complete the step-by-step description by following the numbered labels in each scenario.
- 23.3** Consider a more efficient locking mechanism than inode locking, given what information an inode contains.
- 23.4** Refer to Figure 23.4. Focus on the general specs for the EMC CLARiiON Cx700. Describe what each spec is about.
- 23.5** Study the Top Five Timed Events, Time Model Statistics, and Wait Class presented for the first three test runs associated with the Solaris on Veritas test environment. Identify the categorical events that can be correlated to the double-buffering issue discussed in the text.
- 23.6** By comparing the statistics given for all three test environments, list the I/O characteristics of the data storage used for each test environment. This is an opportunity to learn how I/O performance can be drastically different, based on the level of the underlying storage device.
- 23.7** If you have your own approach to reading an AWR report, compare your approach with what is recommended in this chapter.
- 23.8** Explain why it is a good practice to adhere to the data-and-fact-based approach rather than an opinion-based approach to performing software performance work.

Case Study: Covering Index

And remember, no matter where you go, there you are.
—Confucius

This case study is presented to help demonstrate how an Oracle-based application could perform and scale poorly if proper indexes were missing. Such performance and scalability defects are hard to foresee, and rigorous, realistic performance and scalability testing must be part of a product development life cycle to help keep them at bay.

In order to resolve software performance and scalability issues efficiently, it's very necessary to follow a proven procedure such as the following recommended here:

1. *Getting to Know the Application Architecture.* This is the first step to work toward resolving a software performance and scalability issue. Whether it's an internal project or customer escalation, one needs to get to know the application architecture first. This would include knowing the type of the application, the type of the workload (OLTP or batch jobs), major components of the system (application server, database server, etc.), and the protocols that those components use to interact with each other to fulfill some application functionality. As a start point, this step provides background information for describing a performance and scalability issue.
2. *Quantifying the Problem.* After getting to know the application architecture, the next step is to quantify what the problem is with a proper metric, which

depends on the type of the application. For OLTP applications, the metric would most likely be slow system response times to user actions; and for batch job type of applications, the metric would be low throughput. Note that both of these metrics could deteriorate over time. The problem should be quantified so that the improvement can be assessed more accurately later after it's fixed.

3. *Analyzing Bottlenecks.* This step collects all necessary information for analyzing potential bottlenecks. Such information would include system level resource utilizations (CPU, memory, disk, and network), and component level logs, traces, and reports. For Oracle databases, the AWR report would be the most important source for analyzing bottlenecks if the database were causing the problem.
4. *Applying Optimizations/Tunings.* After the potential bottlenecks are identified, then the next step is to apply optimizations and/or tunings to remove the bottlenecks. A common mistake at this step is to guess wildly without depending on concrete data. This is unfortunate but happens all the time in reality.
5. *Verifying the Fixes.* Whatever optimizations and/or tunings are applied to resolving the identified bottlenecks, they need to be verified by conducting the same test with the same test conditions. The results should be evaluated using the same metric so that a quantitative improvement over the original test or observation can be arrived at. If the improvement is marginal or even negative, one should go back to step 3 and iterate the same procedure until the improvement is satisfactory. In order to make the optimization and/or tuning process as rigorous as possible, one should follow the two principles below:
 - a. Only one parameter is varied at a time. This principle is simple enough but not necessarily followed all the time. One might just change a few variables at a time, and at the end, even if a significant improvement was obtained, it's hard to tell which one actually contributed most and the other ones did not contribute substantially. Making a change in a customer environment is not as easy as one might think, as mostly a rigorous review and approval of every change are required at a customer's site. Besides, making unnecessary or inconsequential changes can make configuring and maintaining a production environment more complex than necessary.
 - b. Settling down to the optimal value of a parameter. Even when the first principle is observed, one should take multiple data points by varying the same parameter and find out the optimal value to settle down. This kind of data-driven approach to solving a software performance and scalability problem, if adopted persistently, is instrumental to fencing off propagating factitious "recommendations" to customers. Note that customers mostly won't be able to tell if a recommendation from a vendor is fact based or opinion based, or they are inclined to believe that it was fact based until they find out otherwise. Recommending "fixes" with no supportive data or verification can severely damage a vendor's image in customer's eyes and cause adverse consequences. Remember that software performance engineering is not a guess work.

The above procedure sets the framework for presenting all case studies throughout this text. Next, we get to the purpose of this case study: how covering indexes can improve the performance and scalability of an Oracle-based enterprise application significantly.

This chapter consists of the following sections:

- Getting to Know the Application Architecture
- Quantifying the Problems
- Analyzing the Bottlenecks
- Applying Optimizations/Tunings
- Verifying the Fixes
- Moral of the Case Study

Let's start with getting to know the application architecture with this case study.

24.1 GETTING TO KNOW THE APPLICATION ARCHITECTURE

The application architecture associated with this case study consisted of an application server and an Oracle database server, which was the same as shown in Figure 23.5 in Chapter 23. The application and database were deployed on two separate systems. The application was based on a real product. The workload was a batch job type, which kept inserting objects into the database continuously. The performance was measured with the throughput metric in terms of the number of objects inserted per second into the database over a period of time.

Next, we quantify the performance and scalability problems encountered with this real world case study.

24.2 QUANTIFYING THE PROBLEMS

The poor scalability with this application is illustrated in Figure 24.1, based on the test data obtained with a certain set of system and application configuration parameters. As is seen, the throughput was decreasing rapidly with time or with the number of objects inserted into the database. What factors were causing this poor scalability? That's the next step—analyzing bottlenecks—as is described in the next section.

24.3 ANALYZING BOTTLENECKS

For this particular case study, the CPU utilizations for both the application server and database server were collected. It is seen from Figure 24.2 that on average the application server CPUs were only about 4% busy while the database server CPUs were about 36% busy, both averaged across all CPUs at the system level. This

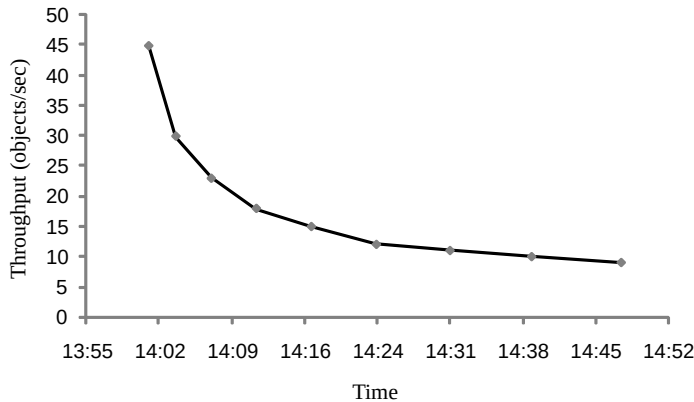


Figure 24.1 Measured poor performance and scalability of an Oracle-based enterprise application.

information indicates that neither the application server nor the database server were undersized.

Since the database was run on Oracle 10g, an AWR report could help pinpoint down the potential bottlenecks. As a matter of fact, the AWR report discussed in detail in Chapter 11 was taken from this test. Therefore, we would not duplicate it here.

To analyze the potential bottlenecks for this case study, refer to the Top Five Timed Events shown in Section 11.3.5. It is seen that out of 100% *Total Call Time*, 96.4% was attributed to *CPU time*. Further drilling-down into the Section 11.6.3 of *SQL ordered by Gets* revealed that five SQL statements incurred large number of buffer gets or logical reads. From there one can follow to the Section of 11.6.9 of Complete List of SQL Text, which finally revealed the SQLs that were causing excessive buffer gets. These five SQLs are listed below to help explain why covering indexes could cure this excessive buffer gets issue, which is the subject of the next section.

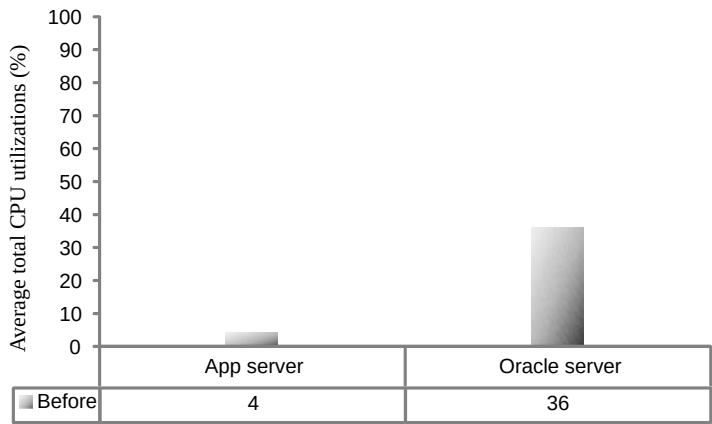


Figure 24.2 Average total CPU utilizations with the app server and the Oracle server associated with the poor performance and scalability shown in Figure 24.1.

Query 1: SELECT documentId, classId, dataGroupId, consistencyId
FROM objectTable WHERE objectId = <value>;

Query 2: SELECT documentId
FROM objectTable WHERE objectId = <value>;

Query 3: SELECT documented, objectId
FROM objectAssociationTable WHERE objectId = <value>;

Query 4: SELECT documentId, objectId
FROM objectTable WHERE objectId = <value>;

Query 5: SELECT documentId, sourceObjectId, destObjectId, objectId,
consistencyId FROM objectAssociationTable WHERE
destObjectId = <value> and classId = <value>;

Note that the entity <value> in each WHERE-clause represents the actual value for a specific column. What actual values were there is not important for our analysis here, and are therefore masked out with the entity <value>.

24.4 APPLYING OPTIMIZATIONS/TUNINGS

Based on the fact that the top event was the 96.4% CPU time caused by excessive buffer gets out of those five SQLs listed in the preceding section, the following three covering indexes were created to help prevent too much data from being loaded into the buffer cache:

- Index 1 on the columns of objectId, documentId, classId, dataGroupId, and consistencyId of the objectTable
- Index 2 on the columns of objectId and documentId of the objectAssociationTable
- Index 3 on the columns of destObjectId, classId, documentId, sourceObjectId, objectId and consistencyId of the objectAssociationTable

Then the same test was repeated and the problem was cured, as described in the next section.

24.5 VERIFYING THE FIXES

The efficacy of curing the poor scalability of this application using covering indexes was verified from several perspectives. First, the most direct way was to check the throughput against Figure 24.1. As shown in Figure 24.3, the throughput was much higher and stayed flat with more and more objects inserted into the database iteration after iteration. Next, Figure 24.4 shows that the database server CPU usage reduced from 36% to 10% while the application server CPU usage increased from 4% to 25%,

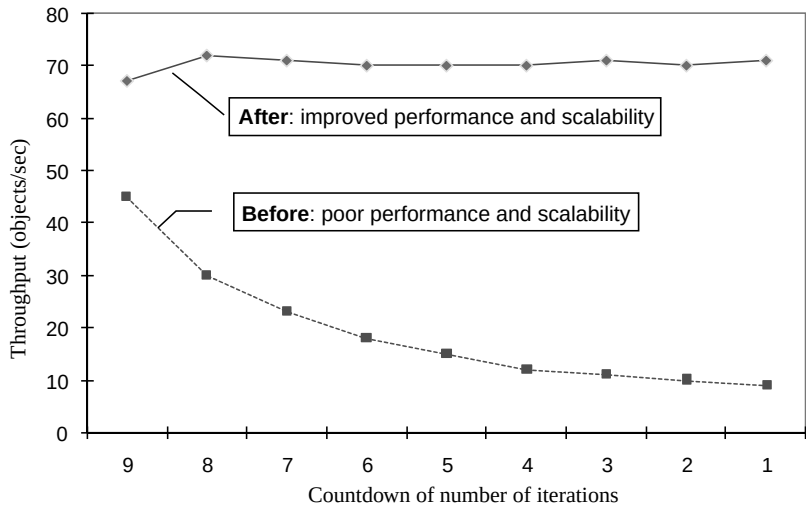


Figure 24.3 Improved performance and scalability after three covering indexes were added.

which implies that the application server was busier because the database server returned the query results much faster than before the covering indexes were applied.

The fix was lastly verified by comparing the two AWR reports taken before and after the fix was applied. To save space, only the relevant sections of the AWR report newly taken with covering indexes applied are shown below (compare the sections of this AWR report with those presented in Chapter 11 to see how they changed after covering indexes were applied). It is seen that CPU time had been reduced from 96.4% to 50.8% while the buffer gets had been reduced by almost three orders of magnitude. This was a perfect outcome that verifications from the actual performance throughput, from the system CPU utilizations, and from the database data access perspective all consistently reinforced the conclusion that the original poor performance and scalability were indeed cured.

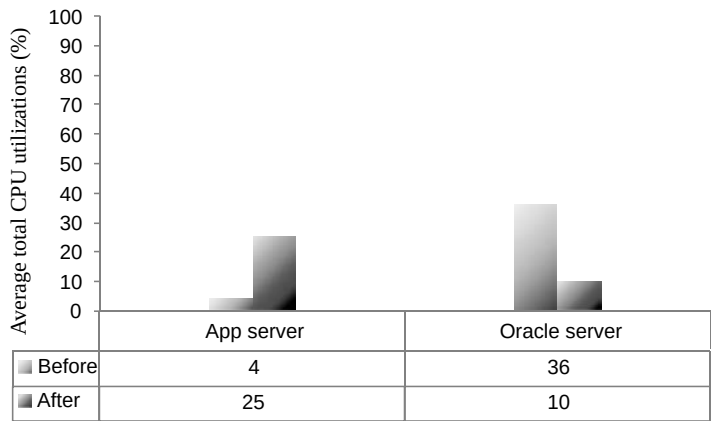


Figure 24.4 Average total CPU utilizations with the app server and the Oracle server associated with the improved performance and scalability shown in Figure 24.3.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ObStore	2563315763	ObStore	1	10.2.0.1.0	NO	snt2k-1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	234	27-Jun-07 15:49:43	79	4.3		
End Snap:	235	27-Jun-07 16:02:28	79	4.3		
Elapsed:		12.74 (mins)				
DB Time:		56.22 (mins)				

24.5.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	760M	776M	Std Block Size:	8K
Shared Pool Size:	236M	220M	Log Buffer:	10,344K

Load Profile

	Per Second	Per Transaction	
Redo size:	1,274,701.73	6,066.34	
Logical reads:	13,051.79	62.11	
Block changes:	4,303.20	20.48	
Physical reads:	0.46	0.00	
Physical writes:	116.28	0.55	
User calls:	4,082.98	19.43	
Parses:	1,869.89	8.90	
Hard parses:	1.15	0.01	
Sorts:	202.19	0.96	
Logons:	0.00	0.00	
Executes:	1,875.72	8.93	
Transactions:	210.13		
% Blocks changed per Read:	32.97	Recursive Call %:	8.52
Rollback per transaction %:	0.00	Rows per Sort:	0.84

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.57	Redo NoWait %:	99.99
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	99.85	Soft Parse %:	99.94
Execute to Parse %:	0.31	Latch Hit %:	98.99
Parse CPU to Parse Elapsd %:	86.05	% Non-Parse CPU:	77.92

Shared Pool Statistics

	Begin	End
Memory Usage %:	95.27	90.59
% SQL with executions>1:	34.00	83.61
% Memory for SQL w/exec>1:	54.13	91.52

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		1,713		50.8	
log file parallel write	36,463	566	16	16.8	System I/O
Streams AQ: enqueue blocked on low memory	2	523	261,699	15.5	Configuration
db file parallel write	4,709	69	15	2.0	System I/O
library cache pin	946	40	43	1.2	Concurrency

24.5.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 3373.4s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
DB CPU	1,712.89	50.78
sql execute elapsed time	685.75	20.33
parse time elapsed	453.24	13.44
hard parse elapsed time	17.70	0.52
hard parse (sharing criteria) elapsed time	1.36	0.04
hard parse (bind mismatch) elapsed time	0.45	0.01
PL/SQL execution elapsed time	0.40	0.01
PL/SQL compilation elapsed time	0.20	0.01
sequence load elapsed time	0.12	0.00
repeated bind elapsed time	0.04	0.00
DB time	3,373.36	
background elapsed time	1,249.91	
background cpu time	33.43	

Wait Class

Y	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
System I/O	42,430	0.00	644	15	0.26
Configuration	229	0.87	529	2310	0.00
Concurrency	47,985	0.03	73	2	0.30
Network	2,470,022	0.00	44	0	15.37
Other	11,224	81.73	8	1	0.07
User I/O	133,250	0.00	2	0	0.83

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file parallel write	36,463	0.00	566	16	0.23
Streams AQ: enqueue blocked on low memory	2	100.00	523	261699	0.00
db file parallel write	4,709	0.00	69	15	0.03
library cache pin	946	0.00	40	43	0.01
SQL*Net more data to client	326,025	0.00	32	0	2.03

Background Wait Events

- ordered by wait time desc, waits desc (idle events last)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file parallel write	36,463	0.00	566	16	0.23
Streams AQ: enqueue blocked on low memory	2	100.00	523	261699	0.00
db file parallel write	4,709	0.00	69	15	0.03
control file parallel write	381	0.00	9	23	0.00
control file sequential read	192	0.00	0	1	0.00
events in waitclass Other	1,878	0.05	0	0	0.01
log file single write	4	0.00	0	8	0.00
direct path write	12	0.00	0	3	0.00
db file sequential read	2	0.00	0	10	0.00
log file sequential read	4	0.00	0	5	0.00
direct path read	12	0.00	0	0	0.00
rdbs ipc message	11,269	24.70	6,948	617	0.07
Streams AQ: waiting for time management or cleanup tasks	1	100.00	3,870	3869895	0.00
Streams AQ: qmn slave idle wait	89	0.00	1,487	16707	0.00
pmon timer	256	100.00	743	2903	0.00
Streams AQ: qmn coordinator idle wait	64	45.31	724	11314	0.00
smon timer	2	100.00	428	213772	0.00

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	8,655
AVG_IDLE_TIME	67,582
AVG_IOWAIT_TIME	0
AVG_SYS_TIME	1,895
AVG_USER_TIME	6,742
BUSY_TIME	277,791
IDLE_TIME	2,163,471
IOWAIT_TIME	0
SYS_TIME	61,396
USER_TIME	216,395
LOAD	1
OS_CPU_WAIT_TIME	7,400
RSRC_MGR_CPU_WAIT_TIME	0
VM_IN_BYTES	0
VM_OUT_BYTES	0
PHYSICAL_MEMORY_BYTES	17,099,644,928
NUM_CPUS	32

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
ObStore	3,361.10	1,701.00	240	9,951,419
SYS\$USERS	16.20	15.70	85	24,905
SYS\$BACKGROUND	0.00	0.00	14	5,664

Service Wait Class Stats

- Wait Class info for services in the Service Statistics section.
- Total Waits and Time Waited displayed for the following wait classes: User I/O, Concurrency, Administrative, Network
- Time Waited (Wt Time) in centisecond (100th of a second)

Service Name	User I/O Total Wts	User I/O Wt Time	Concurrency Total Wts	Concurrency Wt Time	Admin Total Wts	Admin Wt Time	Network Total Wts	Network Wt Time
ObStore	133102	122	47985	7290	0	0	2470018	4420
SYS\$USERS	122	54	0	0	0	0	4	0
SYS\$BACKGROUND	26	5	0	0	0	0	0	0

24.5.3 SQL Statistics

SQL Ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
53	51	22,004	0.00	1.58	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
50	50	21,726	0.00	1.49	c070097bggvup	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
49	49	20,505	0.00	1.45	4pb2scqxpjgcn	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
31	30	10,665	0.00	0.91	1mca8yahft04j	app.exe	INSERT INTO T119 (C200000001, ...
28	28	9,855	0.00	0.84	4natsk0zggjbb	app.exe	INSERT INTO T119 (C200000001, ...

SQL Ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
51	53	22,004	0.00	1.58	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
50	50	21,726	0.00	1.49	c070097bggvup	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
49	49	20,505	0.00	1.45	4pb2scqxpjgcn	app.exe	SELECT C1, C2, C3, C4, C5, C6,...
30	31	10,665	0.00	0.91	1mca8yahft04j	app.exe	INSERT INTO T119 (C200000001, ...
28	28	9,855	0.00	0.84	4natsk0zggjbb	app.exe	INSERT INTO T119 (C200000001, ...

SQL Ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 9,979,944
- Captured SQL account for 77.5% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
895,663	22,004	40.70	8.97	50.68	53.44	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
434,670	22,087	19.68	4.36	25.84	25.87	d0cp76dp8mfhq	app.exe	UPDATE T331 SET C459 = EMPTY_C...
433,035	22,135	19.56	4.34	23.07	27.98	ffbpj4hpp19ty	app.exe	UPDATE T331 SET C456 = EMPTY_C...
361,704	22,057	16.40	3.62	19.56	19.56	1tdwt74ht67q6	app.exe	SELECT C459 FROM T331 WHERE C1...
354,846	22,131	16.03	3.56	17.58	21.74	4su3ayym7r0hu	app.exe	SELECT C456 FROM T331 WHERE C1...

SQL Ordered by Reads

- Total Disk Reads: 348
- Captured SQL account for 50.6% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
114	22,004	0.01	32.76	50.68	53.44	7s4p3jnkpc5sy	app.exe	INSERT INTO T62 (C7, C49000900...
26	21,600	0.00	7.47	22.66	25.63	5k3hvuaqvkbmr	app.exe	INSERT INTO T60 (C490009000, C...
8	20,520	0.00	2.30	19.62	19.93	64n9254p0hpwk	app.exe	INSERT INTO T109 (C179, C40012...
7	22,005	0.00	2.01	12.15	12.21	fazmqf9m11gj7	app.exe	INSERT INTO H62 (entryId, T0, ...
7	11,070	0.00	2.01	15.00	15.00	gxftzdz310ahj	app.exe	UPDATE T119 SET C400131300="S...

SQL Ordered by Executions

- Total Executions: 1,434,254
- Captured SQL account for 75.9% of Total

Executions	Rows Processed	Rows per Exec	CPU per Exec (s)	Elap per Exec (s)	SQL Id	SQL Module	SQL Text
43,995	43,985	1.00	0.00	0.00	gsmv6cvsysnys	app.exe	SELECT T119.C1, C400079600, C4...
43,446	43,445	1.00	0.00	0.00	g3m1pv9cpw6h0	app.exe	SELECT C400079600 FROM T119 WH...
41,026	41,019	1.00	0.00	0.00	079k3hnnw05pq4	app.exe	SELECT C400079600 FROM T147 WH...
22,140	22,140	1.00	0.00	0.00	8qw8ckf1jvtfx	app.exe	INSERT INTO H119 (entryId, T0,...
22,139	22,140	1.00	0.00	0.00	76dbmn6tr9rx7	app.exe	INSERT INTO H331 (entryId, T0,...

SQL Ordered by Parse Calls

- Total Parse Calls: 1,429,794
- Captured SQL account for 76.1% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
44,009	43,995	3.08	<u>gsmv6cvsysnys</u>	app.exe	SELECT T119.C1, C400079600, C4...
43,470	43,446	3.04	<u>g3m1pv9cpw6h0</u>	app.exe	SELECT C400079600 FROM T119 WH...
41,039	41,026	2.87	<u>079k3hnm05pq4</u>	app.exe	SELECT C400079600 FROM T147 WH...
22,140	22,131	1.55	<u>4su3ayym7r0hu</u>	app.exe	SELECT C456 FROM T331 WHERE C1...
22,140	22,126	1.55	<u>5pgcf7ps199y6</u>	app.exe	SELECT T380.C1, T380.C1, C1000...

SQL Ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
482	6	<u>8xxw6k8rbv5bs</u>	app.exe	SELECT T482.C1 FROM T482 WHERE...
193	3	<u>0by3hna8a3bax</u>	app.exe	SELECT T451.C1 FROM T451 WHERE...

A Complete List of SQL Text (only one giant SQL is shown here. Other SQLs are omitted to save space)

SQL Id	SQL Text
1gbu3raazg6dy	SELECT C1, C2, C3, C4, C5, C6, C7, C8, "SYS_B_00", C112, C179, C60513, TO_CHAR(C27012011V, "SYS_B_01"), C27012011C, C27012011D, TO_CHAR(C27012011USD, "SYS_B_02"), TO_CHAR(C27012011EUR, "SYS_B_03"), TO_CHAR(C27012011GBP, "SYS_B_04"), TO_CHAR(C27012011JPY, "SYS_B_05"), TO_CHAR(C27012011CAD, "SYS_B_06"), C100000093, C200000001, C200000003, C200000004, C200000005, C200000006, C200000007, C200000012, C200000013, C200000020, C200000003, C210000000, C230000009, C240000007, C240000025, C240001002, C240001003, C240001004, C240001005, C240001008, C240001009, C260000000, C260000001, C260000002, C260000003, C260000004, C260000005, C260000104, C260000106, C260000107, C260100001, C260100002, C260100004, C260100006, C260100007, C260100009, C260100010, C260100015, C260300004, C260400003, C260442101, TO_CHAR(C260600001V, "SYS_B_07"), C260600001C, C260600001D, TO_CHAR(C260600001USD, "SYS_B_08"), TO_CHAR(C260600001EUR, "SYS_B_09"), TO_CHAR(C260600001GBP, "SYS_B_10"), TO_CHAR(C260600001JPY, "SYS_B_11"), TO_CHAR(C260600001CAD, "SYS_B_12"), C260700001, C260700002, C260700003, C260700004, C260700009, C260800002, C260800052, C263000017, C263000050, TO_CHAR(C270002000V, "SYS_B_13"), C270002000C, C270002000D, TO_CHAR(C270002000USD, "SYS_B_14"), TO_CHAR(C270002000EUR, "SYS_B_15"), TO_CHAR(C270002000GBP, "SYS_B_16"), TO_CHAR(C270002000JPY, "SYS_B_17"), TO_CHAR(C270002000CAD, "SYS_B_18"), TO_CHAR(C270002009V, "SYS_B_19"), C270002009C, C270002009D, TO_CHAR(C270002009USD, "SYS_B_20"), TO_CHAR(C270002009EUR, "SYS_B_21"), TO_CHAR(C270002009GBP, "SYS_B_22"), TO_CHAR(C270002009JPY, "SYS_B_23"), TO_CHAR(C270002009CAD, "SYS_B_24"), TO_CHAR(C270002010V, "SYS_B_25"), C270002010C, C270002010D, TO_CHAR(C270002010USD, "SYS_B_26"), TO_CHAR(C270002010EUR, "SYS_B_27"), TO_CHAR(C270002010GBP, "SYS_B_28"), TO_CHAR(C270002010JPY, "SYS_B_29"), TO_CHAR(C270002010CAD, "SYS_B_30"), TO_CHAR(C270002051V, "SYS_B_31"), C270002051C, C270002051D, TO_CHAR(C270002051USD, "SYS_B_32"), TO_CHAR(C270002051EUR, "SYS_B_33"), TO_CHAR(C270002051GBP, "SYS_B_34"), TO_CHAR(C270002051JPY, "SYS_B_35"), TO_CHAR(C270002051CAD, "SYS_B_36"), C300151500, C300340400, C300421900, C300483500, C300557000, C300559600, C300559700, C300632000, C300714900, C300826900, C300927600, C301002800, C301002900, C301004200, C301047700, C301047800, C301053300, C301071700, C301089100, C301118000, C301136600, C301136800, C301136900, C301137000, C301137100, C301137200, C301137300, C301137400, C301139800, TO_CHAR(C301164700V, "SYS_B_37"), C301164700C, C301164700D, TO_CHAR(C301164700USD, "SYS_B_38"), TO_CHAR(C301164700EUR, "SYS_B_39"), TO_CHAR(C301164700GBP, "SYS_B_40"), TO_CHAR(C301164700JPY, "SYS_B_41"), TO_CHAR(C301164700CAD, "SYS_B_42"), C301172600, C301186800, C301205200, C301517900, C301518000, C400079600, C400124500, C400127400, C400129100, C400129200, C400131200, C400131300, C400137200, C490000200, C490001289, C4900021100, C530001100, C530010100, C530010200, C530014300, C530014400, C530014500, C530019500, C530031600, C530032500, C1000000001, C1000000010, C1000000150, C1000000162, C1000000163 FROM T172 WHERE E0 = "SYS_B_43" and E1 = "SYS_B_44" and E2 = "SYS_B_45"

24.5.4 IO Stats

Tablespace IO Stats

- ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	204	0	4.95	1.00	46,323	61	39,101	0.75
UNDOObs	43	0	0.47	1.00	1,767	2	3,821	0.12
SYSAUX	52	0	6.35	1.00	401	1	0	0.00
SYSTEM	40	0	7.00	1.00	275	0	0	0.00
TEMP	0	0	0.00		9	0	0	0.00

File IO Stats

- ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	/data3/obs/app2.dbf	23	0	5.22	1.00	16,308	21	13,535	0.75
APP	/data3/obs/app3.dbf	78	0	2.95	1.00	14,805	19	10,460	0.98
APP	/data3/obs/app1	103	0	6.41	1.00	15,210	20	15,106	0.60
SYSAUX	/data3/obs/sysaux.dbf	52	0	6.35	1.00	401	1	0	0.00
SYSTEM	/data3/obs/system.dbf	40	0	7.00	1.00	275	0	0	0.00
TEMP	/data3/obs/temp.dbf	0	0			9	0	0	
UNDOObs	/data3/obs/undo.dbf	43	0	0.47	1.00	1,767	2	3,821	0.12

24.5.5 Buffer Pool Statistics

- Standard block size Pools D: default, K: keep, R: recycle
- Default Pools for other block sizes: 2k, 4k, 8k, 16k, 32k

P	Number of Buffers	Pool Hit%	Buffer Gets	Physical Reads	Physical Writes	Free Buff Wait	Writ Comp Wait	Buffer Busy Waits
D	96,030	100	9,923,557	315	44,600	0	0	42,980

24.5.6 Wait Statistics

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	36,353	29	1
1st level bmb	2,610	1	0
undo header	2,255	0	0
undo block	1,566	0	0
segment header	107	0	0
2nd level bmb	37	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	1,213	1,213	0	1,213	1	1.17
TX-Transaction	168,641	168,671	0	69	0	2.88
HW-Segment High Water Mark	1,312	1,312	0	108	0	1.06
FB-Format Block	974	974	0	39	0	0.41
TX-Transaction (allocate ITL entry)	3	3	0	3	0	1.67

24.5.7 init.ora Parameters

Parameter Name	Begin value	End value (if different)
_wait_for_sync	FALSE	
compatible	10.2.0.1.0	
core_dump_dest	/data3/obs/cdump	
cursor_sharing	SIMILAR	
db_block_size	8192	
db_file_multiblock_read_count	16	
db_name	ObStore	
job_queue_processes	10	
open_cursors	500	
pga_aggregate_target	1706033152	
processes	150	
remote_login_passwordfile	EXCLUSIVE	
session_cached_cursors	100	
sga_target	1073741824	
undo_management	AUTO	
undo_tablespace	UNDObs	
user_dump_dest	/data3/app/oracle/admin/obs/udump	

24.6 MORAL OF THE CASE STUDY

This chapter started with a proven software performance and scalability troubleshooting procedure that consists of the following five steps:

- Getting to know the application architecture

- Quantifying the problems
- Analyzing bottlenecks
- Applying optimizations and tunings
- Verifying the fixes

The above procedure was then applied to a case study that demonstrated quantitatively how covering indexes cured the poor performance and scalability of a real application.

Before concluding this chapter, it was pointed out that some Oracle performance tuning texts suggested that the top event of the extremely high CPU time could be easily resolved by adding more CPUs to the database server in the excuse of “hardware is cheaper than development time,” but this case study has proved otherwise. Both software vendors and customers would welcome more such fixes than the recommendation of purchasing more powerful hardware, which may not fix the problem at all.

RECOMMENDED READING

For more in-depth information about Oracle indexing, refer to the following Oracle document:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 24.1** Explain the concept of Oracle buffer gets. What’s the direct consequence of excessive buffer gets? What’s the criterion to determine whether buffer gets are excessive or not?
- 24.2** Is adding more CPUs always the solution to fighting excessive buffer gets? Explain.
- 24.3** How can one use an AWR report to gauge whether the database server is properly sized?
- 24.4** Since buffer gets are read operations on memory, while explaining using covering indexes to cure excessive buffer gets, I was asked by a developer that “how come reading from memory is a bad thing and should we make it read more from disk?”. How could you explain better than I was able to if you were asked the same question?

25

Case Study: ***CURSOR_SHARING***

A man's country is not a certain area of land, of mountains, rivers, and woods, but it is a principle and patriotism is loyalty to that principle.

—George William Curtis

CURSOR_SHARING is one of the key initialization parameters that could affect the performance and scalability of an Oracle-based enterprise application. It can be set to one of the three settings: EXACT (default), SIMILAR, and FORCE. It must be set properly depending on whether bind variables are used in the application.

With a real product that does not use bind variables, this case study shows quantitatively the dramatic performance improvement by setting CURSOR_SHARING to FORCE or SIMILAR. Through this case study, you will not only learn the effects of this parameter, but also get more familiarized with how to read an AWR report in addition to what you have learned so far about how to troubleshoot an Oracle performance problem using AWR reports.

This chapter consists of the following sections:

- The Concept of Bind Variables
- Oracle CURSOR_SHARING Parameter
- Getting to Know the Application Architecture
- Quantifying the Problems

- Analyzing Bottlenecks
- Applying Tuning: CURSOR_SHARING = FORCE
- Applying Tuning: CURSOR_SHARING = SIMILAR
- Moral of the Case Study

Let's start with understanding the concept of a bind variable in Oracle's context next.

25.1 THE CONCEPT OF A BIND VARIABLE

To understand the concept of a bind variable, consider a table named CUSTOMER with the attributes of customerID, first_name, last_name, email_address, and so on. Let's say, as part of the application logic, it's necessary to retrieve the name of a customer based on a given email_address. The retrieval can be accomplished with the following query with a given email_address:

```
select first_name, last_name from customer where email_address =
'mojo@acme.com';

select first_name, last_name from customer where email_address =
'jojo@acme.com';

select first_name, last_name from customer where email_address =
'coco@acme.com';

...
```

In this case, the WHERE-clause in each query is coded with an actual email address, or a literal value. Then each such query will be considered a distinct query by Oracle, which will incur parsing overhead and affect performance of the application. On the other hand, if this query is coded with the literal value in the WHERE-clause replaced with a variable like

```
select first_name, last_name from customer where email_address =
email_var;
```

then there is only one SQL statement for Oracle to parse, and the execution plan or the associated cursor pointing to the optimal execution plan can be reused, saving repeated parsing overhead and thus improving performance. Consider a large database with millions of customers: the effect could be very substantial.

The next question is how the value of an email address is passed to the variable in the WHERE-clause described above. This is the “bind” part of the concept of a bind variable. The syntax depends on what language is used to code the SQL. In SQL*Plus, a query using a bind variable is coded as follows:

```
SQL>variable email_var varchar2(30)
SQL>exec:email_var:= 'mojo@acme.com'
SQL> select first_name, last_name from customer where email_address
=:email_var;

...
```

Note the colon “:” placed in front of the variable name and the equal sign. Also note that bind variables can be used with other types of SQLs as well, such as INSERT, UPDATE, and so on.

By looking at the source code of an application where a SQL statement is coded, one can figure out whether bind variables are used or not. However, if you do not have access to the source code, how would you know if bind variables are used or not in the application? One way to know it is to examine an AWR report taken with a proper workload applied to the application by following the rules below:

- If you see literal values in a SQL statement, then bind variables are not used.
- If you see items like “:B_n” or “:n” where n is an integer number, then bind variables are used.

Next let’s explain why the setting of the parameter CURSOR_SHARING depends on the use of bind variables.

25.2 ORACLE CURSOR_SHARING PARAMETER

As was described previously, each SQL statement needs to be parsed first before being executed. Each parsed SQL statement has a cursor pointing to its parsed structure in memory. If a SQL statement is coded with bind variables, then the same SQL statement applies to all situations where the only difference is the values passed to the SQL statement. In this case, only one cursor is needed for the parsed SQL. However, if literal values instead of bind variables are used, then each SQL statement with a specific literal value would be treated as a different SQL, and therefore, a new cursor is needed for each SQL statement. This can cause a huge parsing overhead and thus degrade performance. The purpose of the parameter CURSOR_SHARING is to enforce all similar SQLs to be treated like a single, same SQL statement regardless of the actual literal values. The mechanism behind it is to rewrite a SQL statement with system-generated bind variables, often showing up in an AWR report as entries like “:SYS_B_n” where n is an integer. How it is enforced resulted in two different settings for the CURSOR_SHARING parameter: SIMILAR and FORCE. Below is a complete description about all three settings of the parameter CURSOR_SHARING from Oracle’s reference document:

- EXACT: Only allows statements with identical in-line text to share the same cursor.
- SIMILAR: Causes statements that may differ in some in-line literals, but are otherwise identical, to share a cursor, unless the literals affect either the meaning of the statement or the degree to which the plan is optimized.
- FORCE: Similar to the case of SIMILAR as described above except that it ignores the degree to which the plan is optimized.

Whether bind variables are used or not with an application, it’s worthwhile to try SIMILAR and FORCE at least once to see if this parameter matters in terms of

performance. Even if you know or you are told by a reliable source that bind variables are indeed used, it's still worthwhile to try **SIMILAR** or **FORCE**.

As with the preceding case study, let's next try to get to know the architecture of the application involved in this case study.

25.3 GETTING TO KNOW THE APPLICATION ARCHITECTURE

The application architecture for this case study is the same as for the previous case study except that both application server and the Oracle server were on Windows 2003 instead of UNIX. The application server was deployed on a VMWare slice with 8 vCPUs @ 2.4 GHz each and 8 GB RAM, where the Oracle Server was deployed on a physical system with 2 Intel Xeon quad cores with each core @ 1.87 GHz, 16 GB RAM, and an internal RAID 0 for data storage.

The application was a multi-threaded Java program that calls the application server APIs to insert objects into the database. Ten threads were used for this case study. Each thread inserted an independent data structure with one root object and the associated child objects, together with the object relations between the root object and each of its child objects inserted into the database as well.

Next, let's quantify the performance problem with this application.

25.4 QUANTIFYING PROBLEMS

The problem exhibited with the test was that the throughput was lower than expected as shown in the below output of the Java test driver program described previously. It is seen that only an average throughput of 22 objects per second was obtained with 10 threads—about 2.2 objects per second per thread only. Each thread ran with 16 iterations, with 167 component objects and 166 relation objects inserted into the database per iteration per thread.

Here is the output of the Java driver program:

Countdown	# of objects	Throughput (Objects/s)
16	3270	24
15	6540	22
14	9810	21
13	13080	22
12	16350	22
11	19620	22
10	22890	22
9	26160	21
8	29430	22
7	32700	21
6	35970	21
5	39240	22
4	42510	20
3	45780	22

2	49050	20
1	52320	22

Test (10 threads, 52320 instances) completed within 2356453 ms with throughput = 22 Objects/s.

Next, let's analyze the bottlenecks associated with this case study.

25.5 ANALYZING BOTTLENECKS

First, let's check the CPU utilizations on the application server and database server. It turned out that on the application server the average total CPU utilization was 5% only, whereas on the database server, it was about 80% (see Figure 25.1). This huge disparity in CPU usage between the application server and the database server was a clear indicator that the problem was with the database server.

How would we analyze the bottlenecks on the Oracle database server? There is no better place to look at than an AWR report. Next, we will show a few key AWR report sections, rather than pointing out the potential problematic areas immediately. You are encouraged to go over and analyze all the statistics carefully first. You can then compare your analysis with my analysis given after this report.

Note that the purpose of listing so many sections is to help you practice your ability to read a comprehensive AWR report and pinpoint down the problematic areas as quickly and as accurately as possible. If I just list a few metrics that I know are most relevant, it might already be a hint to you what the problem was. In reality, you won't get such hints when a whole, overwhelming AWR report is given to you. Note that

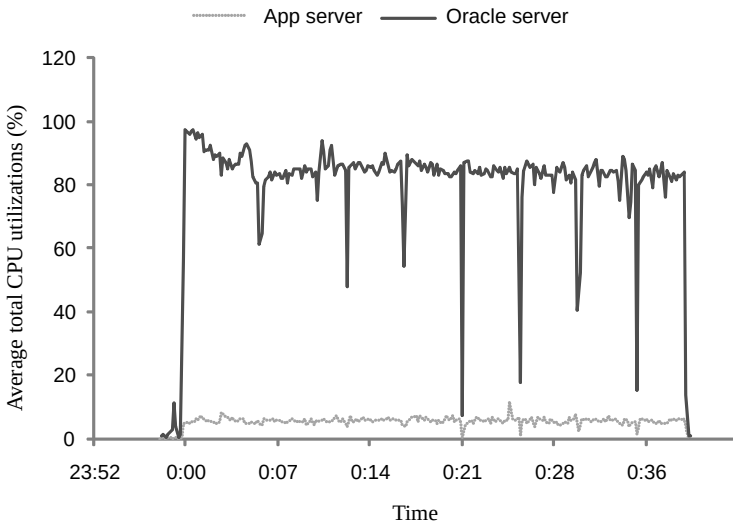


Figure 25.1 Average total CPU utilizations on the app server and Oracle database server recorded during the run with `CURSOR_SHARING` set to `EXACT`.

many parts of the AWR report have been omitted here to make it less overwhelming and also to save some space.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	MI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2237	10-Sep-10 23:59:27	102	3.4		
End Snap:	2238	11-Sep-10 00:40:19	101	3.4		
Elapsed:		40.86 (mins)				
DB Time:		329.98 (mins)				

25.5.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,656M	2,432M	Std Block Size:	8K
Shared Pool Size:	2,160M	2,384M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction	
Redo size:	287,388.87	4,148.62	
Logical reads:	3,117.92	45.01	
Block changes:	1,453.23	20.98	
Physical reads:	0.94	0.01	
Physical writes:	68.01	0.98	
User calls:	595.22	8.59	
Parses:	334.67	4.83	
Hard parses:	191.39	2.76	
Sorts:	35.76	0.52	
Logons:	0.03	0.00	
Executes:	340.82	4.92	
Transactions:	69.27		
% Blocks changed per Read:	46.61	Recursive Call %:	56.76
Rollback per transaction %:	0.01	Rows per Sort:	1.23

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.95	Redo NoWait %:	99.94
Buffer Hit %:	99.97	In-memory Sort %:	100.00
Library Hit %:	77.95	Soft Parse %:	42.81
Execute to Parse %:	1.81	Latch Hit %:	96.90
Parse CPU to Parse Elapsed %:	83.10	% Non-Parse CPU:	-0.19

Shared Pool Statistics

Begin	End	
Memory Usage %:	67.67	76.93
% SQL with executions>1:	95.08	51.19
% Memory for SQL w/exec>1:	93.73	49.72

Top Five Timed Events

ent	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		14,979		75.7	
latch: shared pool	578,324	1,696	3	8.6	Concurrency
latch: library cache	616,819	1,593	3	8.0	Concurrency
log file sync	170,584	528	3	2.7	Commit
log file switch (checkpoint incomplete)	255	149	583	.8	Configuration

Time Model Statistics

- Total time in database user-calls (DB Time) : 19798.6s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
parse time elapsed	18,045.98	91.15
DB CPU	14,978.54	75.65
hard parse elapsed time	1,730.97	8.74
sql execute elapsed time	784.09	3.96
PL/SQL execution elapsed time	4.14	0.02
hard parse (sharing criteria) elapsed time	3.74	0.02
PL/SQL compilation elapsed time	1.64	0.01
connection management call elapsed time	1.17	0.01
repeated bind elapsed time	0.05	0.00
hard parse (bind mismatch) elapsed time	0.03	0.00
sequence load elapsed time	0.00	0.00
DB time	19,798.59	
background elapsed time	226.69	
background cpu time	55.82	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Concurrency	1,209,388	0.00	3,410	3	7.12
Commit	170,584	0.13	528	3	1.00
Configuration	405	45.19	155	382	0.00
System I/O	167,541	0.00	150	1	0.99
Other	33,923	0.61	131	4	0.20
User I/O	2,204	0.00	14	7	0.01
Network	952,801	0.00	2	0	5.61
Application	208	0.00	0	1	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
latch: shared pool	578,324	0.00	1,696	3	3.41
latch: library cache	616,819	0.00	1,593	3	3.63
log file sync	170,584	0.13	528	3	1.00
log file switch (checkpoint incomplete)	255	43.92	149	583	0.00
latch free	31,677	0.00	128	4	0.19

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	197,043
AVG_IDLE_TIME	48,405
AVG_SYS_TIME	4,363
AVG_USER_TIME	192,615
BUSY_TIME	1,576,947
IDLE_TIME	387,781
SYS_TIME	35,483
USER_TIME	1,541,464
RSRC_MGR_CPU_WAIT_TIME	0
VM_IN_BYTES	#####
VM_OUT_BYTES	#####
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
SYS\$USERS	19,779.50	14,966.00	2,125	7,598,383
ORA10GR2	14.60	8.60	32	24,283
ORA10GR2XDB	0.00	0.00	0	0
SYS\$BACKGROUND	0.00	0.00	122	17,922

Service Wait Class Stats

- Wait Class info for services in the Service Statistics section.
- Total Waits and Time Waited displayed for the following wait classes: User I/O, Concurrency, Administrative, Network
- Time Waited (Wt Time) in centisecond (100th of a second)

25.5.2 SQL Statistics

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id
66	50	38	1.74	0.33	6gvch1xu9ca3g
29	26	129,488	0.00	0.15	grwydz59pu6mc
19	16	52,310	0.00	0.10	46q5wjuvnqsk7
11	1	2,608	0.00	0.06	8tq4j4ub4dzx4
5	5	1	5.34	0.03	bc7gjl3ppdtbz
3	2	1,107	0.00	0.02	cb75rw3w1tt0s
2	2	2,560	0.00	0.01	6b1jpl4bdgh8m
2	1	1,232	0.00	0.01	0967kxj496320
2	1	1	2.08	0.01	2nnj66p7y79xb
2	1	41	0.05	0.01	cydnuss99swtd

[Back to SQL Statistics](#)

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 7,643,736
- Captured SQL account for 7.8% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id
258,778	129,488	2.00	3.39	25.60	29.44	grwydz59pu6mc
156,882	52,310	3.00	2.05	15.89	18.91	46q5wjuvnqsk7
38,294	38	1,007.74	0.50	49.68	65.95	6gvch1xu9ca3g
14,840	1	14,840.00	0.19	4.53	5.34	bc7gjl3ppdtbz
12,750	2,624	4.86	0.17	1.35	1.42	4kppbhvhvwt0u
12,598	2,608	4.83	0.16	1.46	11.28	8tq4j4ub4dzx4
12,390	2,560	4.84	0.16	2.03	2.40	6b1jpl4bdgh8m
11,840	2,432	4.87	0.15	1.33	1.51	09px0vjkyd8ym
10,538	2,437	4.32	0.14	0.50	0.77	53saa2zkr6wc3
8,139	796	10.22	0.11	0.65	0.70	7ng34rui5awxq

[Back to SQL Statistics](#)

SQL ordered by Reads

- Total Disk Reads: 2,302
- Captured SQL account for 3.6% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id
255	38	6.71	11.08	49.68	65.95	6gvch1xu9ca3g
63	1	63.00	2.74	4.53	5.34	bc7gv3ppdtbz
23	1,776	0.01	1.00	0.12	0.57	96g93hnrzjtr
9	470	0.02	0.39	0.08	0.17	8swypbbr0m372
6	2	3.00	0.26	0.06	0.12	63fyqfhnd7u5k
6	1	6.00	0.26	0.36	0.66	8avgjb7j4s1nc
5	1	5.00	0.22	1.11	2.08	2nnj66p7y79xb
5	796	0.01	0.22	0.65	0.70	7ng34rui5awxq
5	470	0.01	0.22	0.12	0.22	cqgv56muj63x
5	15	0.33	0.22	0.34	1.33	fdyupq2cqju26

SQL ordered by Parse Calls

- Total Parse Calls: 820,448
- Captured SQL account for 26.5% of Total

Parse Calls	Executions	% Total Parses	SQL Id
129,472	129,488	15.78	grwydz59pu6mc
52,312	52,310	6.38	46q5wjuvnqsk7
2,624	2,624	0.32	4kppbhvhwt0u
2,624	2,624	0.32	5z5rwcpcbkd9
2,608	2,608	0.32	8tq4j4ub4d4x4
2,608	2,608	0.32	adck2kfrwpmdr
2,560	2,560	0.31	6b1jppj4bdgh8m
2,560	2,560	0.31	d5w1tnr3zmbzj
2,432	2,432	0.30	09px0vjkyd8ym
2,432	2,432	0.30	70d2m20rr09kr

25.5.3 IO Stats

Tablespace IO Stats

- ordered by IOs (Reads+Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	1,643	1	5.79	1.09	93,022	38	3,459	10.23
UNDOTBS1	19	0	39.47	1.00	22,205	9	272	1.84
SYSAUX	117	0	24.10	1.29	921	0	0	0.00
SYSTEM	236	0	14.87	1.17	196	0	0	0.00
ATRMWS	17	0	88.82	1.00	16	0	0	0.00
USERS	17	0	86.47	1.00	16	0	0	0.00

25.5.4 Wait Statistics

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	3,303	25	8
1st level bmb	142	10	70
undo header	256	1	2
undo block	16	0	0
2nd level bmb	8	0	0
segment header	5	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
JS-Job Scheduler (queue lock)	8,891	8,891	0	1	1	969.00
CF-Controlfile Transaction	1,332	1,332	0	7	1	80.43
TX-Transaction (index contention)	247	247	0	246	0	0.32
TX-Transaction	181,230	181,054	0	9	0	3.56
HW-Segment High Water Mark	984	984	0	7	0	0.00
FB-Format Block	1,067	1,067	0	3	0	0.00

25.5.5 init.ora Parameters

Parameter Name	Begin value	End value (if different)
compatible	10.2.0.1.0	
cursor_sharing	EXACT	
db_block_size	8192	
db_file_multiblock_read_count	16	
db_name	ORA10GR2	
dispatchers	(PROTOCOL=TCP) (SERVICE=ORA10GR2XDB)	
job_queue_processes	10	
open_cursors	300	
pga_aggregate_target	1707081728	
processes	150	
remote_login_passwordfile	EXCLUSIVE	
sga_target	5133828096	
spfile	D:\ORACLE\PRODUCT\10.2.0\DB_1\SPFILEORA10GR2.ORA	
undo_management	AUTO	
undo_tablespace	UNDOTBS1	
user_dump_dest	D:\ORACLE\PRODUCT\10.2.0\ADMIN\ORA10GR2\UDUMP	

So if you have carefully read all the statistics presented above and come down to this point, which metrics look most suspicious to you? You might already have an answer or have your own way of reading an AWR report, but here is how I analyzed it:

1. First, I would start with the basic formula of *Elapsed time = Service Time (CPU time) + Wait Time*. Oracle is such a good database product that it actually has all bits built in for applying queuing theory to analyzing an Oracle performance problem effectively and efficiently. Let's now plug in all the numbers and see if it works:
 - (a) From the Time Model Statistics, we could get the elapsed time, which is represented as DB time of 19,798.59 seconds.
 - (b) From the same Time Model Statistics, we could get the service time piece in the right-hand-side of the above equation, which is represented as DB CPU of 14,978.54 seconds.
 - (c) Could we get the wait time from the same section of Time Model Statistics? No. The section of Time Model Statistics is either about CPU time or elapsed time mostly. We need to go to the section of Wait Class for wait time. Summing up all the items from the column of Total Wait Time, we would get the total wait time of 4,390 seconds. Adding the service time to the wait time would give us $14,978.54 + 4,390 = 19,368.54$ seconds, which is not exactly equal to the DB time of 19,798.59 seconds but close enough.
2. Now out of the total 19,798.59 seconds of elapsed time, 77% were CPU time and 23% were wait time. This clearly indicates that the bottlenecks were not caused by slow disk or network. Could we make a snap judgment that the bottleneck was with CPU and therefore a more powerful server would be needed for Oracle? This is a kind of very convenient conclusion to jump to, but we would not go that way. We would drill down further into what those CPUs did during that period of time. We need to go back to the section of Time Model Statistics where it shows immediately a large parse time of 18,045.98 seconds at the top and also a significant amount of hard parse time of 1,730.97 seconds. Since an excessive number of hard parses is exhibited as a symptom of poorly reused cursors or SQLs, we could now come to suspect how the `CURSOR_SHARING` parameter was set, which would be a right hit to the root cause of this Oracle performance case study. One can verify from the last section of this AWR report that the `CURSOR_SHARING` parameter was set to EXACT.

Having figured out what the potential bottleneck was, next we could change the `CURSOR_SHARING` parameter to `FORCE` and `SIMILAR`, respectively, and run the same test to see how much improvement could be obtained. The next two sections present the results of the same test with the `CURSOR_SHARING` parameter setting changed to `FORCE` and `SIMILAR` in the two consecutive runs, respectively.

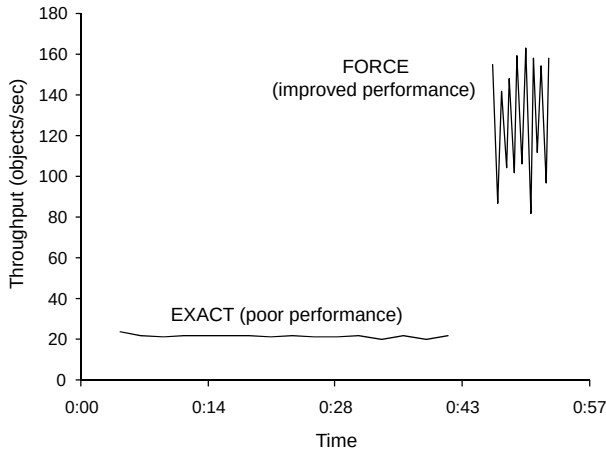


Figure 25.2 Throughput dynamics from the runs with CURSOR_SHARING set to EXACT and FORCE, respectively.

25.6 APPLYING TUNING: CURSOR_SHARING = FORCE

After changing the CURSOR_SHARING parameter from EXACT to FORCE, a throughput of 120 objects per second was obtained immediately. See Figure 25.2 for the throughput dynamics from the runs with the CURSOR_SHARING parameter set to

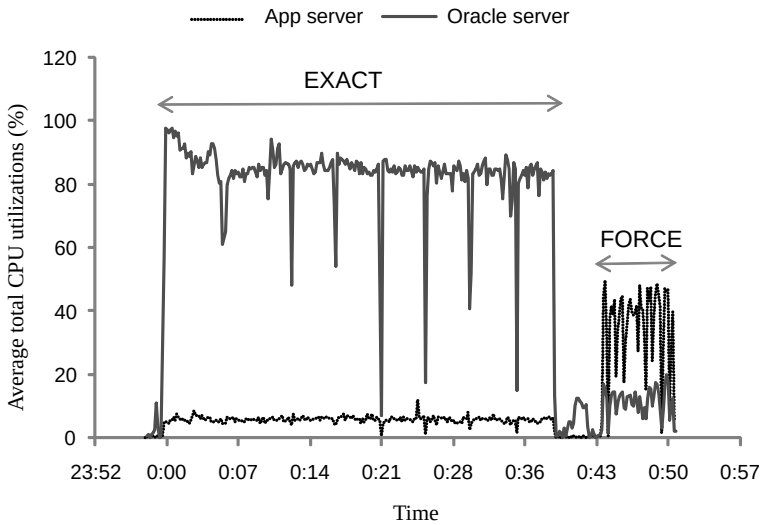


Figure 25.3 Average total CPU utilizations on the app server and Oracle database server recorded during the runs with CURSOR_SHARING set to EXACT and FORCE, respectively.

EXACT and FORCE, respectively. This was a substantial improvement of $120/22 = 5.5$ times, relative to the same test with the CURSOR_SHARING parameter set to EXACT as discussed in the previous section. See also Figure 25.3 for what a huge impact this parameter of CURSOR_SHARING had made on the CPU utilizations on the app server and Oracle server when it was changed from EXACT to FORCE—all for the same amount of computing work done, namely, inserting about 52k objects into the database.

For those who are interested in a detailed comparison between the two sets of statistics (EXACT versus FORCE) obtained while keeping all other test conditions the same, see the below sections for the AWR statistics obtained with the CURSOR_SHARING parameter set to FORCE. The next section provides the results of the same test with the CURSOR_SHARING parameter set to SIMILAR.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	MI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2238	11-Sep-10 00:40:19	101	3.4		
End Snap:	2239	11-Sep-10 00:51:06	102	4.1		
Elapsed:		10.80 (mins)				
DB Time:		18.52 (mins)				

25.6.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,432M	2,432M	Std Block Size:	8K
Shared Pool Size:	2,384M	2,384M	Log Buffer:	14,416K

Load Profile

Per Second	Per Transaction	
Redo size:	1,176,227.80	4,494.42
Logical reads:	19,537.37	74.65
Block changes:	5,822.18	22.25
Physical reads:	9.91	0.04

Physical writes:	303.21	1.16
User calls:	2,246.47	8.58
Parses:	1,102.62	4.21
Hard parses:	2.70	0.01
Sorts:	136.43	0.52
Logons:	0.04	0.00
Executes:	1,313.19	5.02
Transactions:	261.71	
% Blocks changed per Read:	29.80	Recursive Call %: 51.64
Rollback per transaction %:	0.00	Rows per Sort: 106.79

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.62	Redo NoWait %:	99.94
Buffer Hit %:	99.95	In-memory Sort %:	100.00
Library Hit %:	98.94	Soft Parse %:	99.76
Execute to Parse %:	16.03	Latch Hit %:	99.66
Parse CPU to Parse Elapsed %:	97.76	% Non-Parse CPU:	81.85

Shared Pool Statistics

Begin	End	
Memory Usage %:	76.93	74.05
% SQL with executions>1:	51.19	94.33
% Memory for SQL w/exec>1:	49.72	76.91

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
log file sync	171,780	525	3	47.2	Commit
CPU time		349		31.4	
log file switch (checkpoint incomplete)	234	108	461	9.7	Configuration
buffer busy waits	48,299	101	2	9.1	Concurrency
db file parallel write	7,408	79	11	7.1	System I/O

25.6.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 1111.4s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	482.82	43.44
DB CPU	349.21	31.42
parse time elapsed	69.70	6.27
PL/SQL execution elapsed time	13.56	1.22
hard parse elapsed time	11.86	1.07
repeated bind elapsed time	0.42	0.04
PL/SQL compilation elapsed time	0.25	0.02
hard parse (sharing criteria) elapsed time	0.22	0.02
hard parse (bind mismatch) elapsed time	0.06	0.01
connection management call elapsed time	0.02	0.00
sequence load elapsed time	0.01	0.00
failed parse elapsed time	0.00	0.00
DB time	1,111.39	
background elapsed time	219.71	
background cpu time	33.71	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Commit	171,780	0.21	525	3	1.01
System I/O	160,932	0.00	168	1	0.95
Concurrency	68,220	0.07	142	2	0.40
Configuration	560	20.18	127	227	0.00
User I/O	4,553	0.00	17	4	0.03
Other	14,326	47.29	1	0	0.08
Network	949,282	0.00	1	0	5.60
Application	70	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	171,780	0.21	525	3	1.01
log file switch (checkpoint incomplete)	234	40.60	108	461	0.00
buffer busy waits	48,299	0.10	101	2	0.28
db file parallel write	7,408	0.00	79	11	0.04
log file parallel write	150,554	0.00	62	0	0.89

The analysis of the above report is deferred to the next section in conjunction with those obtained with the parameter CURSOR_SHARING set to SIMILAR.

25.7 APPLYING TUNING: CURSOR_SHARING = SIMILAR

After changing the CURSOR_SHARING parameter from FORCE to SIMILAR, a throughput of 121 objects per second was obtained. This essentially was the same as 120 objects per second obtained with the setting of FORCE. For comparison and educational purposes, the following AWR sections are presented.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	M11
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2239	11-Sep-10 00:51:06	102	4.1		
End Snap:	2240	11-Sep-10 01:02:35	101	4.2		
Elapsed:		11.48 (mins)				
DB Time:		20.10 (mins)				

25.7.1 Report Summary

Cache Sizes

Begin	End		
Buffer Cache:	2,432M	2,432M	Std Block Size: 8K
Shared Pool Size:	2,384M	2,384M	Log Buffer: 14,416K

Load Profile

Per Second	Per Transaction			
Redo size:	1,072,121.13			4,355.05
Logical reads:	11,504.81			46.73
Block changes:	5,297.75			21.52
Physical reads:	1.68			0.01
Physical writes:	302.20			1.23
User calls:	2,114.31			8.59
Parses:	986.25			4.01
Hard parses:	0.54			0.00
Sorts:	115.15			0.47
Logons:	0.04			0.00
Executes:	991.36			4.03
Transactions:	246.18			
% Blocks changed per Read:		46.05	Recursive Call %:	13.03
Rollback per transaction %:		0.01	Rows per Sort:	0.28

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.16	Redo NoWait %:	99.94
Buffer Hit %:	100.01	In-memory Sort %:	100.00
Library Hit %:	99.90	Soft Parse %:	99.95
Execute to Parse %:	0.52	Latch Hit %:	99.44
Parse CPU to Parse Elapsd %:	83.32	% Non-Parse CPU:	81.75

Shared Pool Statistics

Begin	End	
Memory Usage %:	74.05	74.68
% SQL with executions>1:	94.33	98.93
% Memory for SQL w/exec>1:	76.91	98.11

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
log file sync	172,453	606	4	50.3	Commit
CPU time		292		24.2	
buffer busy waits	66,436	163	2	13.5	Concurrency
log file switch (checkpoint incomplete)	296	115	388	9.5	Configuration
db file parallel write	8,846	85	10	7.1	System I/O

25.7.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 1206s
- Statistics including the word "background" measure backgroundprocess time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	491.17	40.73
DB CPU	291.54	24.17
parse time elapsed	72.94	6.05
hard parse elapsed time	17.34	1.44
PL/SQL execution elapsed time	3.39	0.28
hard parse (sharing criteria) elapsed time	0.23	0.02
hard parse (bind mismatch) elapsed time	0.17	0.01
PL/SQL compilation elapsed time	0.04	0.00
connection management call elapsed time	0.03	0.00
repeated bind elapsed time	0.00	0.00
sequence load elapsed time	0.00	0.00
DB time	1,205.99	
background elapsed time	230.27	
background cpu time	33.69	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Commit	172,453	0.26	606	4	1.02
Concurrency	93,257	0.10	215	2	0.55
System I/O	154,705	0.00	180	1	0.91
Configuration	681	16.45	131	193	0.00
User I/O	1,400	0.00	12	9	0.01
Other	18,964	56.94	1	0	0.11
Network	950,505	0.00	1	0	5.61
Application	88	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	172,453	0.26	606	4	1.02
buffer busy waits	66,436	0.14	163	2	0.39
log file switch (checkpoint incomplete)	296	33.45	115	388	0.00
db file parallel write	8,846	0.00	85	10	0.05
log file parallel write	143,450	0.00	63	0	0.85

Now let's compare how various statistics had changed out of those three runs with the parameter CURSOR_SHARING set to EXACT, FORCE, and SIMILAR, respectively.

- Total elapsed time: 40.86 minutes (2452 seconds) with EXACT versus 10.86 minutes (652 seconds) and 11.48 minutes (689 seconds) with FORCE and SIMILAR, respectively.
- Load profile:
 - Redo size: 3.16 GB with EXACT versus 810 MB and 739 MB with FORCE and SIMILAR, respectively.
 - Logical reads: 7.6 million with EXACT versus 12.7 million and 7.9 million with FORCE and SIMILAR, respectively.
 - Parses: 820,611 with EXACT versus 718,895 and 679,354 with FORCE and SIMILAR, respectively.
 - Hard parses: 469,288 with EXACT versus 1,760 and 352 with FORCE and SIMILAR, respectively.

- Executes: 835,691 with EXACT versus 856,076 and 682,799 with FORCE and SIMILAR, respectively.
- Transactions: 603,633 with EXACT versus 170,628 and 169,494 with FORCE and SIMILAR, respectively.
- Instance Efficiency:
 - Execute to Parse %: 0.52 with EXACT versus 16.03 and 0.52 with FORCE and SIMILAR, respectively.
 - Parse CPU to Parse Elapsed %: 83.10 with EXACT versus 16.03 and 83.32 with FORCE and SIMILAR, respectively.
 - Soft Parse %: 42.81 with EXACT versus 94.33 and 99.95 with FORCE and SIMILAR, respectively.
 - % Non-Parse CPU: -0.19 with EXACT versus 81.85 and 81.75 with FORCE and SIMILAR, respectively.
- Top Five Timed Events:
 - CPU time: 14,979 seconds or 75.7 % total call time with EXACT versus 349 seconds or 31.4 % total call time and 292 seconds or 24.2 % total call time with FORCE and SIMILAR, respectively.
 - Latch—shared pool: 1,696 seconds with EXACT versus negligible with FORCE and SIMILAR, respectively.
 - Latch—library cache: 1,593 seconds with EXACT versus negligible with FORCE and SIMILAR, respectively.
 - Log file sync: 170,584 waits or 528 seconds with EXACT versus 171,780 waits or 525 seconds and 172,453 waits or 606 seconds with FORCE and SIMILAR, respectively.
 - Log file switch: 255 waits or 149 seconds with EXACT versus 234 waits or 108 seconds and 296 waits or 115 seconds with EXACT versus 234 waits or 108 seconds and 296 waits or 115 seconds with FORCE and SIMILAR, respectively.
- Time Model Statistics:
 - Parse time elapsed: 18,045.98 seconds or 91.15% of DB time with EXACT versus 482.82 seconds or 43.44 % of DB Time and 491.17 seconds or 40.73 % of DB Time with FORCE and SIMILAR, respectively.
 - Hard parse elapsed time: 1,730.97 seconds with EXACT or 11.86 seconds and 17.34 seconds with FORCE and SIMILAR, respectively.
- Wait Class:
 - Concurrency: 1,209,388 waits or 3,410 seconds with EXACT versus 68,220 waits or 142 second total wait time and 93,257 waits or 215 second total wait time with FORCE and SIMILAR, respectively.
 - Commit: 170,584 waits or 528 seconds with EXACT versus 171,780 waits or 525 second total wait time and 172,453 waits or 606 second total wait time with FORCE and SIMILAR, respectively.

- Operating System Statistics: AVG_BUSY_TIME: 1,970 seconds with EXACT versus 62 seconds and 60 seconds with FORCE and SIMILAR, respectively.
- SQL Statistics: no particular hot SQLs in all three cases.

Based on the above comparisons, it's clear that parsing was the major activity that had been improved significantly by changing the CURSOR_SHARING parameter from the default setting of EXACT to FORCE or SIMILAR.

Between FORCE and SIMILAR, which one should be chosen over the other should be determined with rigorous tests with your product. In general, they don't differ much, but only your tests with your product can tell.

25.8 MORAL OF THE CASE STUDY

CURSOR_SHARING is one of the most important parameters that can affect the performance and scalability of an Oracle-based enterprise application. Even if the application uses bind variables, still it's worthwhile to try out the settings of FORCE or SIMILAR for CURSOR_SHARING with adequate tests in order to exploit the potential performance and scalability benefits. If an application supports multiple database platforms and bind variables are not used with Oracle specific implementation, then this parameter needs to be set to either SIMILAR or FORCE in order to achieve the full performance and scalability potential with your product.

I have to mention that this case study was presented with Oracle 10g R2 instead of Oracle 11g R2. Oracle 11g R2 has a feature called adaptive cursor sharing built in. However, as was pointed out in Section 12.4.2, this feature applies to Oracle-based applications that use bind variables only. The product used for the case study presented in this chapter does not use bind variables, since it has to support a variety of database platforms. To verify the effects of various CURSOR_SHARING settings on the performance and scalability of this product on Oracle 11g R2, I ran a series of tests using comparable test conditions as with 10g R2 and obtained throughput numbers of 139, 200, and 196 objects/second with EXACT, SIMILAR, and FORCE, respectively. This helps confirm that the effects of various CURSOR_SHARING settings on the performance and scalability of this Oracle-based application remain largely the same from Oracle 10g to 11g.

RECOMMENDED READING

For a more in-depth discussion on CURSOR_SHARING, refer to the following Oracle document:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 25.1** Which statistics in an AWR report are most indicative of whether an Oracle performance bottleneck is with hardware (either raw power or configuration), software configuration, or application design and implementation, or a combination of multiple factors?
- 25.2** Explain the concept of a bind variable. If viable, construct an Oracle-based application with two options: one uses bind variables and other doesn't. Then design a test and compare the effects of the different CURSOR_SHARING settings (EXACT, FORCE, and SIMILAR) on the performance of the application.
- 25.3** What's the major difference between a hard parse and a soft parse? Which metrics in an AWR report are most indicative of efficient or inefficient parsing with an Oracle-based application?
- 25.4** Assume that you have to use FORCE or SIMILAR for CURSOR_SHARING with your Oracle-based application. What criteria do you use to determine whether FORCE or SIMILAR should be used for your application?
- 25.5** During testing the performance and scalability of a real product on Oracle 11g R2, the following SQL query was identified using the Top Activity feature on the EM DBConsole as the top SQL, which was driving the database server CPUs up to over 90% busy while driving down the throughput of the batch job under test rapidly:

```
SELECT T419.C1, C2, C3, C4, C5 FROM T419
WHERE ((T419.C3 = : "SYS_B_0") AND (T419.C6 = : "SYS_B_1")
AND ((T419.C7 IS NULL) OR (T419.C7 = : "SYS_B_2")))
ORDER BY : "SYS_B_3" ASC
```

After checking all existing indexes for this table, it was found that apparently no index was created for this query. Given the fact that column C7 can take only one of the three values: NULL, 0 (“No”), and 1 (“Yes”), what index in terms of the columns and the sequence of the columns do you suggest in order to cure the issue? (Note that this was the same exercise I went through when testing the product on Oracle 11g R2 as mentioned at the end of this chapter).

26

Case Study: Bulk Transactions

Better a diamond with a flaw than a pebble without.
—Confucius

The business logic of an enterprise application may contain correlated transactions. For example, an object may contain dependent objects, and the dependent objects may need to be associated with their parent object, resulting in relationship objects that have source and target objects reflected therein. Figure 26.1 shows such an example where a COMPUTER object has two dependent objects: PROCESSOR and OPERATINGSYSTEM. There are also two relationship objects describing the relations: (1) between a COMPUTER and a PROCESSOR; and (2) between a COMPUTER and an OPERATINGSYSTEM. Then we have two options on how these objects can be inserted into a database:

1. Insert each object and relation individually into the database and each transaction is committed individually. This is known as *non-bulk transaction*.
2. Insert all five objects into the database in one bulk or batch transaction, namely, with all five objects committed only once. This is known as *bulk-transaction*.

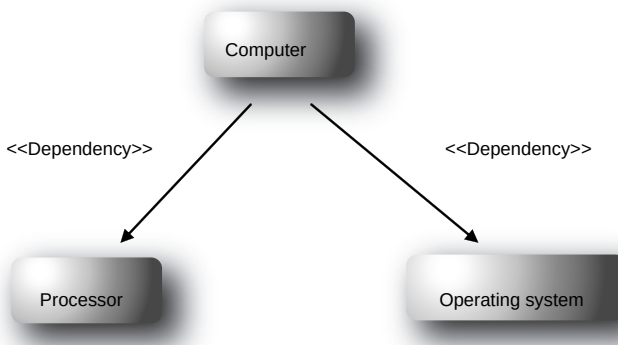


Figure 26.1 An example structural data model showing dependencies among objects.

The interest here is how much one would gain in terms of performance and scalability with bulk versus non-bulk transaction. This case study answers this question quantitatively with a real product.

This chapter consists of the following sections:

- Application Architecture
- Quantifying the Problems
- Identifying Performance and Scalability Optimization Opportunities
- Effects of Bulk Transactions on Performance
- Moral of the Case Study

Let's start with a description of the application architecture next.

26.1 APPLICATION ARCHITECTURE

The same application used for the preceding case study was used for this case study. The difference is that the tests were run with different configuration parameters. If you skipped the preceding case study, refer to Section 25.3 to get familiar with the application architecture first before proceeding to the next section.

26.2 QUANTIFYING PROBLEMS

With this case study, the problem was not that the application did not perform or scale. Rather, we looked for opportunities on how the product could be made even better in terms of performance and scalability to meet ever-increasing demands.

To make such an effort successful, rigorous performance and scalability tests were required to help the product development team to evaluate their design and implementations.

The product was tested first with non-bulk transaction. With a data model of 164 objects + 163 relations, a total number of $164 + 163 = 327$ objects were inserted into the database per iteration. The test driver program looped through that data model with 16 iterations and 10 threads, so at the end, a total number of $(164 + 163) * 16 * 10 = 52,320$ objects (or roughly, 52k) were inserted into the database. The insertion of each object was committed as an independent transaction. A high average throughput of 138 objects per second was achieved.

Next, let’s take a look at the AWR report taken with the above test and see what opportunities we might have to improve the performance and scalability of this product.

26.3 IDENTIFYING PERFORMANCE AND SCALABILITY OPTIMIZATION OPPORTUNITIES

To identify the performance and scalability optimization opportunities for the product under test, a portion of the AWR report taken with the non-bulk transaction mode is presented in this section. The CURSOR_SHARING parameter was set to SIMILAR. To help you sharpen your ability to read an AWR report, go over the statistics below first and then perform your analysis or refer to the analysis following this AWR report.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	XI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2249	11-Sep-10 09:56:31	102	3.4		
End Snap:	2250	11-Sep-10 10:04:27	103	3.7		
Elapsed:		7.94 (mins)				
DB Time:		12.85 (mins)				

26.3.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,544M	2,656M	Std Block Size:	8K
Shared Pool Size:	2,272M	2,160M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction	
Redo size:	1,572,966.84	4,421.62	
Logical reads:	14,137.97	39.74	
Block changes:	7,506.84	21.10	
Physical reads:	13.58	0.04	
Physical writes:	295.26	0.83	
User calls:	3,057.27	8.59	
Parses:	1,423.10	4.00	
Hard parses:	0.53	0.00	
Sorts:	165.54	0.47	
Logons:	0.04	0.00	
Executes:	1,427.01	4.01	
Transactions:	355.74		
% Blocks changed per Read:	53.10	Recursive Call %:	12.09
Rollback per transaction %:	0.00	Rows per Sort:	0.19

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.50	Redo NoWait %:	99.94
Buffer Hit %:	99.91	In-memory Sort %:	100.00
Library Hit %:	99.95	Soft Parse %:	99.96
Execute to Parse %:	0.27	Latch Hit %:	99.64
Parse CPU to Parse Elapsed %:	94.54	% Non-Parse CPU:	80.39

Shared Pool Statistics

	Begin	End
Memory Usage %:	67.54	70.28
% SQL with executions>1:	7.10	98.86
% Memory for SQL w/exec>1:	29.70	96.72

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		270		35.1	
log file sync	171,204	265	2	34.4	Commit
log file switch (checkpoint incomplete)	315	124	394	16.1	Configuration
buffer busy waits	33,890	117	3	15.2	Concurrency
db file parallel write	2,654	35	13	4.5	System I/O

26.3.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 770.7s
- Statistics including the word “background” measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	393.42	51.05
DB CPU	270.20	35.06
parse time elapsed	57.13	7.41
hard parse elapsed time	0.67	0.09
PL/SQL execution elapsed time	0.28	0.04
hard parse (sharing criteria) elapsed time	0.17	0.02
hard parse (bind mismatch) elapsed time	0.03	0.00
connection management call elapsed time	0.02	0.00
repeated bind elapsed time	0.00	0.00
PL/SQL compilation elapsed time	0.00	0.00
DB time	770.72	
background elapsed time	120.75	
background cpu time	33.81	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Commit	171,204	0.09	265	2	1.01
Concurrency	46,354	0.16	149	3	0.27
Configuration	610	19.18	137	225	0.00
System I/O	151,754	0.00	76	1	0.90
User I/O	1,249	0.00	6	5	0.01
Network	949,859	0.00	1	0	5.60
Other	12,323	41.41	1	0	0.07
Application	44	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /bn
log file sync	171,204	0.09	265	2	1.01
log file switch (checkpoint incomplete)	315	33.97	124	394	0.00
buffer busy waits	33,890	0.22	117	3	0.20
db file parallel write	2,654	0.00	35	13	0.02
log file parallel write	146,842	0.00	34	0	0.87

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	5,454
AVG_IDLE_TIME	42,154
AVG_SYS_TIME	#####
AVG_USER_TIME	3,308
BUSY_TIME	43,749
IDLE_TIME	337,362
SYS_TIME	17,175
USER_TIME	26,574
RSRC_MGR_CPU_WAIT_TIME	0
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
SYSSUSERS	765.30	269.90	6,376	6,727,198
ORA10GR2	5.40	0.30	0	4,883
ORA10GR2XDB	0.00	0.00	0	0
SYSSBACKGROUND	0.00	0.00	98	3,431

Service Wait Class Stats

- Wait Class info for services in the Service Statistics section.
- Total Waits and Time Waited displayed for the following wait classes: User I/O, Concurrency, Administrative, Network
- Time Waited (Wt Time) in centisecond (100th of a second)

Service Name	User I/O Total Wts	User I/O Wt Time	Concurcy Total Wts	Concurcy Wt Time	Admin Total Wts	Admin Wt Time	Network Total Wts	Network Wt Time
SYSSUSERS	1055	605	46346	14875	0	0	949229	115
ORA10GR2	0	0	0	0	0	0	618	0
SYSSBACKGROUND	194	22	8	7	0	0	0	0

26.3.3 SQL Statistics

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
106	13	25,598	0.00	13.78	<u>dvsmg0345ng4m</u>	app.exe	INSERT INTO T177 (C2, C7, C8, ...
65	25	26,080	0.00	8.49	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
49	16	24,319	0.00	6.30	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
40	12	25,759	0.00	5.21	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300="S...

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
25	65	26,080	0.00	8.49	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
16	49	24,319	0.00	6.30	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
13	106	25,598	0.00	13.78	<u>dvsmg0345ng4m</u>	app.exe	INSERT INTO T177 (C2, C7, C8, ...
12	40	25,759	0.00	5.21	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300="S...
11	22	24,319	0.00	2.81	<u>5nbpkagga7a0h</u>	app.exe	INSERT INTO T226 (C2, C7, C8, ...

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 6,735,893
- Captured SQL account for 94.2% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1,492,559	26,080	57.23	22.16	25.16	65.41	bq7rd3yrf2827	app.exe	INSERT INTO T179 (C2, C7, C8, ...
652,415	24,319	26.83	9.69	16.10	48.55	gr95kumnq95nj	app.exe	INSERT INTO T236 (C2, C7, C8, ...
442,474	25,759	17.18	6.57	11.50	40.16	d6cn7x5924z69	app.exe	UPDATE T236 SET C400131300="S...
376,011	25,598	14.69	5.58	12.66	106.24	dvsmg0345ng4m	app.exe	INSERT INTO T177 (C2, C7, C8, ...
360,237	24,319	14.81	5.35	10.96	21.67	5nbpkagga7a0h	app.exe	INSERT INTO T226 (C2, C7, C8, ...

SQL ordered by Reads

- Total Disk Reads: 6,471
- Captured SQL account for 93.0% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
3,199	26,080	0.12	49.44	25.16	65.41	bq7rd3yrf2827	app.exe	INSERT INTO T179 (C2, C7, C8, ...
954	24,319	0.04	14.74	16.10	48.55	gr95kumnq95nj	app.exe	INSERT INTO T236 (C2, C7, C8, ...
487	25,728	0.02	7.53	5.25	5.42	f8xs5864bzb26	app.exe	SELECT * FROM (SELECT T179.C...
340	25,759	0.01	5.25	11.50	40.16	d6cn7x5924z69	app.exe	UPDATE T236 SET C400131300="S...
337	25,598	0.01	5.21	12.66	106.24	dvsmg0345ng4m	app.exe	INSERT INTO T177 (C2, C7, C8, ...

SQL ordered by Parse Calls

- Total Parse Calls: 678,024
- Captured SQL account for 95.8% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
52,318	52,266	7.72	46q5wjuvngsk7	app.exe	SELECT T264.C1, C20051, C20052...
52,155	52,120	7.69	c6801bm8xj79j	app.exe	SELECT T236.C1, C400079600, C4...
51,518	51,507	7.60	4jd6fdxb15ugs	app.exe	SELECT C179, C400079600 FROM T...
48,638	48,621	7.17	f7r3zwcq8th0j	app.exe	SELECT C400079600 FROM T273 WH...
26,239	26,240	3.87	0k3ta9hv7s6hb	app.exe	INSERT INTO H236 (entryld, T0,...

SQL ordered by Sharable Memory

- Only Statements with Sharable Memory greater than 1,048,576 are displayed

Sharable Mem (b)	Executions	% Total	SQL Id	SQL Module	SQL Text
6,363,792		0.28	1pxqzugfbcrdk		** SQL Text Not Available **
5,976,144		0.26	fchdfxrdqk16n		** SQL Text Not Available **
5,142,984		0.23	0gsavp9apga5n		** SQL Text Not Available **
4,590,432		0.20	44v0t18cz509t		** SQL Text Not Available **

SQL ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
297		0gsavp9apga5n		** SQL Text Not Available **
297		1pxqzugfbcrdk		** SQL Text Not Available **
279		fchdfxrdqk16n		** SQL Text Not Available **
264		44v0t18cz509t		** SQL Text Not Available **

26.3.4 Wait Statistics

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	31,112	106	3
undo header	1,339	10	8
1st level bmb	1,385	0	0
segment header	30	0	0
2nd level bmb	21	0	0
undo block	18	0	0
file header block	1	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	2,123	2,123	0	2,122	31	14.68
CF-Controlfile Transaction	431	431	0	8	0	3.88
FB-Format Block	1,390	1,389	0	76	0	0.21
HW-Segment High Water Mark	1,327	1,327	0	117	0	0.00
TX-Transaction	181,487	181,189	0	28	0	0.00
TX-Transaction (allocate ITL entry)	28	28	0	28	0	0.00

First, it is seen from the header section that the total elapsed time for this AWR report was 7.94 minutes or 476 seconds. Using this total duration of 476 seconds, some of the metric values given in terms of “per second” can be converted to “total” so that the comparison will make more sense.

In summary, the following metric values are worthwhile to consider for identifying how this application can be improved for better performance and scalability:

- Load Profile:
 - Redo size: 749 MB total or 4.4 kB per transaction
 - Logical reads: 6,729,688 total or 39.74 per transaction
 - Block changes: 140,544 or 21.10 per transaction
 - User calls: 1,455,260 total or 8.59 per transaction
 - Parses: 677,396 total or 4 per transaction
 - Executes: 679,572 total or 4 per transaction
 - Transactions: 169,332 total
 - % Blocks changed per read: 53.10
- Top Five Timed Events:
 - CPU time: 270 seconds or 35.1 % total call time
 - Log file sync: 265 seconds or 34.4 % total call time (note that this is high and alarming)
- Time Model Statistics:
 - Sql execute elapsed time: 393.42 seconds or 51.05 % of DB time
- Wait Class:
 - Commit: 171,294 waits or 265 second total wait time
 - Concurrency: 46,354 waits or 149 second total wait time
 - Configuration: 610 waits or 137 second total wait time
- Wait Events:
 - Log file sync: 171,204 waits or 265 second total wait time
 - Log file switch (checkpoint incomplete): 315 waits for 124 second total wait time
 - Buffer busy waits: 33,890 waits or 117 second total wait time

- Operating System Statistics: AVG_BUSY_TIME: 54.54 seconds.
- SQL Statistics: SQL ordered by Elapsed Time: the top four SQLs were three INSERTs and one UPDATE. The highest % Total DB Time was 13.78%.
- Buffer Wait Statistics: data block: 31,112 waits or 106 second Total Wait Time.
- Enqueue Activity: TX-Transaction (index contention): 2,122 waits or 31 second wait time.

Next, we will see how these metric values had changed when the same test was run in bulk transaction mode.

26.4 EFFECTS OF BULK TRANSACTIONS ON PERFORMANCE

After the same test was run with the test driver configured to use bulk transaction instead of non-bulk transaction APIs obtained while keeping all other test conditions the same, a higher average throughput of 197 objects per second was obtained. Compared with the average throughput of 138 objects per second obtained with non-bulk transactions, this throughput represents an impressive improvement of $197/138 = 1.43$ times. The AWR report associated with this significant throughput improvement obtained with bulk transactions is given below.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	XI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2250	11-Sep-10 10:04:27	103	3.7		
End Snap:	2251	11-Sep-10 10:09:55	103	3.7		
Elapsed:		5.46 (mins)				
DB Time:		9.00 (mins)				

26.4.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,656M	2,656M	Std Block Size:	8K
Shared Pool Size:	2,160M	2,160M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction
Redo size:	1,887,778.64	46,626.11
Logical reads:	177,640.55	4,387.53
Block changes:	11,175.87	276.03
Physical reads:	0.31	0.01
Physical writes:	405.40	10.01
User calls:	2,303.03	56.88
Parses:	1,907.75	47.12
Hard parses:	36.66	0.91
Sorts:	80.17	1.98
Logons:	0.03	0.00
Executes:	1,909.69	47.17
Transactions:	40.49	

% Blocks changed per Read:	6.29	Recursive Call %:	4.83
Rollback per transaction %:	0.00	Rows per Sort:	0.44

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.98	Redo NoWait %:	99.98
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	97.61	Soft Parse %:	98.08
Execute to Parse %:	0.10	Latch Hit %:	98.55
Parse CPU to Parse Elapsed %:	86.23	% Non-Parse CPU:	76.71

Shared Pool Statistics

	Begin	End
Memory Usage %:	70.28	75.72
% SQL with executions>1:	98.86	99.37
% Memory for SQL w/exec>1:	96.72	96.17

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		453		83.8	
log file switch (checkpoint incomplete)	73	39	529	7.2	Configuration
db file parallel write	4,161	30	7	5.6	System I/O
log file sync	13,214	19	1	3.6	Commit
log file switch completion	178	12	68	2.2	Configuration

26.4.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 540s
- Statistics including the word "background" measure background process time, and so do not contribute to the DB time statistic
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
DB CPU	452.74	83.84
sql execute elapsed time	366.31	67.83
parse time elapsed	125.34	23.21
hard parse elapsed time	18.18	3.37
PL/SQL execution elapsed time	0.19	0.03
hard parse (sharing criteria) elapsed time	0.09	0.02
hard parse (bind mismatch) elapsed time	0.01	0.00
connection management call elapsed time	0.00	0.00
PL/SQL compilation elapsed time	0.00	0.00
sequence load elapsed time	0.00	0.00
repeated bind elapsed time	0.00	0.00
DB time	540.01	
background elapsed time	62.47	
background cpu time	10.41	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Configuration	306	13.73	51	166	0.02
System I/O	19,310	0.00	49	3	1.46
Concurrency	68,609	0.00	27	0	5.18
Commit	13,214	0.05	19	1	1.00
Network	740,720	0.00	1	0	55.88
Other	2,371	32.64	1	0	0.18
User I/O	181	0.00	0	3	0.01
Application	22	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file switch (checkpoint incomplete)	73	45.21	39	529	0.01
db file parallel write	4,161	0.00	30	7	0.31
log file sync	13,214	0.05	19	1	1.00
log file switch completion	178	5.06	12	68	0.01
log file parallel write	13,605	0.00	12	1	1.03

Operating System Statistics

Statistic	Total
AVG_IDLE_TIME	25,550
AVG_SYS_TIME	#####
AVG_USER_TIME	5,702
BUSY_TIME	57,398
IDLE_TIME	204,492
SYS_TIME	11,714
USER_TIME	45,684
RSRC_MGR_CPU_WAIT_TIME	0
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
SYSS\$USERS	536.90	452.60	22	58,159,958
ORA10GR2	3.10	0.10	0	1,045
ORA10GR2XDB	0.00	0.00	0	0
SYSS\$BACKGROUND	0.00	0.00	78	516

Service Wait Class Stats

- Wait Class info for services in the Service Statistics section.
- Total Waits and Time Waited displayed for the following wait classes: User I/O, Concurrency, Administrative, Network
- Time Waited (Wt Time) in centisecond (100th of a second)

Service Name	User I/O	User	Concurcy	Concurcy	Admin	Admin	Network	Network
	Total Wts	I/O Wt Time	Total Wts	Wt Time	Total Wts	Wt Time	Total Wts	Wt Time
SYSS\$USERS	25	12	68600	2690	0	0	740472	88
ORA10GR2	0	0	2	0	0	0	245	0
SYSS\$BACKGROUND	156	36	7	6	0	0	0	0

26.4.3 SQL Statistics

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
52	52	48,600	0.00	9.67	<u>f7r3zwcq8th0j</u>	app.exe	SELECT C400079600 FROM T273 WH...
32	31	25,760	0.00	5.83	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300=:"S...
30	28	24,638	0.00	5.59	<u>gnah6jc7q5ycd</u>	app.exe	SELECT C179, C400127400, C4000...
29	23	24,317	0.00	5.28	<u>ds3hscqndsvdd</u>	app.exe	UPDATE T226 SET C301047800=:"S...
27	27	23,359	0.00	5.04	<u>79rty6t37r6b1</u>	app.exe	SELECT C179, C400127400, C4000...

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
52	52	48,600	0.00	9.67	<u>f7r3zwcq8th0j</u>	app.exe	SELECT C400079600 FROM T273 WH...
31	32	25,760	0.00	5.83	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300=:"S...
28	30	24,638	0.00	5.59	<u>gnah6jc7q5ycd</u>	app.exe	SELECT C179, C400127400, C4000...
27	27	23,359	0.00	5.04	<u>79rty6t37r6b1</u>	app.exe	SELECT C179, C400127400, C4000...
27	27	27,637	0.00	4.97	<u>4jd6fdxb15ugs</u>	app.exe	SELECT C179, C400079600 FROM T...

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 58,161,115
- Captured SQL account for 88.0% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
12,442,704	48,600	256.02	21.39	52.10	52.24	f7r3zwcq8th0j	app.exe	SELECT C400079600 FROM T273 WH...
6,797,104	27,637	245.94	11.69	26.79	26.84	4jd6fdxb15ugs	app.exe	SELECT C179, C400079600 FROM T...
6,531,099	25,760	253.54	11.23	31.33	31.51	d6cn7x5924z69	app.exe	UPDATE T236 SET C400131300="S...
6,429,582	24,638	260.96	11.05	27.89	30.18	gnah6jc7q5ycd	app.exe	SELECT C179, C400127400, C4000...
6,116,021	23,359	261.83	10.52	27.09	27.21	79rty6t37r6b1	app.exe	SELECT C179, C400127400, C4000...

SQL ordered by Reads

- Total Disk Reads: 100
- Captured SQL account for 15.0% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id
8	26,079	0.00	8.00	20.14	26.86	bq7rd3yrf2827
7	1	7.00	7.00	0.92	1.01	bc7gv3ppdtbz
7	25,760	0.00	7.00	31.33	31.51	d6cn7x5924z69

SQL ordered by Executions

- Total Executions: 625,250
- Captured SQL account for 92.3% of Total

Executions	Rows Processed	Rows per Exec	CPU per Exec (s)	Elap per Exec (s)	SQL Id	SQL Module	SQL Text
52,152	52,135	1.00	0.00	0.00	c6801bm8xj79j	app.exe	SELECT T236.C1, C400079600, C4...
48,600	48,597	1.00	0.00	0.00	f7r3zwcq8th0j	app.exe	SELECT C400079600 FROM T273 WH...
27,637	27,636	1.00	0.00	0.00	4jd6fdxb15ugs	app.exe	SELECT C179, C400079600 FROM T...
26,240	26,238	1.00	0.00	0.00	0k3ta9hv7s6hb	app.exe	INSERT INTO H236 (entryId, T0,...
26,080	26,077	1.00	0.00	0.00	g9ttytv00vsax	app.exe	INSERT INTO H179 (entryId, T0,...
26,079	26,077	1.00	0.00	0.00	bq7rd3yrf2827	app.exe	INSERT INTO T179 (C2, C7, C8, ...
25,760	25,760	1.00	0.00	0.00	d6cn7x5924z69	app.exe	UPDATE T236 SET C400131300="S...

SQL ordered by Parse Calls

- Total Parse Calls: 624,613
- Captured SQL account for 92.4% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
52,160	52,152	8.35	c6801bm8xj79j	app.exe	SELECT T236.C1, C400079600, C4...
48,608	48,600	7.78	f7r3zwcq8th0j	app.exe	SELECT C400079600 FROM T273 WH...
27,638	27,637	4.42	4jd6fdxb15ugs	app.exe	SELECT C179, C400079600 FROM T...
26,240	26,240	4.20	0k3ta9hv7s6hb	app.exe	INSERT INTO H236 (entryId, T0,...
26,080	26,080	4.18	g9ttytv00vsax	app.exe	INSERT INTO H179 (entryId, T0,...

SQL ordered by Sharable Memory

- Only Statements with Sharable Memory greater than 1,048,576 are displayed

Sharable Mem (b)	Executions	% Total	SQL Id	SQL Module	SQL Text
6,363,792		0.28	1pxqzugfbcrdk		** SQL Text Not Available **
5,976,144		0.26	fchdfxrdqk16n		** SQL Text Not Available **
5,142,984		0.23	0gsavp9apga5n		** SQL Text Not Available **
4,590,432		0.20	44v0t18cz509t		** SQL Text Not Available **

SQL ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
297		0gsavp9apga5n		** SQL Text Not Available **
297		1pxqzugfbcrdk		** SQL Text Not Available **
279		fchdfxrdqk16n		** SQL Text Not Available **
264		44v0t18cz509t		** SQL Text Not Available **

26.4.4 Wait Statistics**Buffer Wait Statistics**

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	8,758	3	0
1st level bmb	493	0	0
undo header	639	0	0
undo block	141	0	0
segment header	14	0	0
2nd level bmb	13	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	603	603	0	602	6	10.25
FB-Format Block	872	872	0	41	0	8.76
TX-Transaction	20,563	20,563	0	19	0	3.16
CF-Controlfile Transaction	240	240	0	4	0	4.00
HW-Segment High Water Mark	794	794	0	40	0	0.38
TX-Transaction (allocate ITL entry)	1	1	0	1	0	0.00

First, it is seen from the header section that the total elapsed time for this AWR report was 5.46 minutes or 328 seconds versus 7.94 minutes or 476 seconds for the test run with non-bulk transactions. The other metric values are compared systematically with those from non-bulk transactions as follows:

- Load Profile:
 - Redo size: 619 MB or 46.6 kB per transaction versus 749 MB total or 4.4 kB per transaction for the previous run. Note that the size of a transaction was increased by about a factor of 10 from non-bulk to bulk transactions.
 - Logical reads: 58,2625,920 or 4,388 per transaction versus 6,729,688 total or 39.74 per transaction with non-bulk transactions. This was about a factor of 110 increase.
 - Block changes: 3,665,685 or 276 per transaction versus 140,544 or 21.10 per transaction. This metric was increased by 13 times.
 - User calls: 755,384 or 56.88 per transaction versus 1,455,260 total or 8.59 per transaction with non-bulk transactions. Note that the total number of user calls was reduced by as much as about 2 times.
 - Parses: 625,496 or 47 per transaction versus 677,396 total or 4 per transaction with non-bulk transactions.
 - Executes: 626,378 or 47 per transaction versus 679,572 total or 4 per transaction with non-bulk transactions.

- Transactions: 13,281 total versus 169,332 with non-bulk transactions. Note that 12 times less user transactions were executed in the bulk transaction mode.
- % Blocks changed per read: 6.29 versus 53.10 with non-bulk transactions.
- Top Five Timed Events:
 - CPU time: 453 seconds or 83.8% total call time versus 270 seconds or 35.1 % total call time with non-bulk transactions.
 - Log file sync: 13,214 waits or 16 second wait time 3.6% total call time versus 171,204 waits or 265 second wait time or 34.4 % total call time with non-bulk transactions. Also note that log file sync retreated from number two to number four top timed events with bulk transactions.
- Time Model Statistics:
 - Sql execute elapsed time: 366.31 seconds or 67.83 % of DB Time versus 393.42 seconds or 51.05 % of DB time with non-bulk transactions.
- Wait Class:
 - Commit: 13,214 waits or 19 second total wait time versus 171,294 waits or 265 second total wait time with non-bulk transactions.
 - Concurrency: 68,609 waits or 27 second total wait time versus 46,354 waits or 149 second total wait time with non-bulk transactions.
 - Configuration: 306 waits or 51 second total wait time versus 610 waits or 137 second total wait time with non-bulk transactions.
- Wait Events:
 - Log file sync: 13,214 waits or 19 second total wait time versus 171,204 waits or 265 second total wait time with non-bulk transactions.
 - Log file switch (checkpoint incomplete): 73 waits or 39 second total wait time versus 315 waits or 124 second total wait time with non-bulk transactions.
 - Buffer busy waits: retreated out of the top five wait events versus 33,890 waits or 117 second total wait time with non-bulk transactions.
- Operating System Statistics: AVG_BUSY_TIME: 71.66 seconds versus 54.54 seconds with non-bulk transactions.
- SQL Statistics: SQL ordered by Elapsed Time: all INSERT SQLs retreated out of the top five SQLs versus the top four SQLs of three INSERTs and one UPDATE with non-bulk transactions. This indicates that INSERT SQLs were less a problem than with non-bulk transactions.
- Buffer Wait Statistics: data block: 8,758 waits or 3 second Total Wait Time versus 31,112 waits or 106 second Total Wait Time with non-bulk transactions.
- Enqueue Activity: TX-Transaction (index contention): 603 waits or 6 second wait time versus 2,122 waits or 31 second wait time with non-bulk transactions.

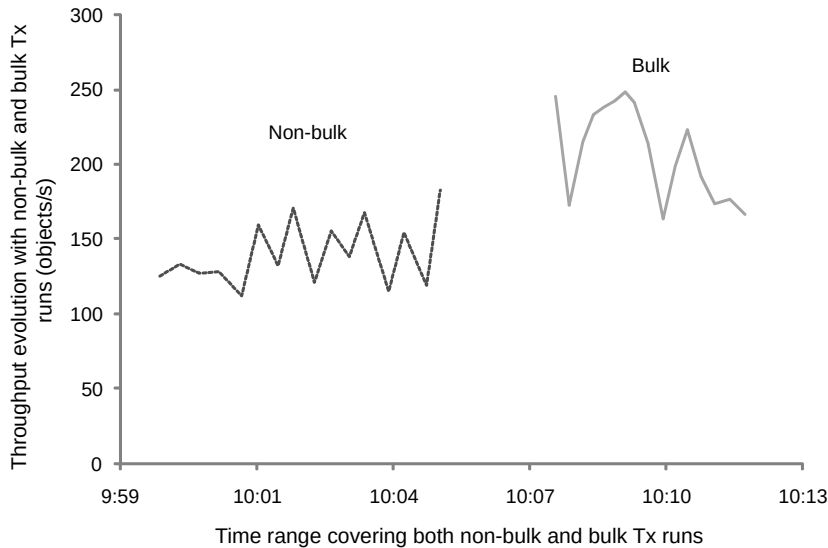


Figure 26.2 Throughput dynamics from the runs using non-bulk and bulk APIs, respectively. The average throughput number was 138 objects per second with the non-bulk API run versus 197 objects per second with the bulk API run. The bulk size was 327 with the bulk API run.

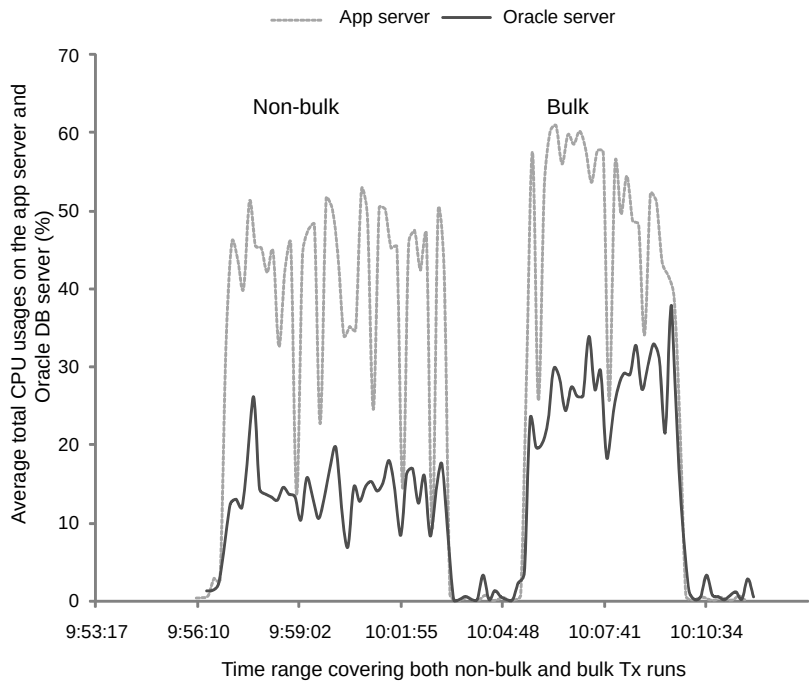


Figure 26.3 Average total CPU utilizations on the app server and Oracle database server recorded during the runs with non-bulk and bulk APIs, respectively. The bulk size was 327 with the bulk API run.

It is seen that the overhead related to transaction commit was reduced significantly when the test was switched from the non-bulk transaction APIs to bulk transaction APIs. This was with a specific case of a bulk size of 327 objects in one transaction. To illustrate the dependence of the benefits of using bulk transactions on the bulk size, a similar test with a smaller bulk size of 5 was conducted with bulk transactions, followed with a non-bulk transaction run. The throughput numbers were 155 and 184 objects per second with non-bulk and bulk transactions, followed by a non-bulk transaction run. This improvement of $184/155 = 1.19$ times was smaller than that of 1.43 times with the preceding test case, which used a larger bulk size of 327. This was expected as the benefits of using bulk transactions diminish when the bulk size gets smaller.

To summarize, Figure 26.2 shows the throughput evolutions of the non-bulk Tx and bulk Tx runs with the bulk size of 327 objects. With the non-bulk Tx run, each object is committed individually, while with the bulk Tx run, all 327 objects were committed once within one bulk transaction. Figure 26.3 shows the corresponding CPU utilizations of the application server and Oracle server which were two separate servers on the same LAN. The app server was an 8-CPU VMWare slice with a total CPU GHz power of $8 \times 2.4 \text{ GHz} = 19.2 \text{ GHz}$ out of a four quad core ESX host system, while the database server was an 8-CPU (two quad-cores) physical machine with a total CPU GHz power of $8 \times 1.86 \text{ GHz} = 14.88 \text{ GHz}$. It is seen that the app server incurred a lot more CPU activities than the database server.

Figures 26.4 and 26.5 are similar to Figures 26.2 and 26.3 except that: (1) the bulk size was 5 instead of 327, and (2) the bulk Tx run was conducted first and then

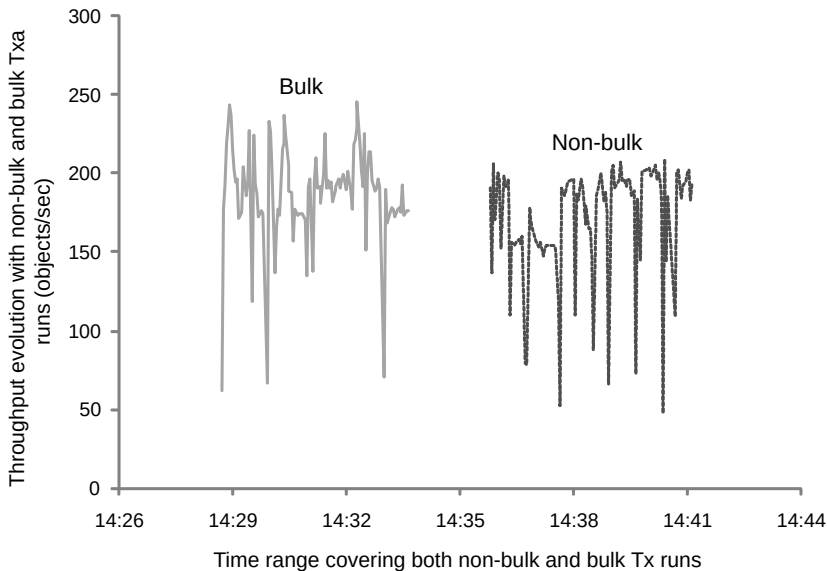


Figure 26.4 Throughput dynamics from the runs using bulk (with a smaller bulk size of 5) and non-bulk APIs, respectively. The average throughput number was 155 objects per second with the non-bulk API run versus 184 objects per second with the bulk API run.

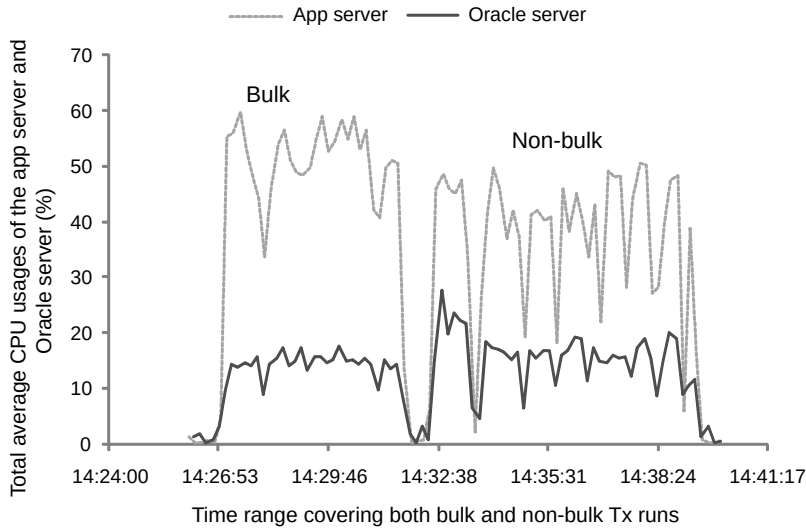


Figure 26.5 Average total CPU utilizations on the app server and Oracle database server recorded during the runs with bulk and non-bulk APIs, respectively. The bulk size was 5 with the bulk API run.

followed with a non-bulk Tx run. In both cases, higher throughput numbers were obtained with bulk Tx runs, driven by higher CPU utilizations on both the app server and database server. This is the best case one can expect that by providing a performance-oriented feature of the bulk Tx APIs, hardware resources were utilized more efficiently, resulting in better performance and scalability.

26.5 MORAL OF THE CASE STUDY

Transaction level optimization is an important factor in determining the performance and scalability of an enterprise application based on Oracle or other database platforms. In this case study, we quantitatively demonstrated with a real product that over 40% performance improvement could be achieved with bulk transactions over non-bulk transactions. One could achieve more or less performance gain, depending on the bulk size, which is application and workload dependent.

In addition to bulk versus non-bulk transaction techniques, another common scenario is with demarcated transactions where different demarcation options define different scopes that have different performance and scalability implications. This subject is beyond the scope of this text.

RECOMMENDED READING

To understand transactions in the context of database and applications, refer to the following texts:

- J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques* (The Morgan Kaufmann Series in Data Management Systems), 1st edn, Morgan Kaufmann, San Francisco, 1992.
- M. Little, J. Maron, and G. Pavlik, *Java Transaction Processing: Design and Implementation*, Prentice Hall, Upper Saddle River, 2004.

EXERCISES

- 26.1** If you are working on a database-centric enterprise application, find out whether bulk transaction is exposed as an option at the server API level. If it is, then design an elaborate test case and quantify the benefits of bulk over non-bulk transactions from performance and scalability perspectives.
- 26.2** If you are working on a database-centric enterprise application, explore the database (or system) transaction boundaries implemented. An enterprise application can run in non-managed (i.e., standalone, simple Web- or Swing applications) or managed environments such as J2EE and .Net. In a non-managed environment, the application developer has to manually set transaction boundaries, in other words, begin, commit, or rollback database transactions. A managed environment usually provides container-managed transactions, with the transaction assembly defined declaratively through external configuration files. Programmatic transaction demarcation is then no longer necessary. If viable, quantify the effects of demarcated versus non-demarcated transactions on the performance and scalability of your application.
- 26.3** In contrast to the case study of covering indexes, why was the high buffer gets per transaction in this case study less an issue?

27

Case Study: Missing Statistics

I hear and I forget. I see and I remember, I do and I understand.
—Confucius

This case study quantitatively demonstrates that sometimes an Oracle database performance issue could be just due to stale or missing statistics. It's actually hard to make a quick, accurate judgment that stale statistics are the only factor and everything else is fine. Whenever in doubt and feasible, just update the database statistics and see if the problem goes away. However, there might be some constraints in production on whether a DBA can manually run a database statistics gathering job at will. So this case study is more educational than an advice for production environments. In production, a DBA needs to take a heuristic approach and come up with a more elaborate plan on what database objects and when to run statistics gathering job. The details are beyond the scope of this text.

This case study consists of the following sections:

- Decaying Performance due to Missing Statistics
- First Run with No Statistics
- Second Run with Missing Statistics

- Third Run with Updated Statistics
- Moral of the Case Study

Next, we begin with illustrating how the performance of an Oracle-based application decayed quickly with time and how it was boosted significantly after the database statistics were updated.

27.1 DECAYING PERFORMANCE DUE TO MISSING STATISTICS

For this case study, the application architecture and workload were similar to those associated with the previous case studies, namely, a Java program inserting objects into an Oracle database with 10 threads and 16 iterations for a total number of 52,320 objects per run. So after three consecutive runs, there would be $52,320 \times 3 = 156,960$ objects in the database.

Figure 27.1 shows how quickly the throughput of the application decayed during the first and second runs (the first and second segments on the figure), and then how it was improved drastically after the application schema was analyzed prior to the third run (the third segment in the figure). The average throughput numbers were 101, 74, and 130 objects per second for those three runs, respectively (note the choppy throughput evolution curve for each run and we have to use the statistical average to compare these runs with each other). It's obvious that the decaying performance was due to missing statistics required for the Oracle optimizer to choose better execution

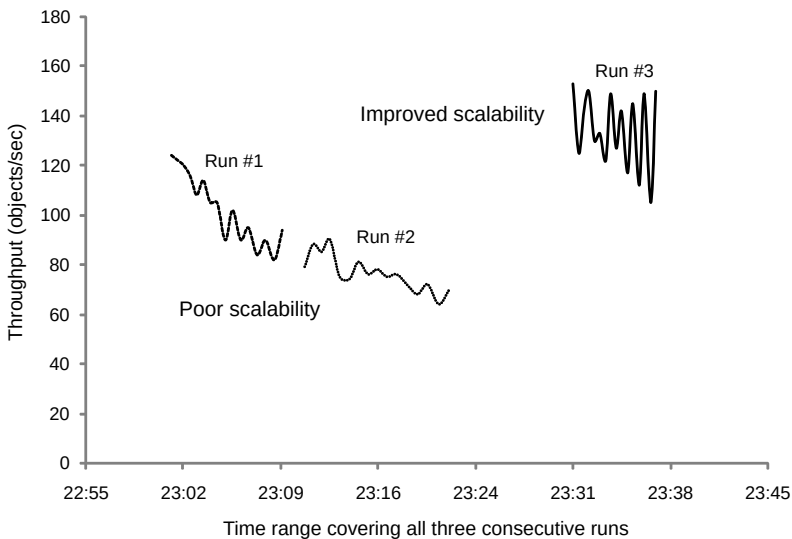


Figure 27.1 Throughput dynamics of the three consecutive runs. Run #1 started with an empty database, run #2 started with the objects inserted into the database from run #1 but with no optimizer statistics gathered, while the optimizer statistics were analyzed prior to launching run #3. It is seen that throughput was deteriorating until the database was analyzed prior to launching run #3.

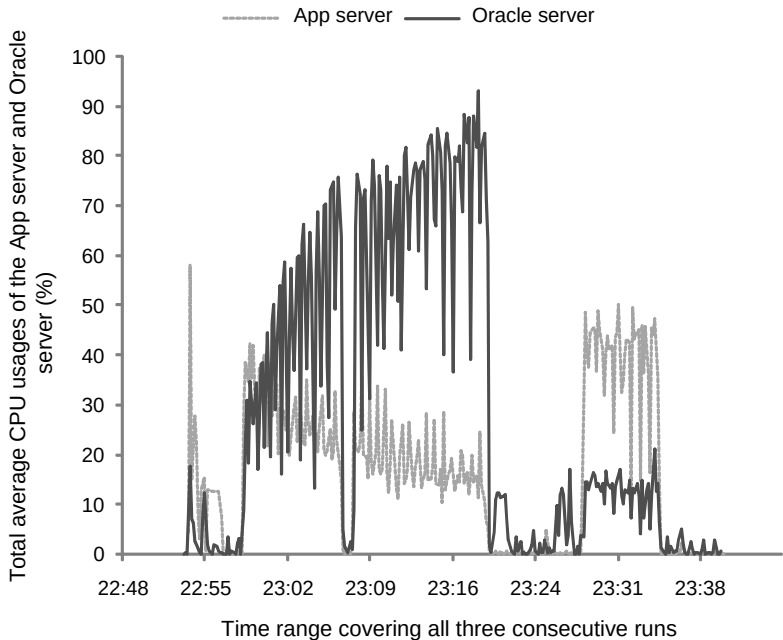


Figure 27.2 CPU utilization dynamics on both the application server and the Oracle server during the entire period of the three consecutive runs.

plans, as the only thing that occurred prior to the third run was a manual statistics update for the entire application schema.

To illustrate the effects of the database statistics update on system resources, Figure 27.2 shows how CPU utilizations evolved both on the application server and on the Oracle database server. It is seen that the Oracle database server CPU utilization grew rapidly to the range of 80% to 90% during the first and second runs when there were no database statistics gathered. During the third run with the database analyzed manually *a priori*, the Oracle database server CPU utilization stayed around 15% with a flat trend. The application server followed an opposite pattern in contrast to the Oracle server. It's even more interesting to see that if the application server and the Oracle database server were installed on the same system, then the total CPU utilization would be more or less flat, which disguises the issue to some extent. If the application and Oracle database were installed on the same system, the two processes need to be monitored separately in order to identify the issue.

To be complete, Figure 27.3 shows the average total network traffic in total Mbytes/s (send and receive combined) on the Oracle server, whereas Figure 27.4 shows the average disk IO usage evolved during the entire test period that covered those three consecutive runs. It's clear that neither network nor I/O was a performance factor in this case, as expected.

Next, we focus on analyzing how the effects of missing optimizer statistics can be identified in AWR reports.

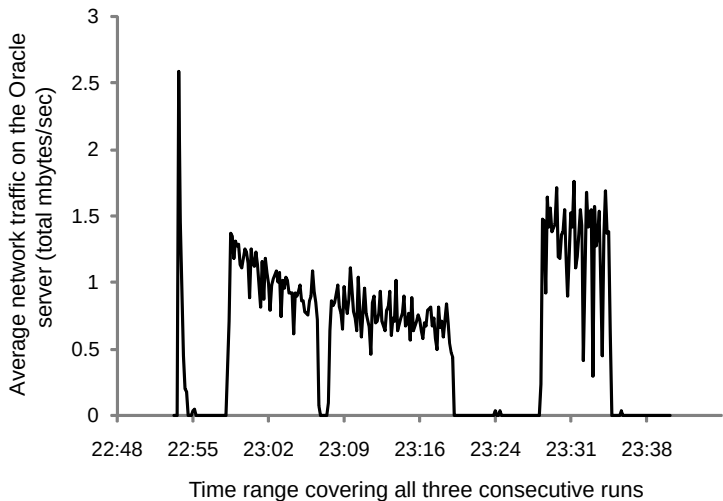


Figure 27.3 Network utilization dynamics on the Oracle server during the entire period of the three consecutive runs.

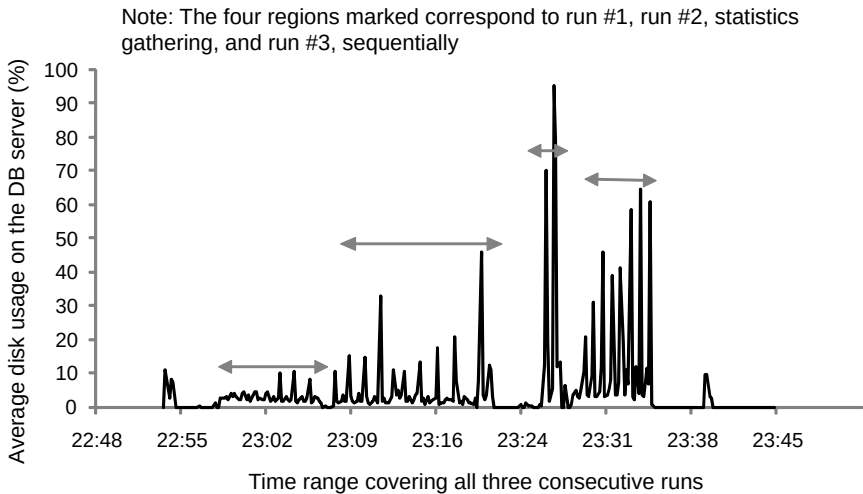


Figure 27.4 Disk I/O utilization dynamics on the Oracle server during the entire period of the three consecutive runs.

27.2 FIRST RUN WITH NO STATISTICS

This section lists the major statistics from the AWR report taken with the two snapshots that covered the first run. Since the test started with an empty database without having any objects inserted into the database from the Java test driver, this provides a reference point about what an AWR report would look like when there had been no database statistics to start with at all.

As with the previous case studies, you are encouraged to go over the following AWR report first to help enhance your ability to read an AWR report. You might want to try to find out the statistics that imply the abnormality due to missing statistics. That would be a potential opportunity for improving the performance and scalability of the application under test. Then you can compare your findings with my analysis given after this AWR report. Hopefully you will find that my analysis reinforces your findings, which will no doubt help enhance your practical ability to troubleshoot the performance and scalability issues with your product.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	XI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2232	10-Sep-10 22:57:55	103	3.6		
End Snap:	2233	10-Sep-10 23:07:14	103	3.6		
Elapsed:		9.31 (mins)				
DB Time:		37.59 (mins)				

27.2.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,752M	2,752M	Std Block Size:	8K
Shared Pool Size:	2,064M	2,064M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction		
Redo size:	1,233,323.11	4,064.10		
Logical reads:	85,667.79	282.30		
Block changes:	6,343.21	20.90		
Physical reads:	0.17	0.00		
Physical writes:	118.65	0.39		
User calls:	2,607.72	8.59		
Parses:	1,301.07	4.29		
Hard parses:	43.12	0.14		
Sorts:	140.77	0.46		
Logons:	0.04	0.00		
Executes:	1,304.63	4.30		
Transactions:	303.47			
% Blocks changed per Read:		7.40	Recursive Call %:	22.77
Rollback per transaction %:		0.00	Rows per Sort:	0.16

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.96	Redo NoWait %:	99.96
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	97.28	Soft Parse %:	96.69
Execute to Parse %:	0.27	Latch Hit %:	94.65
Parse CPU to Parse Elapsed %:	27.98	% Non-Parse CPU:	93.72

Shared Pool Statistics

	Begin	End
Memory Usage %:	41.00	72.61
% SQL with executions>1:	76.65	94.07
% Memory for SQL w/exec>1:	84.50	97.23

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		1,664		73.8	
library cache pin	18,172	132	7	5.8	Concurrency
log file switch (checkpoint incomplete)	213	127	596	5.6	Configuration
kksfbc child completion	2,133	118	55	5.2	Other
log file sync	172,835	96	1	4.2	Commit

27.2.2 Wait Events Statistics**Time Model Statistics**

- Total time in database user-calls (DB Time) : 2255.4s

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	1,685.67	74.74
DB CPU	1,663.72	73.77
parse time elapsed	657.93	29.17
hard parse elapsed time	295.12	13.09
hard parse (sharing criteria) elapsed time	22.22	0.99
hard parse (bind mismatch) elapsed time	22.20	0.98
PL/SQL execution elapsed time	0.27	0.01
connection management call elapsed time	0.02	0.00
PL/SQL compilation elapsed time	0.00	0.00
repeated bind elapsed time	0.00	0.00
DB time	2,255.35	
background elapsed time	51.63	
background cpu time	33.45	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Concurrency	1,326,442	0.00	266	0	7.82
Configuration	545	21.10	135	248	0.00
Other	10,216	44.53	119	12	0.06
Commit	172,835	0.02	96	1	1.02
System I/O	157,063	0.00	18	0	0.93
Network	949,933	0.00	1	0	5.60
User I/O	212	0.00	0	0	0.00
Application	50	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
library cache pin	18,172	0.00	132	7	0.11
log file switch (checkpoint incomplete)	213	50.23	127	596	0.00
kksfbc child completion	2,133	99.62	118	55	0.01
log file sync	172,835	0.02	96	1	1.02
latch: library cache	47,451	0.00	76	2	0.28

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	22,995
AVG_IDLE_TIME	32,824
AVG_SYS_TIME	2,350
AVG_USER_TIME	20,631
BUSY_TIME	184,083
IDLE_TIME	262,742
SYS_TIME	18,906
USER_TIME	165,177
RSRC_MGR_CPU_WAIT_TIME	0
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

Service Statistics

- ordered by DB Time

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
SYS\$USERS	2,252.60	1,663.30	7	47,846,123
ORA10GR2	2.70	0.40	3	4,758
ORA10GR2XDB	0.00	0.00	0	0
SYS\$BACKGROUND	0.00	0.00	84	2,994

27.2.3 SQL Statistics

SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
1,448	1,126			64.21	<u>f8xs5864bzp26</u>		SELECT * FROM (SELECT T179.C...
61	16	24,320	0.00	2.70	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
45	28	26,080	0.00	1.98	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
1,126	1,448			64.21	<u>f8xs5864bzp26</u>		SELECT * FROM (SELECT T179.C...
28	45	26,080	0.00	1.98	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
16	61	24,320	0.00	2.70	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 47,853,772
- Captured SQL account for 16.1% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1,214,870	26,080	46.58	2.54	28.20	44.65	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
579,261	24,320	23.82	1.21	16.47	60.82	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
391,851	25,760	15.21	0.82	12.13	21.35	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300=: "S...

SQL ordered by Sharable Memory

- Only Statements with Sharable Memory greater than 1,048,576 are displayed

Sharable Mem (b)	Executions	% Total	SQL Id	SQL Module	SQL Text
720,487,056		33.29	f8xs5864bzp26		SELECT * FROM (SELECT T179.C...
1,441,817	1	0.07	31a13pnjps7j3		SELECT source, (case w...

Complete List of SQL Text (only one included here to save space)

SQL Id	SQL Text
f8xs5864bzp26	SELECT * FROM (SELECT T179.C1, C490008000, C490009000, C179, C400129200 FROM T179 WHERE ((T179.C490009000 = :''SYS_B_0'') AND (T179.C400079600 = :''SYS_B_1'') AND ((T179.C400129100 IS NULL) OR (T179.C400129100 = :''SYS_B_2''))) ORDER BY :''SYS_B_3'' ASC) WHERE ROWNUM <= :''SYS_B_4''

27.2.4 IO Stats

Tablespace IO Stats

- ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	14	0	0.00	1.00	28,397	51	19,865	2.00
UNDOTBS1	14	0	0.00	1.00	25,405	45	1,287	3.26

File IO Stats

- ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	D:\ODVAPP01.DBF	14	0	0.00	1.00	28,397	51	19,865	2.00
UNDOTBS1	D:\ODV\UNDOTBS01.DBF	14	0	0.00	1.00	25,405	45	1,287	3.26

27.2.5 Wait Statistics

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	18,909	40	2
undo header	1,163	4	4

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	1,498	1,498	0	1,496	10	6.51
HW-Segment High Water Mark	3,223	3,222	1	226	0	0.62
TX-Transaction	179,664	179,467	0	124	0	1.11
FB-Format Block	1,599	1,599	0	102	0	0.77
TX-Transaction (allocate ITL entry)	1	1	0	1	0	15.00
CF-Controlfile Transaction	452	452	0	6	0	0.00
CU-Cursor	33,950	33,896	0	1	0	0.00

27.2.6 init.ora Parameters

Parameter Name	Begin value	End value (if different)
compatible	10.2.0.1.0	
cursor_sharing	SIMILAR	
db_block_size	8192	
db_file_multiblock_read_count	16	
db_name	ORA10GR2	
db_recovery_file_dest_size	2147483648	
job_queue_processes	10	
open_cursors	300	
pga_aggregate_target	1707081728	
processes	150	
sga_target	5133828096	
undo_management	AUTO	
undo_tablespace	UNDOTBS1	

Now let's try to identify the anomalous metric values from the above AWR report, section to section, which could help us realize that the performance of this application could have been better if all the necessary database statistics were in place. First, we need to convert the total elapsed time of 9.31 minutes displayed in the header section to 559 seconds so that we can convert all metric values represented in per second into absolute values. It makes more sense to look at some of the metric values in terms of "total" rather than "per second."

In summary, the following metric values look more worrisome:

- Load Profile: Logical reads: 47,854,027 total or 282.30 per transaction
- Instance Efficiency: Parse CPU to Parse Elapsed %: 27.98
- Top Five Timed Events: CPU time: 1,664 seconds or 73.8%
- Time Model Statistics: sql execute elapsed time: 1,685.67 seconds or 74.74%
- Wait class: Concurrency: 1,326,442 waits or 266 second total wait time
- Operating System Statistics: AVG_BUSY_TIME: 230 seconds
- SQL Statistics:
 - Elapsed time of one SQL with ID f8xs. . . : 1,448 seconds or 64.21% total DB time
 - CPU time of the same SQL: 1,126 seconds or 64.21% total DB time
 - Buffer Gets: spread uniformly across all SQLs
 - Sharable Memory: 720 MB or 33.29% total for the same SQL (note the number of executions missing)
 - Version Count: 17,263 for the same SQL

Out of all these seemingly anomalous metric values listed above, the most conspicuous ones are the high logical reads well spread across *all* SQLs, suggesting that database statistics might be missing or stale.

27.3 SECOND RUN WITH MISSING STATISTICS

This section lists some major statistics from the AWR report that covers the entire duration of the second run. During this run, the performance of the application had further deteriorated due to missing database statistics. Again, the question is how we could know the database had missing statistics based on the information available from this report.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	XI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2233	10-Sep-10 23:07:14	103	3.6		
End Snap:	2234	10-Sep-10 23:20:00	101	3.5		
Elapsed:		12.76 (mins)				
DB Time:		74.44 (mins)				

27.3.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,752M	2,656M	Std Block Size:	8K
Shared Pool Size:	2,064M	2,160M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction	
Redo size:	899,702.67	4,065.32	
Logical reads:	161,682.82	730.57	
Block changes:	4,608.39	20.82	
Physical reads:	1.23	0.01	
Physical writes:	145.27	0.66	
User calls:	1,900.11	8.59	
Parses:	946.56	4.28	
Hard parses:	30.70	0.14	
Sorts:	103.42	0.47	
Logons:	0.04	0.00	
Executes:	950.20	4.29	
Transactions:	221.31		
% Blocks changed per Read:	2.85	Recursive Call %:	22.47
Rollback per transaction %:	0.00	Rows per Sort:	0.22

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.99	Redo NoWait %:	99.94
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	97.31	Soft Parse %:	96.76
Execute to Parse %:	0.38	Latch Hit %:	97.04
Parse CPU to Parse Elapsed %:	34.57	% Non-Parse CPU:	96.28

Shared Pool Statistics

	Begin	End
Memory Usage %:	72.61	71.45
% SQL with executions>1:	94.07	91.81
% Memory for SQL w/exec>1:	97.23	97.76

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		3,739		83.7	
log file sync	173,221	177	1	4.0	Commit
library cache pin	16,041	161	10	3.6	Concurrency
log file switch (checkpoint incomplete)	193	119	615	2.7	Configuration
latch: library cache	44,196	104	2	2.3	Concurrency

27.3.2 Wait Events Statistics**Time Model Statistics**

- Total time in database user-calls (DB Time) : 4466.2s

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	3,782.10	84.68
DB CPU	3,739.08	83.72
parse time elapsed	676.02	15.14
hard parse elapsed time	286.73	6.42
hard parse (sharing criteria) elapsed time	15.13	0.34
hard parse (bind mismatch) elapsed time	14.95	0.33
PL/SQL execution elapsed time	0.43	0.01
connection management call elapsed time	0.02	0.00
PL/SQL compilation elapsed time	0.02	0.00
repeated bind elapsed time	0.01	0.00
sequence load elapsed time	0.00	0.00
DB time	4,466.23	
background elapsed time	83.35	
background cpu time	42.13	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Concurrency	1,970,389	0.00	344	0	11.62
Commit	173,221	0.04	177	1	1.02
Configuration	343	43.73	135	393	0.00
Other	7,446	37.66	94	13	0.04
System I/O	164,139	0.00	38	0	0.97
User I/O	991	0.10	5	5	0.01
Network	949,345	0.00	1	0	5.60
Application	80	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	173,221	0.04	177	1	1.02
library cache pin	16,041	0.00	161	10	0.09
log file switch (checkpoint incomplete)	193	42.49	119	615	0.00
latch: library cache	44,196	0.00	104	2	0.26
kksfbc child completion	1,622	99.94	93	57	0.01

Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	49,145
AVG_IDLE_TIME	27,405
AVG_SYS_TIME	2,782
AVG_USER_TIME	46,340
BUSY_TIME	393,352
IDLE_TIME	219,423
SYS_TIME	22,411
USER_TIME	370,941
RSRC_MGR_CPU_WAIT_TIME	0
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

27.3.3 SQL Statistics**SQL ordered by Elapsed Time**

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
3,277	2,900			73.36	<u>f8xs5864bzp26</u>		SELECT * FROM (SELECT T179.C...
79	32	26,080	0.00	1.77	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
53	11	24,319	0.00	1.19	<u>5nbpkagga7a0h</u>	app.exe	INSERT INTO T226 (C2, C7, C8, ...

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
2,900	3,277			73.36	<u>f8xs5864bzb26</u>		SELECT * FROM (SELECT T179.C...
32	79	26,080	0.00	1.77	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
16	35	24,320	0.00	0.79	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 123,829,638
- Captured SQL account for 6.4% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1,352,065	26,080	51.84	1.09	32.10	79.14	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
609,361	24,320	25.06	0.49	16.41	35.41	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
455,950	25,760	17.70	0.37	13.40	16.43	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300="S...

SQL ordered by Reads

- Total Disk Reads: 943
- Captured SQL account for 50.7% of Total

Physical Reads	Executions	Reads per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
232	26,080	0.01	24.60	32.10	79.14	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
44	24,320	0.00	4.67	16.41	35.41	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
42	25,600	0.00	4.45	14.07	27.38	<u>dvsmsg0345ng4m</u>	app.exe	INSERT INTO T177 (C2, C7, C8, ...
36	24,319	0.00	3.82	10.92	53.13	<u>5nbpkagga7a0h</u>	app.exe	INSERT INTO T226 (C2, C7, C8, ...

SQL ordered by Sharable Memory

- Only Statements with Sharable Memory greater than 1,048,576 are displayed

Sharable Mem (b)	Executions	% Total	SQL Id	SQL Module	SQL Text
893,901,800		39.47	f8xs5864bzp26		SELECT * FROM (SELECT T179.C...

SQL ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
21,377		f8xs5864bzp26		SELECT * FROM (SELECT T179.C...

Complete List of SQL Text (only one included here)

SQL Id	SQL Text
f8xs5864bzp26	SELECT * FROM (SELECT T179.C1, C490008000, C490009000, C179, C400129200 FROM T179 WHERE ((T179.C490009000 = :SYS_B_0) AND (T179.C400079600 = :SYS_B_1) AND (T179.C400129100 IS NULL) OR (T179.C400129100 = :SYS_B_2))) ORDER BY :SYS_B_3" ASC) WHERE ROWNUM <= :SYS_B_4"

27.3.4 IO Stats

Tablespace IO Stats

- ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	591	1	3.94	1.06	64,140	84	9,902	6.04
UNDOTBS1	20	0	1.50	1.00	29,223	38	739	12.65

27.3.5 Wait Statistics

Buffer Wait Statistics

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	9,490	60	6
undo header	626	9	15
undo block	113	0	0
1st level bmb	392	0	0
segment header	17	0	0
2nd level bmb	5	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	950	950	0	908	2	2.25
CF-Controlfile Transaction	618	618	0	7	0	20.00
CU-Cursor	34,784	34,752	0	2	0	8.00
HW-Segment High Water Mark	1,188	1,188	0	34	0	0.00
FB-Format Block	1,071	1,071	0	19	0	0.00
TX-Transaction	179,858	179,735	0	8	0	0.00

Now let's try to identify further the anomalous metric values from the above AWR report taken from a test run that had non-existing database statistics during the entire run. Once again, we need to convert the total elapsed time of 12.76 minutes displayed in the header section to 766 seconds so that we can convert all metric values represented in per second into absolute values. Compare the total elapsed time of 12.76 minutes from this run with the total elapsed time of 9.31 minutes or 559 seconds from the previous run.

In summary, the following metric values look more worrisome:

- Load Profile: Logical reads: 123,840,040 or 730.57 per transaction versus 47,854,027 total or 282.30 per transaction for the previous run
- Instance Efficiency: Parse CPU to Parse Elapsed %: 34.57% versus 27.98% for the previous run
- Top Five Timed Events: CPU time: 3,739 seconds or 83.7% versus 1,664 seconds or 73.8% for the previous run
- Time Model Statistics: sql execute elapsed time: 3,782.10 seconds or 84.68% versus 1,685.67 seconds or 74.74% for the previous run
- Wait class: Concurrency: 1,970,389 waits or 344 second total wait time versus 1,326,442 waits or 266 second total wait time for the previous run
- Operating System Statistics: AVG_BUSY_TIME: 491 seconds versus 230 seconds for the previous run
- SQL Statistics:
 - Elapsed time of one SQL with ID f8xs. . . : 3,277 seconds or 73.36% total DB time versus 1,448 seconds or 64.21% total DB time for the previous run
 - CPU time of the same SQL: 2,900 seconds or 73.36% versus 1,126 seconds or 64.21% for the previous run
 - Buffer Gets: spread even more uniformly across all SQLs than the previous run

- Sharable Memory: 894 MB or 39.47% versus 720 MB or 33.29% total for the same SQL (note the number of executions missing) for the previous run
- Version Count: 21,377 for the same SQL versus 17,263 for the previous run

It is seen that the number of logical reads had increased from about 5 million to about 124 millions—almost exponentially—the same number of similar transactions. The fact that this huge number of logical reads was spread across all SQLs could be used as an indicator that the database statistics might be missing or stale. In the next section, the AWR report taken with the third run that had all database statistics created manually right prior to the test will reveal how those metric values had changed with significantly improved application performance.

27.4 THIRD RUN WITH UPDATED STATISTICS

This section lists the major statistics from the AWR report that covered the third run which had the statistics manually gathered for the entire schema *a priori*. What we hope to find out is what had changed compared with the previous two runs that had no statistics in the database.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORA10GR2	4022777666	ora10gr2	1	10.2.0.1.0	NO	XI1
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	2235	10-Sep-10 23:27:36	102	3.9		
End Snap:	2236	10-Sep-10 23:35:00	103	3.9		
Elapsed:		7.40 (mins)				
DB Time:		13.66 (mins)				

27.4.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	2,656M	2,656M	Std Block Size:	8K
Shared Pool Size:	2,160M	2,160M	Log Buffer:	14,416K

Load Profile

	Per Second	Per Transaction		
Redo size:	1,650,951.10	4,325.91		
Logical reads:	15,439.72	40.46		
Block changes:	8,061.42	21.12		
Physical reads:	0.70	0.00		
Physical writes:	325.19	0.85		
User calls:	3,276.13	8.58		
Parses:	1,523.49	3.99		
Hard parses:	0.15	0.00		
Sorts:	177.04	0.46		
Logons:	0.04	0.00		
Executes:	1,526.70	4.00		
Transactions:	381.64			
% Blocks changed per Read:		52.21	Recursive Call %:	11.41
Rollback per transaction %:		0.00	Rows per Sort:	0.16

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.07	Redo NoWait %:	99.94
Buffer Hit %:	100.00	In-memory Sort %:	100.00
Library Hit %:	100.00	Soft Parse %:	99.99
Execute to Parse %:	0.21	Latch Hit %:	99.46
Parse CPU to Parse Elapsed %:	95.98	% Non-Parse CPU:	81.49

Shared Pool Statistics

	Begin	End
Memory Usage %:	67.91	67.86
% SQL with executions>1:	54.79	96.23
% Memory for SQL w/exec>1:	79.41	98.64

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
log file sync	172,011	308	2	37.6	Commit
CPU time		277		33.8	
log file switch (checkpoint incomplete)	239	121	508	14.8	Configuration
buffer busy waits	63,594	92	1	11.2	Concurrency
enq: TX - index contention	8,491	75	9	9.2	Concurrency

27.4.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 819.7s

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	393.49	48.00
DB CPU	277.25	33.82
parse time elapsed	56.66	6.91
PL/SQL execution elapsed time	0.25	0.03
hard parse elapsed time	0.12	0.01
connection management call elapsed time	0.01	0.00
hard parse (sharing criteria) elapsed time	0.01	0.00
PL/SQL compilation elapsed time	0.00	0.00
sequence load elapsed time	0.00	0.00
repeated bind elapsed time	0.00	0.00
DB time	819.71	
background elapsed time	123.50	
background cpu time	31.75	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
Commit	172,011	0.11	308	2	1.01
Concurrency	88,362	0.05	168	2	0.52
Configuration	610	17.38	126	206	0.00
System I/O	149,832	0.00	75	1	0.88
User I/O	715	0.00	2	3	0.00
Network	949,051	0.00	1	0	5.60
Other	18,300	56.01	1	0	0.11
Application	42	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	172,011	0.11	308	2	1.01
log file switch (checkpoint incomplete)	239	42.68	121	508	0.00
buffer busy waits	63,594	0.06	92	1	0.38
enq: TX - index contention	8,491	0.00	75	9	0.05
db file parallel write	3,245	0.00	37	11	0.02

27.4.3 Operating System Statistics

Statistic	Total
AVG_BUSY_TIME	5,209
AVG_IDLE_TIME	39,170
AVG_SYS_TIME	2,057
AVG_USER_TIME	3,137
BUSY_TIME	41,780
IDLE_TIME	313,484
SYS_TIME	16,574
USER_TIME	25,206
RSRC_MGR_CPU_WAIT_TIME	0
PHYSICAL_MEMORY_BYTES	17,178,804,224
NUM_CPUS	8
NUM_CPU_CORES	2

Service Statistics

Service Name	DB Time (s)	DB CPU (s)	Physical Reads	Logical Reads
SYS\$USERS	819.20	276.80	185	6,848,632
ORA10GR2	0.50	0.40	14	5,657
ORA10GR2XDB	0.00	0.00	0	0
SYS\$BACKGROUND	0.00	0.00	109	2,681

Service Wait Class Stats

Service Name	User I/O Total Wts	User I/O Wt Time	Concurcy Total Wts	Concurcy Wt Time	Admin Total Wts	Admin Wt Time	Network Total Wts	Network Wt Time
SYS\$USERS	484	115	88351	16759	0	0	948416	118
ORA10GR2	14	6	0	0	0	0	626	0
SYS\$BACKGROUND	217	99	11	7	0	0	0	0

27.4.4 SQL Statistics

SQL ordered by Elapsed Time

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
90	29	26,079	0.00	10.99	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
46	17	24,319	0.00	5.65	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
39	5	24,320	0.00	4.79	<u>96y1bkvwymb1q</u>	app.exe	INSERT INTO H226 (entryId, T0,...

SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
29	90	26,079	0.00	10.99	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
17	46	24,319	0.00	5.65	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
14	34	25,598	0.00	4.10	<u>dvsmg0345ng4m</u>	app.exe	INSERT INTO T177 (C2, C7, C8, ...

SQL ordered by Gets

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- Total Buffer Gets: 6,856,963
- Captured SQL account for 94.1% of Total

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
1,524,895	26,079	58.47	22.24	28.56	90.10	<u>bq7rd3yrf2827</u>	app.exe	INSERT INTO T179 (C2, C7, C8, ...
676,597	24,319	27.82	9.87	17.25	46.29	<u>gr95kumnq95nj</u>	app.exe	INSERT INTO T236 (C2, C7, C8, ...
501,493	25,760	19.47	7.31	12.12	39.08	<u>d6cn7x5924z69</u>	app.exe	UPDATE T236 SET C400131300=-"S...
364,369	25,598	14.23	5.31	13.93	33.65	<u>dvsmg0345ng4m</u>	app.exe	INSERT INTO T177 (C2, C7, C8, ...
338,457	24,320	13.92	4.94	10.82	32.78	<u>5nbpkagga7a0h</u>	app.exe	INSERT INTO T226 (C2, C7, C8, ...

SQL ordered by Parse Calls

- Total Parse Calls: 676,600
- Captured SQL account for 94.3% of Total

Parse Calls	Executions	% Total Parses	SQL Id	SQL Module	SQL Text
52,313	52,307	7.73	<u>dyrf8h354f0dd</u>	app.exe	SELECT T264.C1, C20051, C20052...
52,154	52,037	7.71	<u>c6801bm8xj79j</u>	app.exe	SELECT T236.C1, C400079600, C4...
51,520	51,499	7.61	<u>4jd6fdxb15ugs</u>	app.exe	SELECT C179, C400079600 FROM T...
48,639	48,611	7.19	<u>f7r3zwcq8th0j</u>	app.exe	SELECT C400079600 FROM T273 WH...
26,238	26,240	3.88	<u>0k3ta9hv7s6hb</u>	app.exe	INSERT INTO H236 (entryId, T0,...
26,080	26,080	3.85	<u>g9ttytv00vsax</u>	app.exe	INSERT INTO H179 (entryId, T0,...

SQL ordered by Sharable Memory

- Only Statements with Sharable Memory greater than 1,048,576 are displayed

Sharable Mem (b)	Executions	% Total	SQL Id	SQL Module	SQL Text
687,397,936		30.35	<u>f8xs5864bzp26</u>		SELECT * FROM (SELECT T179.C...

SQL ordered by Version Count

- Only Statements with Version Count greater than 20 are displayed

Version Count	Executions	SQL Id	SQL Module	SQL Text
16,439		<u>f8xs5864bzp26</u>		SELECT * FROM (SELECT T179.C...

Complete List of SQL Text

SQL Id	SQL Text
f8xs5864bzp26	SELECT * FROM (SELECT T179.C1, C490008000, C490009000, C179, C400129200 FROM T179 WHERE ((T179.C490009000 = "SYS_B_0") AND (T179.C400079600 = "SYS_B_1") AND ((T179.C400129100 IS NULL) OR (T179.C400129100 = "SYS_B_2")))) ORDER BY "SYS_B_3" ASC) WHERE ROWNUM <= "SYS_B_4"

27.4.5 Wait Statistics**Buffer Wait Statistics**

- ordered by wait time desc, waits desc

Class	Waits	Total Wait Time (s)	Avg Time (ms)
data block	58,548	86	1
undo header	2,600	5	2
1st level bmb	2,340	0	0
segment header	43	0	0
2nd level bmb	24	0	0
undo block	13	0	0

Enqueue Activity

- only enqueues with waits are shown
- Enqueue stats gathered prior to 10g should not be compared with 10g data
- ordered by Wait Time desc, Waits desc

Enqueue Type (Request Reason)	Requests	Succ Gets	Failed Gets	Waits	Wt Time (s)	Av Wt Time(ms)
TX-Transaction (index contention)	8,491	8,491	0	8,236	76	9.19
CF-Controlfile Transaction	451	451	0	9	0	13.89
FB-Format Block	1,143	1,143	0	121	0	0.13
HW-Segment High Water Mark	1,010	1,010	0	195	0	0.00
TX-Transaction	180,717	180,322	0	58	0	0.00
TX-Transaction (allocate ITL entry)	20	20	0	13	0	0.00

Now let's see how a manual database statistics gathering job had changed the anomalous metric values from the test runs with missing statistics. Once again, we need to convert the total elapsed time of 7.40 minutes displayed in the header section to 444 seconds so that we can convert all metric values represented in per second into absolute values. Compare the total elapsed time of 7.40 minutes from this run with the total elapsed time of 12.76 minutes or 766 seconds from the previous run.

In summary, the following metric values illustrate the changes resulting from a manual database statistics gathering job:

- Load Profile: Logical reads: 6,855,236 or 40.46 per transaction versus 123,840,040 or 730.57 per transaction for the previous run. This is a reduction of a factor of 18.
- Instance Efficiency: Parse CPU to Parse Elapsed %: 95.98 versus 34.57 for the previous run. This implies that parsing was a lot more efficient than before. When the database statistics are up-to-date, the Oracle optimizer has less guesswork to do in determining the optimal execution plans for SQL statements.
- Top Five Timed Events: log file sync became the top wait event with 37.6% total call time. CPU time fell down to 277 seconds or 33.8% versus 1,664 seconds or 73.8% for the previous run.
- Time Model Statistics: sql execute elapsed time: 393.49 seconds versus 3,782.10 seconds for the previous run. This is almost a factor of 10 improvement.
- Wait class: Commit became the top wait class with 308 second total wait time. Concurrency class incurred 88,362 waits or 168 second total wait time versus 1,970,389 waits or 344 second total wait time. One can say that concurrency contention was reduced by a factor of 22 in terms of the number of waits.
- Operating System Statistics: AVG_BUSY_TIME: 52 seconds versus 491 seconds for the previous run. Once again, this is an improvement of over nine times.
- SQL Statistics:
 - Elapsed time of one SQL with ID f8xs...: below 5 seconds or 1% total DB time versus 3,277 seconds 73.36% total DB time for the previous run. The hot SQL previously seen disappeared from the AWR report, which means that it's executed efficiently with new optimal execution plan arrived at by the CBO with fresh statistics provided.
 - CPU time of the same SQL: below 14 seconds or 2% total DB time versus 2,900 seconds or 73.36% for the previous run. Once again, this is a huge improvement of a factor of 207 times.

- Buffer Gets: more related to DML statements. This was a good sign, which means that the SELECT SQL statements were not causing excessive buffer gets.
- Sharable Memory: 687 MB or 30.35% versus 894 MB or 39.47% for the same SQL (note the number of executions missing) for the previous run.
- Version Count: 16,439 versus 21,377 for the same SQL. This might indicate that there still is room to improve this SQL so that it would be less sensitive to database statistics.

All the improved metrics point to the single fact that when the statistics of a database are fresher, the database as well as the application will “breathe” better, leading to better performance and scalability. Let’s wrap up this case study next.

27.5 MORAL OF THE CASE STUDY

From this case study, we see that missing or stale database statistics can cause a database to slow down significantly. The Oracle optimizer depends on proper database statistics to determine optimal execution plans for SQL statements. When such database statistics are missing or become stale, the optimizer will not be able to come up with the most efficient execution plans, resulting in over-usage of the hardware resources. Such problems are curable based on AWR reports, as we have demonstrated in this chapter.

Once again, when a very large number of logical reads are observed, accompanied with a high CPU utilization on the database server, one should not immediately jump to the conclusion that more memory or more powerful CPUs are needed. It’s desirable to keep a database server busy, but only for doing necessary and useful work.

RECOMMENDED READING

Refer to the following Oracle document on Oracle database tuning in general:

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf.

EXERCISES

- 27.1** Compare the Redo size metric values of those three test runs and conclude whether this metric was affected by updating database statistics. Explain why the Redo size metric was or was not affected by database statistics.
- 27.2** Compare the Block changes metric values of those three test runs and conclude whether this metric was affected by updating database statistics. Explain why the Block changes metric was or was not affected by database statistics.

- 27.3** Is it appropriate to use the “per second” values when comparing multiple runs with tunings applied? Give an example using this case study.
- 27.4** Explain the Wait Classes by category from an AWR report. Which wait classes are related to hardware resources, and which wait classes are related to software?

28

Case Study: Misconfigured SAN Storage

Adopt the pace of nature: her secret is patience.

—Ralph Waldo Emerson

This case study provides a compelling example that working on Oracle performance and scalability issues require more than just programming skills or Oracle tuning skills. When performance and scalability bottlenecks are actually with the hardware rather than with how the application and database were designed or implemented, knowledge and experience in hardware (computers, networks, and storage devices, etc.) are indispensable for a successful troubleshooting effort. Specific to this case study, we demonstrate quantitatively how a fast SAN storage could perform poorly when misconfigured.

The particular SAN storage involved here was an Apple's Xserve RAID, which is a high-availability, high-performance, scalable storage solution at a very competitive price. In this case study, we would not delve into the architecture of the application, as the problem was not with the application or Oracle database. The problem was a misconfigured Xserve RAID in the first place. We demonstrate that after this RAID was reconfigured properly, superior disk IO performance was achieved and thus the high performance with the application.

This chapter consists of the following sections:

- Architecture of the Apple's Xserve RAID
- Problem Analysis
- Reconfiguring the RAID and Verifying
- Moral of the Case Study

Let's take this opportunity to get familiar with Apple's Xserve RAID so that we would be able to understand the context of the problem better. This also is a good learning opportunity if you have vague or no knowledge about how a SAN device works, as Apple's Xserve RAID is representative of all common SAN technologies from various vendors on the market.

28.1 ARCHITECTURE OF THE APPLE'S XSERVE RAID

To start with, Figure 28.1 shows what an Apple's Xserve RAID looks like. You are encouraged to map each numbered label with its corresponding component to get a rough idea about how this SAN device works. Some of the key components of this RAID are introduced as follows:

1. First, note that this RAID has two modules, one on the left side and the other on the right side, forming an image reflection of each other. Then at the bottom of each module are 7 disk drives where volumes of data are stored physically.

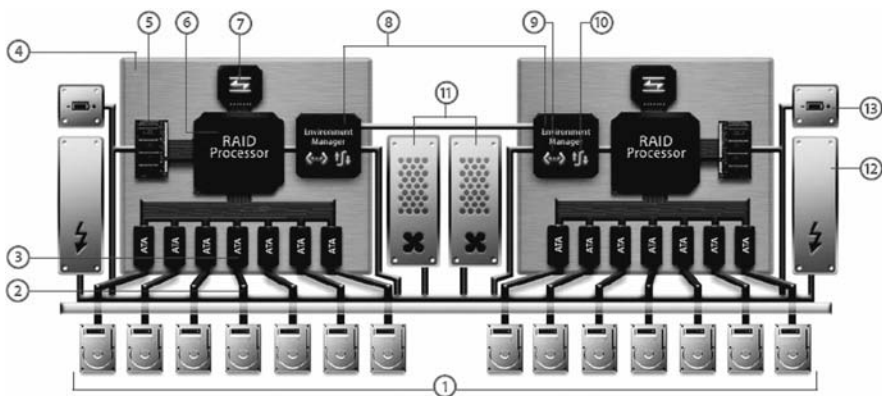


Figure 28.1 Apple Xserve SAN system architecture. The labeled components are: (1) dual drive modules, (2) independent ATA drive channels, (3) drive controllers, (4) RAID controller modules, (5) cache memory, (6) dual independent RAID processors, (7) fibre channel ports, (8) redundant environment managers, (9) Ethernet ports, (10) serial ports, (11) redundant cooling modules, (12) redundant power suppliers, and (13) cache backup battery modules (optional) (Courtesy Apple, Inc.).

These are hot-swappable drives that support RAID levels 0, 1, 3, 5, and 0 + 1. The maximum capacity is about 7 TB with all 14 drives.

- 2. Above the drives are the drive controllers connected with the drives via independent ATA drive channels with a 1:1 connection between a drive and a drive controller.
- 3. The drive controllers are then connected to the dual independent RAID processors via a common bus. This processor chip is the hub to a few other necessary components such as the cache memory (left), fibre channel ports (upper), the Ethernet ports and serial ports (right). This is a cross-bar structure as can be easily identified.

The actual specs of this RAID are not important to this case study. So next we move on to the topic of what problem we encountered. To keep it interesting, let’s first not reveal how it was misconfigured in detail, and the only information available at this point is that it had 7 drives and I requested to have all of them configured as a RAID 0 in a certain way which will be revealed later.

28.2 PROBLEM ANALYSIS

The problem was simple. After it was configured and turned to me for use, I got very horrible performance with the suite of performance and scalability tests I had been running for years. So I took an AWR report which is shared below. It’s easy to identify what the problem was, so I encourage you to find it out first before it’s revealed following this AWR report.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
XI3100M	92986712	xi3100m	1	10.2.0.1.0	NO	XI3
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	7	18-Oct-07 17:31:58	66	8.7		
End Snap:	8	18-Oct-07 18:33:50	68	16.8		
Elapsed:		61.86 (mins)				
DB Time:		596.46 (mins)				

28.2.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	392M	424M	Std Block Size:	8K
Shared Pool Size:	168M	136M	Log Buffer:	6,968K

Load Profile

	Per Second	Per Transaction		
Redo size:	500,828.71	8,917.92		
Logical reads:	4,318.07	76.89		
Block changes:	1,345.48	23.96		
Physical reads:	24.98	0.44		
Physical writes:	92.86	1.65		
User calls:	1,100.90	19.60		
Parses:	489.63	8.72		
Hard parses:	2.37	0.04		
Sorts:	50.07	0.89		
Logons:	0.04	0.00		
Executes:	495.67	8.83		
Transactions:	56.16			
% Blocks changed per Read:		31.16	Recursive Call %:	14.73
Rollback per transaction %:		0.02	Rows per Sort:	2.82

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.74	Redo NoWait %:	99.93
Buffer Hit %:	100.44	In-memory Sort %:	100.00
Library Hit %:	99.31	Soft Parse %:	99.52
Execute to Parse %:	1.22	Latch Hit %:	99.50
Parse CPU to Parse Elapsed %:	40.62	% Non-Parse CPU:	86.90

Shared Pool Statistics

	Begin	End
Memory Usage %:	80.92	95.61
% SQL with executions>1:	77.40	75.81
% Memory for SQL w/exec>1:	84.86	90.91

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
direct path read	81,439	12,045	148	33.7	User I/O
log file sync	218,852	7,415	34	20.7	Commit
db file sequential read	7,857	1,266	161	3.5	User I/O
CPU time		1,187		3.3	
log file switch (checkpoint incomplete)	579	298	516	.8	Configuration

28.2.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 35,787.4s

Statistic Name	Time (s)	% of DB Time
sql execute elapsed time	2,654.02	7.42
DB CPU	1,186.51	3.32
parse time elapsed	561.17	1.57
hard parse elapsed time	335.96	0.94
PL/SQL compilation elapsed time	81.50	0.23
PL/SQL execution elapsed time	28.05	0.08
hard parse (sharing criteria) elapsed time	3.70	0.01
connection management call elapsed time	2.49	0.01
hard parse (bind mismatch) elapsed time	0.87	0.00
sequence load elapsed time	0.07	0.00
repeated bind elapsed time	0.04	0.00
DB time	35,787.36	
background elapsed time	5,286.06	
background cpu time	38.42	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
User I/O	418,830	0.00	13,451	32	2.01
Commit	218,852	0.10	7,415	34	1.05
System I/O	173,588	0.00	816	5	0.83
Configuration	2,952	68.26	330	112	0.01
Concurrency	54,781	0.10	276	5	0.26
Network	3,814,667	0.00	93	0	18.30
Other	11,484	27.80	22	2	0.06
Application	626	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
direct path read	81,439	0.00	12,045	148	0.39
log file sync	218,852	0.10	7,415	34	1.05
db file sequential read	7,857	0.00	1,266	161	0.04
log file switch (checkpoint incomplete)	579	45.60	298	516	0.00
control file sequential read	4,003	0.00	283	71	0.02

28.2.3 IO Stats

Tablespace IO Stats

- ordered by IOs (Reads + Writes) desc

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	84,966	23	198.51	1.00	239,799	65	31,118	3.37
UNDOTBS1	46	0	69.78	1.00	30,696	8	10,205	0.37
SYSTEM	3,277	1	239.20	1.80	913	0	4	577.50
SYSAUX	1,261	0	144.47	1.27	1,488	0	0	0.00
USERS	45	0	82.44	1.00	43	0	0	0.00
TEMP	2	0	160.00	1.00	0	0	0	0.00

File IO Stats

- ordered by Tablespace, File

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	G:\OD\APP01.ORA	41,441	11	350.99	1.00	120,501	32	18,434	3.67
APP	H:\OD\APP00.	43,525	12	53.34	1.00	119,298	32	12,684	2.92
SYSAUX	H:\OD\SYSAUX01.DBF	1,261	0	144.47	1.27	1,488	0	0	0.00
SYSTEM	H:\OD\SYSTEM01.DBF	3,277	1	239.20	1.80	913	0	4	577.50
TEMP	H:\OD\TEMP01.DBF	2	0	160.00	1.00	0	0	0	
UNDOTBS1	H:\OD\UNDOTBS01.D	46	0	69.78	1.00	30,696	8	10,205	0.37
USERS	H:\OD\USERS01.DBF	45	0	82.44	1.00	43	0	0	0.00

28.2.4 init.ora Parameters

Parameter Name	Begin value	End value (if different)
cursor_sharing	SIMILAR	
db_block_size	8192	
db_file_multiblock_read_count	16	
db_name	XI3100M	
job_queue_processes	10	
open_cursors	300	
pga_aggregate_target	203423744	
processes	150	
sga_target	612368384	
undo_management	AUTO	

At this point, you may have noticed from the IO stats section that there was a huge average read time of nearly 200 milliseconds incurred during the test. This was unexpected, as in general the average read time from a SAN storage device should be better than 10 milliseconds. The myth is revealed in the next section.

28.3 RECONFIGURING THE RAID AND VERIFYING

The myth was revealed after communicating with the system administrator who configured this SAN device for me. I was informed that those 7 drives were split into 3 and 4 in the two modules, instead of putting all 7 drives into either one of the two modules as I originally requested. After the RAID was reconfigured with all 7 drives put into one module, I ran the same test again and the result was astonishingly different—an average disk read time of 5 milliseconds instead of nearly 200 milliseconds was obtained. The application throughput was improved from 18 objects/second to 80 objects/second—a factor of 4.4 improvement.

For educational purposes, the major parts of the AWR report with this SAN device reconfigured are shared below.

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
XI3100M	93415537	xi3100m	1	10.2.0.1.0	NO	XI3
	Snap Id	Snap Time	Sessions	Cursors/Session		
Begin Snap:	11	23-Oct-07 20:37:09	68	8.6		
End Snap:	12	23-Oct-07 20:50:51	68	17.4		
Elapsed:		13.70 (mins)				
DB Time:		37.94 (mins)				

28.3.1 Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	432M	432M	Std Block Size:	8K
Shared Pool Size:	128M	128M	Log Buffer:	6,968K

Load Profile

	Per Second	Per Transaction	
Redo size:	2,281,631.88	9,003.24	
Logical reads:	19,715.86	77.80	
Block changes:	6,128.00	24.18	
Physical reads:	107.20	0.42	
Physical writes:	385.19	1.52	
User calls:	4,974.31	19.63	
Parses:	2,195.75	8.66	
Hard parses:	3.58	0.01	
Sorts:	221.57	0.87	
Logons:	0.04	0.00	
Executes:	2,209.21	8.72	
Transactions:	253.42		
% Blocks changed per Read:	31.08	Recursive Call %:	10.87
Rollback per transaction %:	0.00	Rows per Sort:	4.41

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.78	Redo NoWait %:	99.94
Buffer Hit %:	100.46	In-memory Sort %:	100.00
Library Hit %:	99.67	Soft Parse %:	99.84
Execute to Parse %:	0.61	Latch Hit %:	99.52
Parse CPU to Parse Elapsed %:	85.62	% Non-Parse CPU:	89.22

Shared Pool Statistics

	Begin	End
Memory Usage %:	99.00	95.45
% SQL with executions>1:	77.37	79.51
% Memory for SQL w/exec>1:	92.84	91.96

Top Five Timed Events

Event	Waits	Time(s)	Avg Wait(ms)	% Total Call Time	Wait Class
CPU time		1,319		58.0	
log file sync	218,633	229	1	10.1	Commit
log file switch (checkpoint incomplete)	610	205	337	9.0	Configuration
direct path read	81,372	165	2	7.3	User I/O
log file parallel write	176,229	115	1	5.1	System I/O

28.3.2 Wait Events Statistics

Time Model Statistics

- Total time in database user-calls (DB Time) : 2,276.2s

Statistic Name	Time (s)	% of DB Time
DB CPU	1,319.46	57.97
sql execute elapsed time	653.48	28.71
parse time elapsed	183.16	8.05
hard parse elapsed time	17.22	0.76
PL/SQL execution elapsed time	2.78	0.12
PL/SQL compilation elapsed time	1.80	0.08
hard parse (sharing criteria) elapsed time	0.90	0.04
hard parse (bind mismatch) elapsed time	0.32	0.01
connection management call elapsed time	0.11	0.00
sequence load elapsed time	0.08	0.00
repeated bind elapsed time	0.02	0.00
DB time	2,276.17	
background elapsed time	297.10	
background cpu time	41.93	

Wait Class

Wait Class	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
User I/O	418,474	0.00	239	1	2.01
Commit	218,633	0.01	229	1	1.05
Configuration	1,235	11.50	224	182	0.01
System I/O	180,856	0.00	117	1	0.87
Concurrency	49,166	0.06	97	2	0.24
Network	3,821,805	0.00	91	0	18.35
Other	11,136	22.77	7	1	0.05
Application	58	0.00	0	0	0.00

Wait Events (Top Five)

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
log file sync	218,633	0.01	229	1	1.05
log file switch (checkpoint incomplete)	610	20.98	205	337	0.00
direct path read	81,372	0.00	165	2	0.39
log file parallel write	176,229	0.00	115	1	0.85
db file sequential read	6,322	0.00	68	11	0.03

28.3.3 IO Stats

Tablespace IO Stats

Tablespace	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	85,878	104	5.01	1.00	241,192	293	26,480	2.59
UNDOTBS1	40	0	2.00	1.00	33,573	41	9,860	0.16
SYSTEM	1,328	2	6.19	1.08	269	0	0	0.00
SYSAUX	312	0	6.79	1.40	434	1	0	0.00
USERS	39	0	2.05	1.00	39	0	0	0.00
TEMP	1	0	10.00	1.00	3	0	0	0.00

File IO Stats

Tablespace	Filename	Reads	Av Reads/s	Av Rd(ms)	Av Blks/Rd	Writes	Av Writes/s	Buffer Waits	Av Buf Wt(ms)
APP	G:\ODIAPP00.ORA	85,878	104	5.01	1.00	241,192	293	26,480	2.59
SYSAUX	G:\ODISYSAUX01.DBF	312	0	6.79	1.40	434	1	0	0.00
SYSTEM	G:\ODISYSTEM01.DBF	1,328	2	6.19	1.08	269	0	0	0.00
TEMP	G:\ODITEMP01.DBF	1	0	10.00	1.00	3	0	0	
UNDOTBS1	G:\ODIUNDOTBS01.DBF	40	0	2.00	1.00	33,573	41	9,860	0.16
USERS	G:\ODIUSERS01.DBF	39	0	2.05	1.00	39	0	0	0.00

In the next section, we summarize what we have learned from this case study so that you can apply it to your own product if a similar situation occurs.

28.4 MORAL OF THE CASE STUDY

Based on this case study, I hope it's clear that:

- The performance and scalability of an Oracle-based enterprise application depend on many factors such as the application design and implementation, Oracle tuning, and the raw power of hardware. But even with given sufficient raw power of a hardware device, how the device is configured and used could determine crucially how well the application would perform and scale. In this case, we did not immediately claim that the Apple Xserve RAID was slow. Instead, we checked how the device was configured, and after reconfiguring it, we achieved the adequate performance with the same hardware device. We could have claimed that the device was under-powered and we needed a faster device, which would be a completely wrong path, but we didn't go that way based on the software performance and scalability principles I adhere to.
- In resolving a software performance or scalability issue, one needs to take a quantitative approach, rather than a vague, qualitative approach, in order to get the issue resolved effectively and efficiently. In this case, we used a quantitative metric of the average disk read time, which was about 200 milliseconds and

5 milliseconds before and after the device was reconfigured, respectively. This way, when we said it was slow, it was indeed slow, and when we said it's fixed, it's indeed fixed, as we had solid data to support the statements.

The moral of this case study is that one should resist the temptation of requesting more powerful hardware the first moment a performance issue is identified. The raw power of a hardware device becomes an issue only when all other performance factors have been ruled out, such as software/hardware misconfigurations, inefficient SQLs, improper implementations and/or designs, and so on. If I were given the privilege to nominate and share a most compelling notion about software performance and scalability work with the greater computing community, then this is it. Thanks for your endurance to come this far, and I am sure that you have been empowered to contribute significantly to the success of your organization that performance and scalability of your product matter to your customers.

RECOMMENDED READING

The following text is recommended for a general understanding of SAN:

Marc Farley, *Storage Networking Fundamentals: An Introduction to Storage Devices, Subsystems, Applications, Management, and File Systems (Vol 1)*, 1st edn, Cisco Press, Indianapolis, 2004.

For more information about the Apple Xserve RAID, refer to the following document from Apple: http://manuals.info.apple.com/en/xserveraid_userguide.pdf

For more case studies on software performance and scalability, refer to the following text:

Henry H. Liu, *Software Performance and Scalability: A Quantitative Approach*, Wiley, Hoboken, 2009.

EXERCISES

28.1 Refer to Figure 28.1 and answer the following questions:

1. What is the purpose of having two modules with a SAN storage device?
2. What is the purpose of having an on-board memory cache?
3. Explain why a huge disk performance disparity would occur if the 7 drives were split and inserted into two modules to construct a RAID.

28.2 Refer to the two AWR reports presented in this case study and answer the following questions:

1. In the first AWR report, the Wait Class statistics section shows that there were over 3 million network waits versus 418 thousand disk I/Os. Why was disk I/O a much more severe bottleneck than the network wait events? Could you claim that network was a bottleneck as well?

2. Did reconfiguring the SAN storage reduce the numbers of network waits and disk I/O waits?
 3. Without using the IO statistics section that provides average disk read times, how could you identify the SAN storage device as the bottleneck based on the statistics contained in the other sections of the first AWR report?
- 28.3** Compare the two AWR reports and see how the top five timed events and top five wait events had changed after the SAN device was reconfigured.
- 28.4** Explain the following I/O related metrics based on the two AWR reports presented in this case study:
1. Direct path read
 2. Log file sync
 3. DB file sequential read
 4. Log file switch
 5. DB file parallel write

APPENDIX A

Oracle Product Documentations

The will . . . is the driving force of the mind. If it's injured, the mind falls to pieces.
—August Strindberg, *The Father*

The most comprehensive and authoritative texts about Oracle are Oracle's own product documentations accompanying each release of Oracle. Specifically, the following Oracle documents are helpful for understanding Oracle performance and scalability features covered in this book.

A.1 ORACLE DATABASE CONCEPTS

Oracle Corp, *Oracle Database Concepts*, 11g Release 1 (11.1) B28318-05 (556 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm

A.2 ORACLE DATABASE ADMINISTRATOR'S GUIDE

Oracle Corp, *Oracle Database Administrator's Guide*, 11g Release 1 (11.1) B28310-04 (882 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf.

Oracle Database Performance and Scalability: A Quantitative Approach, First Edition. Henry H. Liu.
© 2012 John Wiley & Sons, Inc. Published 2012 by John Wiley & Sons, Inc.

A.3 ORACLE DATABASE REFERENCE

Oracle Corp, *Oracle Database Reference*, 11g Release 1 (11.1) B28320-03 (1132 pages), April 2009, available free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28320.pdf. This document covers all Initialization Parameters (Part I), all Static Data Dictionary Views (Part II), all Dynamic Performance Views (Part III), and Descriptions of all Wait Events (Appendix C). This is the place you need to go if you want to know more about a specific item, be it an initialization parameter, a static data dictionary view, a dynamic performance view, or a wait event.

A.4 ORACLE DATABASE PERFORMANCE TUNING GUIDE

Oracle Corp, *Oracle Database Performance Tuning Guide*, 11g Release 2 (11.2) E10821-05 (532 pages), February 2010, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10821.pdf. This document covers: Performance Tuning (Part I), Performance Planning (Part II), Optimizing Instance Performance (Part III), and Optimizing SQL Statements (Part IV). This is the document to study if you want to know all about Oracle performance optimization and tuning methodologies as well as all the tunable knobs.

A.5 ORACLE DATABASE 2 DAY + PERFORMANCE TUNING GUIDE

Oracle Corp, *Oracle Database 2 Day + Performance Tuning Guide*, 11g Release 2 (11.2) E10822-02 (168 pages), September 2009, available for free online at: http://download.oracle.com/docs/cd/E11882_01/server.112/e10822.pdf. This document covers: Getting Started (Part I), Proactive Database Tuning (Part II), Reactive Database Tuning (Part III), and SQL Tuning (Part IV). This is a valuable supplement to document A.4 above.

A.6 ORACLE DATABASE 2 DAY DBA

Oracle Corp, *Oracle Database 2 Day DBA*, 11g Release 1 (11.1) 28301-03 (272 pages), March 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28301.pdf. This document covers how to install, configure, manage, administrate, and monitor an Oracle Server.

A.7 ORACLE DATABASE SQL LANGUAGE REFERENCE

Oracle Corp, *Oracle Database SQL Language Reference*, 11g Release 1 (11.1) B28286-05 (1446 pages), September 2008, available for free online at:

http://download.oracle.com/docs/cd/B28359_01/server.111/b28286.pdf. This document is a complete reference of Oracle SQL.

A.8 ORACLE DATABASE SAMPLE SCHEMAS

Oracle Corp, *Oracle Database Sample Schemas*, 11g Release 1 (11.1) B28328-03 (48 pages), July 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/server.111/b28328.pdf. This document covers all Oracle 11g sample schemas.

A.9 ORACLE DATABASE PL/SQL PACKAGES AND TYPES REFERENCE

Oracle Corp, *Oracle Database PL/SQL Packages and Types Reference*, 11g Release 1 (11.1) B28419-03 (5100 pages), April 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28419.pdf. This document covers all Oracle 11g PL/SQL packages such as those for gathering CBO statistics and types references.

A.10 ORACLE DATABASE PL/SQL LANGUAGE REFERENCE

Oracle Corp, *Oracle Database PL/SQL Language Reference*, 11g Release 1 (11.1) B28359-01 (712 pages), August 2009, available for free online at: http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28370/toc.htm.

A.11 ORACLE DATABASE JDBC DEVELOPER'S GUIDE AND REFERENCES

Oracle Corp, *Oracle Database JDBC Developer's Guide and References*, 11g Release 1(11.1) B31224-04 (508 pages), July 2008, available for free online at: http://download.oracle.com/docs/cd/B28359_01/java.111/b31224.pdf.

APPENDIX B

*Using SQL*Plus with Oracle*

Music is the poetry of the air.

—Richter

This appendix offers a brief summary about how to use SQL*Plus with Oracle to perform common tasks related to your performance and scalability work. Although this is not a complete coverage of SQL*Plus, it should be sufficient for most of the common tasks you conduct as I have been for many years. Also in this appendix, it is assumed that the OS is Microsoft Windows, for example, Windows 2003, or Windows XP and above.

B.1 INSTALLATION

If you intend to use SQL*Plus from the client perspective, for example, on your desktop or laptop to access an Oracle Server installed remotely, then you need to install the proper version of the Oracle client software on your system. After installation, your PATH environment variable is set to point to the bin directory of your install. For example, if your client install directory is `c:\ora10gc`, then `c:\ora10gc\bin` will be on your PATH, which can be verified easily at a MS-DOS command prompt by typing `echo %PATH%`. Keep in mind that this is where you will invoke your `sqlplus.exe` program.

Oracle Database Performance and Scalability: A Quantitative Approach, First Edition. Henry H. Liu.
© 2012 John Wiley & Sons, Inc. Published 2012 by John Wiley & Sons, Inc.

If you are on your Oracle Server system with a specific version of Oracle installed (assuming it's Oracle 11g here for convenience), then the `sqlplus.exe` program is already available from `<server_install_dir>\product\11.1.0\db_1\bin` directory. If you don't have an Oracle client installed on your Oracle Server system, then when you type `sqlplus` at an MS-DOS command prompt, your `sqlplus.exe` program will be invoked from the above directory, and there is no ambiguity here.

However, when you are on a system that has both Oracle client and Server or multiple versions of Oracle client software installed, you need to check from where the `sqlplus.exe` is invoked if you don't precede it with an exact path. I typically just add the exact path to the `sqlplus.exe` program so that there is no ambiguity with it.

The next thing to make sure is to match your `sqlplus.exe` program with a proper `tnsname.ora` file, as is briefed in the next section.

B.2 SQL*PLUS AND TNSNAMES.ORA FILE

To connect to an Oracle Server with the `sqlplus.exe` program, you need a matching `tnsnames.ora` file with a proper entry in it for your Oracle Server. The `tnsnames.ora` file is located in the directory of `<client_install_dir>\NETWORK\ADMIN` on the client machine or `<server_install_dir>\product\...\db_1\NETWORK\ADMIN` on the server machine where the "...\" part depends on the version of the Oracle server.

After locating the `tnsnames.ora` file, make sure an entry similar to the following exists in it or create one if it doesn't:

```
<yourConnectionString> =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP) (HOST = <yourHost>) (PORT =
1521))
  )
  (CONNECT_DATA =
    (SID = <yourSID>)
    (SERVER = DEDICATED)
  )
)
```

Note that you need to replace the above three environment-specific entries with your own. I usually use the `SID_hostName` for the `<yourConnectionString>` entry at the beginning of the above sample. The entry `SID = <yourSID>` can also be replaced with `SERVICE_NAME = <yourServiceName>`. Then at an MS-DOS prompt, issue a command of:

```
tnsping <yourConnectionString>
```

And if you get a reply ended with “OK <xx msec>,” you are all set now for using `sqlplus.exe` with your Oracle Server. If not, you need to resolve the issue first, because that means that your Oracle Server is not reachable.

Next, let me help you get familiar with the basics of SQL*Plus.

B.3 BASICS OF SQL*PLUS

SQL*Plus has three parts:

- You can execute SQL*Plus commands which may or may not have anything to do with Oracle or SQL.
- SQL*Plus enables us to execute Oracle SQL statements to query or manipulate the database. This is the major purpose why we use it.
- SQL*Plus enables a user to execute PL/SQL blocks of code to do more complicated things. The difference between SQL and PL/SQL is that SQL is a query language while PL/SQL is a procedural language.

To connect to your Oracle database, the command syntax is, assuming that you have the proper credential and the SID information:

```
sqlplus <username>/<password>@<yourConnectionString>
```

Most of the time, I use the `system` user for all installs and accounts with a password that I can remember.

Next, let's review some commonly used SQL*Plus commands.

B.4 COMMON SQL*PLUS COMMANDS

First, remember that a SQL*Plus command doesn't need to end with a semicolon (“;”).

For convenience, we use two conventions in presenting the SQL*Plus examples: (1) if you encounter [...], that means the part in [...] is optional, and (2) if you encounter {alb}, that means select one in {alb}, namely, either a or b.

The following list shows some of the most commonly used SQL*Plus commands with the comments beginning with //(do not type this part in your exercise):

- SQL>SET AUTOCOMMIT {ON|OFF} // turn on or off auto-commit for the SQLs to be executed in this session. This feature is off by default.
- SQL>SHOW AUTOCOMMIT
- SQL>DESC[RIBE] <your_table> // query all the columns of your table
- SQL>HOST <host commands> // execute a command on the host

- `SQL>SET LINESIZE 500` // set line size to 500 characters. The default is 80, which may break a line of your query result into multiple lines.
- `SQL>SET PAGESIZE 0` // set page size to unlimited lines per page. This setting can help avoid listing the results section by section with repeated titles.
- `SQL>CLEAR SCREEN`
- `SQL>COLUMN C1 FORMAT A40` // set column width to 40 characters for column C1
- `SQL>@myScript.sql` // execute this sql script file
- `SQL>SPOOL myOutput.txt` // spool the output into this file
- `SQL>SET TERM[OUT] {OFFION}` // control whether to display output
- `SQL>SPOOL OFF` // end spooling

In the next section, we will illustrate how to use SQL*Plus to execute SQL statements.

B.5 USING SQL*PLUS TO EXECUTE SQL STATEMENTS

There are two ways to execute SQL statements with SQL*Plus: enter SQL statements at the `SQL>` command line prompt directly, or put all your SQL statements into one file and execute them like a batch file. Let's demonstrate how these two different approaches work out using examples.

To execute your SQL statements interactively, connect to your Oracle database with the user who has proper privileges. Then enter your SQL statements directly at the command line prompt. Note that if you want to have your SQLs like `DELETE`, `UPDATE`, `INSERT`, and so on, committed, either execute `COMMIT` directly or turn `AUTOCOMMIT` on as described in the previous section. The other scenario is that you want to spool your query results into a text file. In this case, use the `SPOOL` command to create a text file for your query output. You may also want to turn display off using the `SET TERM OFF` command described in the previous section. It's also a good idea to set line size and page size to very large values so that you don't have lines wrapped and also page separation lines interrupting your output which may be processed with `EXCEL™` or other programs offline. Such details are left for you to deal with based on your needs with the help of the SQL*Plus commands introduced in the previous section.

Assuming that you have the `SCOTT` sample schema created on your Oracle Server, you can try out the following commands (use your connect string):

- `C:\myOracle\product\11.1.0\db_1\sqlplus
scott/tiger@orcl`
- `SQL>SELECT TABLE_NAME FROM USER_TABLES;`
- `SQL>DESC DEPT`
- `SQL>SELECT DEPTNO || '\, ' || DNAME || '\, ' || LOC FROM DEPT;`
- `SQL>HOST dir myScript.sql`

- SQL>@myScript
-

The last command executes the `myScript.sql` script, which could contain many commands such as the mixed SQL*Plus commands and SQL commands shown above. Note that SQL*Plus expects the command file has a `.sql` extension. Otherwise it will not work.

The purpose of this section is to illustrate how to execute SQL statements using SQL*Plus, not how to programming in SQL. The next section illustrates how to execute a PL/SQL block.

B.6 USING SQL*PLUS TO EXECUTE PL/SQL BLOCKS

PL/SQL stands for Procedural Language (PL)/Structured Query Language (SQL). It's one of the three languages that Oracle supports, together with SQL and Java. If you happen to be familiar with the programming language Ada, then it should be easier for you to learn PL/SQL, because PL/SQL resembles Ada in syntax. But once again, this section is not about how to program in PL/SQL. Instead, we merely illustrates how to execute PL/SQL under SQL*Plus.

The following excerpt shows how to execute a PL/SQL block interactively under SQL*Plus. To execute it, log into your SQL*Plus, and type all the lines one by one verbatim. Then you should see the “Hello World!” output. You can also put the following lines into a file, say `myPLSQL.sql`, and execute it using the `@myPLSQL` SQL*Plus command.

```
SQL>SET SERVEROUT [PUT] ON
SQL> BEGIN
  dbms_output.put_line ('Hello World!');
END;
/
```

In the next few sections, I'll provide some commands as a handy reference for taking care of your common tasks related to your Oracle-based application performance and scalability testing work. This does not serve as a complete list. Instead, it represents a minimum set of tasks that I conduct on a daily basis as well, so it might help save you some time from searching other texts.

B.7 USING SQL*PLUS AUTOTRACE TO OBTAIN EXECUTION PLANS AND OPTIMIZER STATISTICS

If you are interested in the EXPLAIN PLAN and the associated optimizer statistics for a known DML SQL statement, such as SELECT, DELETE, UPDATE, or INSERT, the easiest way is through the SQL*Plus tool. This feature is controlled by the settings of a

variable named AUTOTRACE. You can issue a command of SET AUTOTRACE <option> at a SQL*Plus command prompt to enable this auto trace feature by choosing a proper value for <option> to suit your needs. Valid commands with various options for this feature are listed as follows:

<option>	Output
OFF	Disables the autotrace feature. This is the default.
ON	Print SQL result, EXPLAIN PLAN, and statistics
ON STATISTICS	Print SQL result and statistics w/o printing EXPLAIN PLAN
TRACEONLY	Print EXPLAIN PLAN and statistics w/o printing the result

To use this feature, the user with which you log into SQL*Plus must have the PLUSTRACE role granted. If you use the built-in system account, you already have this privilege. If you use a different user without this role granted, use a DBA account to grant this role to your preferred user by executing the command GRANT PLUSTRACE <your_user>.

You also need to have the PLAN_TABLE created with the user with which you login. Once again, if you use the built-in system account, this table has already been created. If this table does not exist for the user with which you login, you can create it using the command @%ORACLE_HOME%/RDBMS/ADMIN/UTLXPLAN.SQL at a SQL*Plus command prompt.

B.8 USING SQL*PLUS TIMING COMMAND

By executing the SET TIMING command at a SQL*Plus prompt, you can measure the elapsed time of the execution of a SQL statement. This command can also be used in conjunction with the autotrace feature described in the preceding section. However, consider the following settings that may affect your measured elapsed times of a SQL statement:

- *SET APPINFO OFF*. This command eliminates the overhead associated with registering the script, and thus making your SQL timing more accurate.
- *SET ARRAYSIZE <n>*. This setting specifies the array size or the number of rows that SQL*Plus will fetch from the database at one time. The valid values are from 1 to 5000. The default value is n=15. A large value increases the throughput but requires more memory. According to Oracle's documentation, values over approximately 100 provide little added performance, which is consistent with my long-time observations as well.

- *SET DEFINE OFF*. This setting controls whether SQL*Plus parses scripts for substitution variables. If it is set to OFF, SQL*Plus does not parse scripts for substitution variables. If your script does not use substitution variables, setting DEFINE OFF may make your SQL run faster.
- *SET FLUSH OFF*. This setting controls when output is sent to the user's display device. The setting of OFF allows the host operating system to buffer output which may make your SQL run faster by reducing the amount of program input and output. Use OFF if you run a script that does not require user input and you do not need to see the output until the script finishes running.
- *SET SERVER OUTPUT OFF*. This setting controls whether SQL*Plus checks for and displays DBMS output. If it is set to OFF, SQL*Plus does not check for DBMS output and does not display output after applicable SQL or PL/SQL statements. Suppressing output checking and display may make your SQL run faster.
- *SET TRIMOUT ON*. This setting determines whether SQL*Plus allows trailing blanks at the end of each displayed line. The setting of ON removes blanks at the end of each line, which may make your SQL run faster, especially when you run SQL*Plus remotely against your Oracle Server. To measure the true performance of a SQL statement, you should time it on the database server or you need to take the network latency into account with your timing result when you run SQL*Plus remotely against your Oracle Server. Note that TRIMOUT ON does not affect spooled output.
- *SET TRIMSPOOL ON*. This setting determines whether SQL*Plus allows trailing blanks at the end of each spooled line. The setting of ON removes blanks at the end of each line, which may make your SQL run faster, especially when you run SQL*Plus remotely against your Oracle Server and the output is massive.

B.9 EXPORTING/IMPORTING ORACLE DATABASES WITH SQL*PLUS

Although one can use the DB console to perform export/import tasks, it's a lot more straightforward to perform such tasks using the expdp/impdp utilities at a command line prompt on an Oracle database Server. This section provides a complete procedure for performing such tasks using the expdp and impdp utilities.

Although there is a default directory for exported files, you might consider creating your own directory, which will not be wiped out if your Oracle Server is uninstalled at some point later. If you decide to do so, use the following command to create a dump directory outside your Oracle installation (for example, at C:\oraExports), with read and write permissions on your directory granted to your Oracle Schema owner as well:

```
cmd>sqlplus sys/yourPassword@yourConnectionString as sysdba
SQL>create or replace directory mydump_dir as 'c:/oraExports/';
SQL>grant read, write on directory mydump_dir to SYSTEM,
<yourSchemaOwner>;
SQL>quit
```

Note that the directory you specify must exist *a priori*. Otherwise, you will get errors when performing export tasks. Also note that Oracle does not check whether it's a valid directory path.

To export a full database, use the following command at the OS command line prompt without logging into your Oracle database:

```
expdp system/<password>@<yourConnectionString> full=y
directory=mydump_dir dumpfile=my_expdp.dmp
```

Note that you can optionally add `logfile=mylog.log` to the above command. In addition, if you don't specify a directory, the exported file will be put into the default directory of `%ORACLE_HOME%/admin/dpdump`. You can query at a SQL command prompt the `DBA_DIRECTORIES` or `ALL_DIRECTORIES` table for the owner, directory_name, and directory_path of all directories in your Oracle database.

To import a full database from an export file generated with the above command, execute the following command:

```
impdp system/password@<yourConnectionString> full=y
directory=mydump_dir dumpfile=my_expdp.dmp
```

To export/import an Oracle schema only without exporting/importing the entire database, make two changes to the above two commands: (1) use your Schema owner in place of the `system` account, and (2) replace “full=y” in the above commands with “Schemas=yourSchema” where `yourSchema` is the schema you want to export/import. Also you may need to execute the SQL statement of `drop user yourUserName cascade` and add your dropped user back with proper attributes before importing your previously exported schema.

To export/import tables of a schema only without exporting/importing the entire schema, the procedure is the same as exporting/importing a schema except that you need to replace the part of “Schemas=yourSchema” in the schema export/import commands with “tables=table1,table2,...” where `table1, table2, ...` is a list of tables of the schema you want to export/import.

To speed up export/import, you can add `parallel=n` where `n` is the number of threads for an export/import job. Exporting/importing a large database may take many hours or even longer. So be cautious with it as Oracle may not allow you to stop it as soon as it's initiated.

B.10 CREATING AWR REPORTS WITH SQL*PLUS

Starting from Oracle 10g, a new feature named Automatic Workload Repository (AWR) has been introduced for reporting various Oracle database metrics on an Oracle system. This is a very powerful and easy-to-learn tool for everyone who is concerned with whether an Oracle Server is running optimally with a given

workload. Throughout this text, many case studies on diagnosing and resolving real world Oracle performance and scalability issues have been presented based on the AWR feature. In this section, we give a brief overview of how to create an AWR report.

An AWR report is created using the script `awrrpt.sql` located in the directory of `%ORACLE_HOME%/rdbms/admin`. You can create an AWR report directly on the Oracle server or remotely on a client system that has an Oracle 10g (or above) client installed.

An AWR report is created based on the snapshots taken at various points of time during a period that Oracle has been running without being interrupted by shutdown/restart operations. A snapshot records all database activities at the time when the snapshot was taken. By default, Oracle takes snapshots hourly, but you can change this schedule. Sometimes, when your test lasts less than an hour or you want to have more accurate begin and end timestamps of your test instead of the integer hour times scheduled by Oracle, you can manually create a snapshot at a SQL*Plus prompt with the following command:

```
SQL>execute dbms_workload_repository.create_snapshot
```

Note that whether on a local or a remote system, it's a good idea that you always change to the same directory to create AWR reports so that you can keep all your AWR reports in the same place.

Now you can connect to your Oracle Server and create your AWR report as follows:

```
%ORACLE_HOME%/bin/sqlplus system/<yourPassword>@<yourSID>
SQL>@?/rdbms/admin/awrrpt
- Hit return to select HTML for report-type
- Enter the number of days that covers your snapshots
- Select a begin_snap_id based on the timestamp displayed
- Select an end_snap_id based on the timestamp displayed
- Enter the report name, e.g., myReport.html
```

Note that you need to add the `“.html”` extension explicitly; otherwise, the report will not be in the html format. Then you can find the AWR report generated in the directory you started SQL*Plus.

Refer to Chapter 11 and several other case study chapters for how to read and analyze an AWR report.

B.11 CHECKING TABLESPACE USAGE WITH SQL*PLUS

Very often, you may not realize that some of your Oracle tablespaces have reached 100% full until some abnormal symptoms begin to show up from your application. This is especially true when you do not have automatic data file extension set up.

In this section, a SQL script is provided to illustrate how one can check the usage of each tablespace on an Oracle Server with SQL*Plus. The script below shows such attributes for a tablespace as the total size, used space in MB and used space in percentage:

```
SELECT df.tablespace_name "Tablespace",
       fs.bytes / (1024 * 1024) "Size (MB)",
       df.bytes_used / (1024 * 1024) "Used (MB)",
       Round(df.bytes_used * 100 / fs.bytes) "% Used"
FROM dba_temp_files fs,
     (SELECT tablespace_name, bytes_used
      FROM v$temp_space_header
      GROUP BY tablespace_name, bytes_used) df
WHERE fs.tablespace_name (+) = df.tablespace_name
UNION ALL
SELECT df.tablespace_name,
       df.bytes / (1024 * 1024),
       (df.bytes - SUM(fs.bytes)) / (1024 * 1024),
       Round((df.bytes - SUM(fs.bytes)) * 100 / df.bytes)
FROM dba_free_space fs,
     (SELECT tablespace_name, SUM(bytes) bytes
      FROM dba_data_files
      GROUP BY tablespace_name) df
WHERE fs.tablespace_name (+) = df.tablespace_name
GROUP BY df.tablespace_name, df.bytes ORDER BY 1 ASC;
```

The above script, named `check_ts_used_space.sql`, was executed against an Oracle 10g database using the follow commands:

```
SQL> set pagesize 50
SQL> @c:\tests\check_ts_used_space
```

If a tablespace is becoming full and automatic data file management is not enabled, one should add additional storage to keep the application running normally. One can first query the existing data files for the tablespace, and then extend the existing data file or add a new data file. To add a new data file, execute the following command with the proper entries filled based on your situation. (Note: this example adds 20 GB. Change this setting based on your needs.):

```
ALTER TABLESPACE "yourTablespaceName"
ADD
    DATAFILE 'YourDataFilePath\yourDataFileName.DBF'
    SIZE 20000M
```

You can also add more attributes such as `autoextend`, or use the DB console to avoid looking up more complex syntax for this task.

B.12 CREATING EM DBCONSOLE WITH SQL*PLUS

If you created your database without having EM DBConsole created, you can create it by following the procedure below:

- Set the `ORACLE_SID` environment variable to the SID of your Oracle database using the command `set ORACLE_SID=<your_SID>`.
- Execute the command `%ORACLE_HOME%\bin\emca -repos create` at a command prompt. Note that `%ORACLE_HOME%` represents your `ORACLE_HOME` environment variable. If this environment variable is not set, you can manually change to that directory using the `cd` command. Enter proper information about your SID, listener port, and passwords for SYS and SYSMAN users to complete this step.
- Create your DBConsole with the command `emca -config dbcontrol db` at a command prompt. Enter proper information about your SID, listener port, and passwords for SYS and SYSMAN users to complete this step. At the end, you will be informed whether it's completed successfully. If successful, there should be a line stating the URL for your DB console. If not, you might want to drop the DB console repository and repeat the same steps described above.

To drop and recreate your EM DBConsole, follow the procedure below:

- To drop your EM DBConsole repository, execute the following command with proper information about your SID, listener port, and passwords for SYS and SYSMAN users entered:
 - `emca -deconfig dbconsole db -repos drop`
- To recreate your repository, execute the following command:
 - `emca -config dbcontrol db -repos recreate`

On Windows, after a successful installation of EM DBConsole, you should see a service named `OracleDBConsole<your_SID>` from your Services management snap-in. To troubleshoot any issues, use the following manual commands:

```
cmd>%ORACLE_HOME%\bin\emctl status dbconsole
cmd>%ORACLE_HOME%\bin\emctl start dbconsole
cmd>%ORACLE_HOME%\bin\emctl stop dbconsole
```

To change from the secure protocol `https` to regular `http`, execute the following command:

```
%ORACLE_HOME%/bin/emctl unsecure dbconsole
```

I once got the following error after logging into the EM DBConsole with the HTTPS protocol

```
java.util.MissingResourceException: Can't find resource  
for bundle oracle.sysman.
```

I dropped and recreated the repository, but still got the same error. After changing from HTTPS to HTTP protocol using the above command, the error disappeared.

APPENDIX C

A Complete List of All Wait Events in Oracle 11g

A great artist is always before his time or behind it.
—George Moore

This appendix lists all wait events available in Oracle 11g. The list was obtained with the following query:

```
SQL> SELECT wait_class, name FROM v$event_name ORDER BY  
wait_class ASC;
```

The purpose of including this list here is to help you make a quick association between a wait event and its category as implied by its class name, thus providing a context on why that wait event transpired with your application. You can also use this comprehensive list of wait events to test your proficiency in Oracle: if you randomly pick a few wait events and you know clearly what they are about, then you are already an above-average Oracle professional.

Member Wait Events Contained in Each Wait Class
-----**Administrative**

JS kill job wait
JS coord start wait
Backup: sbtinit
Backup: sbtopen
Backup: sbtread
Backup: sbtwrite
Backup: sbtclose
Backup: sbtinfo
Backup: sbtremove
Backup: sbtbackup
Backup: sbtclose2
Backup: sbtcommand
Backup: sbtend
Backup: sbterror
Backup: sbtinfo2
Backup: sbtinit2
Backup: sbtread2
Backup: sbtremove2
Backup: sbtrestore
Backup: sbtwrite2
Backup: sbtpcbbackup
Backup: sbtpccancel
Backup: sbtpccommit
Backup: sbtpcend
Backup: sbtpcquerybackup
Backup: sbtpcqueryrestore
Backup: sbtpcrestore
Backup: sbtpcstart
Backup: sbtpcstatus
Backup: sbtpcvalidate
Backup: sbtgetbuf
Backup: sbtrelbuf
Backup: sbtmapbuf
Backup: sbtbufinfo
multiple dbwriter suspend/resume for file offline
buffer pool resize
switch logfile command
wait for possible quiesce finish
switch undo - offline
alter rbs offline
enq: TW - contention
index (re)build online start

index (re)build online cleanup
 index (re)build online merge
 alter system set dispatcher
 connection pool wait
 enq: DB - contention
 enq: ZG - contention
 ASM COD rollback operation completion
 ASM mount : wait for heartbeat
 JS kgl get object wait

Application

enq: PW - flush prewarm buffers
 WCR: replay lock order
 enq: RO - fast object reuse
 enq: KO - fast object checkpoint
 enq: TM - contention
 enq: TX - row lock contention
 Wait for Table Lock
 enq: RC - Result Cache: Contention
 Streams capture: filter callback waiting for ruleset
 Streams: apply reader waiting for DDL to apply
 SQL*Net break/reset to client
 SQL*Net break/reset to dblink
 enq: UL - contention
 OLAP DML Sleep
 enq: RO - contention

Cluster

ASM PST query : wait for [PM][grp][0] grant
 gc claim
 retry contact SCN lock master
 gc buffer busy acquire
 gc buffer busy release
 pi renounce write complete
 gc current request
 gc cr request
 gc cr disk request
 gc cr multi block request
 gc current multi block request
 gc block recovery request
 gc cr block 2-way
 gc cr block 3-way
 gc cr block busy
 gc cr block congested
 gc cr failure

gc cr block lost
gc cr block unknown
gc current block 2-way
gc current block 3-way
gc current block busy
gc current block congested
gc current retry
gc current block lost
gc current split
gc current block unknown
gc cr grant 2-way
gc cr grant busy
gc cr grant congested
gc cr grant unknown
gc cr disk read
gc current grant 2-way
gc current grant busy
gc current grant congested
gc current grant unknown
gc freelist
gc remaster
gc quiesce
gc object scan
gc current cancel
gc cr cancel
gc assume
gc domain validation
gc recovery free
gc recovery quiesce
lock remastering

Commit

log file sync
enq: BB - 2PC across RAC instances

Concurrency

latch: MQL Tracking Latch
enq: WG - lock fso
latch: row cache objects
row cache lock
row cache read
cursor: mutex X
cursor: mutex S
cursor: pin S wait on X
latch: shared pool

```

library cache pin
library cache lock
library cache load lock
library cache: mutex X
library cache: mutex S
resmgr:internal state change
resmgr:internal state cleanup
resmgr:sessions to exit
pipe put
logout restrictor
os thread startup
Shared IO Pool Memory
latch: cache buffers chains
buffer busy waits
latch: In memory undo latch
enq: TX - index contention
latch: Undo Hint Latch

```

Configuration

```

enq: TX - allocate ITL entry
statement suspended, wait error to be cleared
enq: HW - contention
enq: SS - contention
sort segment request
enq: SQ - contention
Global transaction acquire instance locks
wait for EMON to process ntfns
free buffer waits
checkpoint completed
write complete waits
latch: redo writing
latch: redo copy
log buffer space
log file switch (checkpoint incomplete)
log file switch (private strand flush incomplete)
log file switch (archiving needed)
log file switch completion
enq: ST - contention
undo segment extension
undo segment tx slot

```

Idle

```

PL/SQL lock timer
(other 79 wait events in this wait class are omitted here
...)

```

Network

remote db operation
TEXT: URL_DATASTORE network wait
remote db file write
ARCH wait for netserver start
ARCH wait for netserver init 1
ARCH wait for netserver init 2
ARCH wait for flow-control
ARCH wait for netserver detach
ARCH wait for net re-connect
LNS wait on ATTACH
LNS wait on SENDREQ
LNS wait on DETACH
LNS wait on LGWR
LGWR wait on ATTACH
LGWR wait on SENDREQ
LGWR wait on DETACH
LGWR wait on LNS
ARCH wait on ATTACH
ARCH wait on SENDREQ
ARCH wait on DETACH
TCP Socket (KGAS)
dispatcher listen timer
dedicated server timer
SQL*Net message to client
SQL*Net message to dblink
SQL*Net more data to client
SQL*Net more data to dblink
SQL*Net more data from client
SQL*Net message from dblink
SQL*Net more data from dblink
SQL*Net vector data to client
SQL*Net vector data from client
SQL*Net vector data to dblink
SQL*Net vector data from dblink
remote db file read

Other

null event
(other 629 wait events in this wait class are omitted here
...)

Queueing

Streams capture: resolve low memory condition
Streams AQ: enqueue blocked on low memory

Streams AQ: enqueue blocked due to flow control
 Streams capture: waiting for subscribers to catch up

Scheduler

resmgr:cpu quantum
 resmgr:I/O prioritization
 resmgr:become active

System I/O

control file single write
 control file parallel write
 control file sequential read
 io done
 Network file transfer
 Standby redo I/O
 RMAN backup & recovery I/O
 Log archive I/O
 ksfd: async disk IO
 ARCH sequential i/o
 db file parallel write
 log file parallel write
 log file single write
 log file sequential read
 ARCH random i/o
 kfk: async disk IO
 RFS write
 RFS random i/o
 RFS sequential i/o
 LGWR random i/o
 LGWR sequential i/o
 LNS ASYNC control file txn
 recovery read

User I/O

db file sequential read
 buffer read retry
 db file single write
 Datapump dump file I/O
 db file parallel read
 direct path read
 direct path read temp
 direct path write
 dbms_file_transfer I/O
 DG Broker configuration file I/O
 Data file init write

dbverify reads
Log file init write
Shared IO Pool IO Completion
local write wait
BFILE read
securefile direct-write completion
securefile direct-read completion

User I/O

Intelligent Storage OSS I/O completion
direct path write temp
read by other session
db file scattered read

APPENDIX D

A Complete List of All Metrics with the V\$STATNAME View

*If a man will begin with certainties, he shall end in doubts; but if he will be content to begin
with doubts, he shall end in certainty.*
—Francis Bacon

This appendix lists all metrics stored in the V\$STATNAME view in Oracle 11g. The purpose of providing this complete list is to help you recognize the relevant metrics for troubleshooting various Oracle performance and scalability issues. The meaning of each metric is self-explanatory by its name. You can infer the context of a metric by its class as represented in a number as shown below:

- 1—User
- 2—Redo
- 4—Enqueue
- 8—Cache
- 16—OS
- 32—Real Application Cluster

64—SQL

128—Debug

An interesting fact is that these class numbers are additive. For example, by adding 8 and 64, a new class number of 72 is derived, which represents a new class of SQL cache.

A quiz here: If you randomly pick a few items from the list below and you know clearly what they are about, then your knowledge about Oracle performance and scalability is above average.

The SQL query executed against the V\$STATNAME view and the corresponding output are displayed as follows:

```
SQL> SELECT class, name from V$STATNAME ORDER BY class ASC;
```

```
CLASS NAME
```

```
-----
1 SQL*Net roundtrips to/from dblink
1 bytes via SQL*Net vector to client
1 bytes via SQL*Net vector from client
1 bytes via SQL*Net vector to dblink
1 bytes via SQL*Net vector from dblink
1 Workload Capture: size (in bytes) of recording
1 Workload Capture: dbtime
1 Workload Capture: user calls
1 Workload Capture: user calls flushed
1 Workload Capture: unreplayable user calls
1 Workload Capture: user txns
1 Workload Capture: user logins
1 Workload Capture: unsupported user calls
1 Workload Capture: errors
1 Workload Replay: dbtime
1 Workload Replay: network time
1 Workload Replay: think time
1 Workload Replay: time gain
1 Workload Replay: time loss
1 Workload Replay: user calls
1 Workload Replay: deadlocks resolved
1 logons cumulative
1 logons current
1 opened cursors cumulative
1 opened cursors current
1 user commits
1 user rollbacks
1 user calls
1 recursive calls
1 recursive cpu usage
1 session logical reads
```


1 session stored procedure space
1 CPU used by this session
1 DB time
1 cluster wait time
1 concurrency wait time
1 application wait time
1 user I/O wait time
1 session connect time
1 session uga memory
1 session uga memory max
1 session pga memory
1 session pga memory max
1 serializable aborts
1 commit batch/immediate requested
1 commit batch requested
1 commit immediate requested
1 commit batch/immediate performed
1 commit batch performed
1 commit immediate performed
1 commit wait/nowait requested
1 commit nowait requested
1 commit wait requested
1 commit wait/nowait performed
1 commit nowait performed
1 commit wait performed
1 java call heap total size
1 java call heap total size max
1 java call heap used size
1 java call heap used size max
1 java call heap live size
1 java call heap live size max
1 java call heap object count
1 java call heap object count max
1 java call heap live object count
1 java call heap live object count max
1 java call heap gc count
1 java call heap collected count
1 java call heap collected bytes
1 java session heap used size
1 java session heap used size max
1 java session heap live size
1 java session heap live size max
1 java session heap object count
1 java session heap object count max
1 java session heap live object count
1 java session heap live object count max
1 java session heap gc count
1 java session heap collected count
1 java session heap collected bytes

1 bytes sent via SQL*Net to client
1 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
1 bytes sent via SQL*Net to dblink
1 bytes received via SQL*Net from dblink
2 redo blocks read for recovery
2 redo entries
2 redo size
2 redo entries for lost write detection
2 redo size for lost write detection
2 redo buffer allocation retries
2 redo wastage
2 redo writer latching time
2 redo writes
2 redo blocks written
2 redo blocks written for direct writes
2 redo write time
2 redo blocks checksummed by FG (exclusive)
2 redo blocks checksummed by LGWR
2 redo log space requests
2 redo log space wait time
2 redo ordering marks
2 redo subscn max counts
2 redo blocks read total
2 redo blocks read (memory)
2 redo blocks read total by LNS
2 redo blocks read (memory) by LNS
2 flashback log writes
4 enqueue timeouts
4 enqueue waits
4 enqueue deadlocks
4 enqueue requests
4 enqueue conversions
4 enqueue releases
8 commit cleanout failures: block lost
8 commit cleanout failures: cannot pin
8 commit cleanout failures: hot backup in progress
8 commit cleanout failures: buffer being written
8 commit cleanout failures: callback failure
8 commit cleanouts
8 commit cleanouts successfully completed
8 recovery array reads
8 recovery array read time
8 CR blocks created
8 current blocks converted for CR
8 switch current to new buffer
8 write clones created in foreground
8 write clones created in background
8 write clones created for recovery

8 recovery block gets from cache
8 physical reads cache prefetch
8 physical reads prefetch warmup
8 prefetched blocks aged out before use
8 prefetch warmup blocks aged out before use
8 prefetch warmup blocks flushed out before use
8 physical reads retry corrupt
8 physical reads direct (lob)
8 physical writes direct (lob)
8 cold recycle reads
8 shared hash latch upgrades - no wait
8 shared hash latch upgrades - wait
8 physical reads for flashback new
8 total number of slots
8 Effective IO time
8 Number of read IOs issued
8 background checkpoints started
8 background checkpoints completed
8 number of map operations
8 number of map misses
8 lob reads
8 lob writes
8 lob writes unaligned
8 table lookup prefetch client count
8 physical read total IO requests
8 physical read total multi block requests
8 physical read total bytes
8 physical write total IO requests
8 physical write total multi block requests
8 physical write total bytes
8 db block gets
8 db block gets from cache
8 db block gets from cache (fastpath)
8 db block gets direct
8 consistent gets
8 consistent gets from cache
8 consistent gets from cache (fastpath)
8 consistent gets - examination
8 consistent gets direct
8 physical reads
8 physical reads cache
8 physical reads direct
8 physical read IO requests
8 physical read bytes
8 db block changes
8 consistent changes
8 recovery blocks read
8 recovery blocks read for lost write detection
8 recovery blocks skipped lost write checks

8 physical writes
8 physical writes direct
8 physical writes from cache
8 physical write IO requests
8 physical reads direct temporary tablespace
8 physical writes direct temporary tablespace
8 physical write bytes
8 db corrupt blocks detected
8 db corrupt blocks recovered
8 physical writes non checkpoint
8 summed dirty queue length
8 DBWR checkpoint buffers written
8 DBWR thread checkpoint buffers written
8 DBWR tablespace checkpoint buffers written
8 DBWR parallel query checkpoint buffers written
8 DBWR object drop buffers written
8 DBWR transaction table writes
8 DBWR undo block writes
8 DBWR revisited being-written buffer
8 DBWR lru scans
8 DBWR checkpoints
8 prefetch clients - keep
8 prefetch clients - recycle
8 prefetch clients - default
8 prefetch clients - 2k
8 prefetch clients - 4k
8 prefetch clients - 8k
8 prefetch clients - 16k
8 prefetch clients - 32k
8 change write time
8 redo synch writes
8 redo synch time
8 exchange deadlocks
8 free buffer requested
8 dirty buffers inspected
8 pinned buffers inspected
8 hot buffers moved to head of LRU
8 free buffer inspected
8 commit cleanout failures: write disabled
32 global enqueue gets sync
32 global enqueue gets async
32 global enqueue get time
32 global enqueue releases
32 IPC CPU used by this session
32 gcs messages sent
32 ges messages sent
32 global enqueue CPU used by this session
32 calls to get snapshot scn: kcmgss
32 GTX processes spawned by autotune

32 GTX processes stopped by autotune
32 queries parallelized
32 DML statements parallelized
32 DDL statements parallelized
32 DFO trees parallelized
32 Parallel operations not downgraded
32 Parallel operations downgraded to serial
32 Parallel operations downgraded 75 to 99 pct
32 Parallel operations downgraded 50 to 75 pct
32 Parallel operations downgraded 25 to 50 pct
32 Parallel operations downgraded 1 to 25 pct
32 PX local messages sent
32 PX local messages recv'd
32 PX remote messages sent
32 PX remote messages recv'd
33 Clusterwide global transactions
33 Clusterwide global transactions spanning RAC nodes
33 Forwarded 2PC commands across RAC nodes
40 DBWR fusion writes
40 gc cr blocks served
40 gc cr block build time
40 gc cr block flush time
40 gc cr block send time
40 gc current blocks served
40 gc current block pin time
40 gc current block flush time
40 gc current block send time
40 gc cr blocks received
40 gc cr block receive time
40 gc current blocks received
40 gc current block receive time
40 gc local grants
40 gc remote grants
40 gc blocks lost
40 gc claim blocks lost
40 gc blocks corrupt
40 gc CPU used by this session
40 gc reader bypass grants
64 parse time elapsed
64 parse count (total)
64 parse count (hard)
64 parse count (failures)
64 frame signature mismatch
64 execute count
64 sorts (memory)
64 sorts (disk)
64 sorts (rows)
64 total bytes read and filtered by intelligent storage
64 total bytes returned by intelligent storage after filtering

64 table scans (short tables)
64 table scans (long tables)
64 table scans (rowid ranges)
64 table scans (cache partitions)
64 table scans (direct read)
64 table scan rows gotten
64 table scan blocks gotten
64 table fetch by rowid
64 table fetch continued row
64 cluster key scans
64 cluster key scan block gets
64 rows fetched via callback
64 sage scans
64 blocks sage cache can process
64 blocks sage txn can process
64 blocks sage data can process
64 sage commit cache queries
64 transactions found in sage commit cache
64 blocks helped by sage commit cache
64 blocks sage skipped due to chained rows
64 index crx upgrade (prefetch)
64 index crx upgrade (found)
64 index crx upgrade (positioned)
64 native hash arithmetic execute
64 native hash arithmetic fail
64 index fast full scans (full)
64 index fast full scans (rowid ranges)
64 index fast full scans (direct read)
64 heap block compress
64 HSC OLTP Space Saving
64 HSC OLTP Compressed Blocks
64 HSC IDL Compressed Blocks
64 HSC Compressed Segment Block Changes
64 HSC Heap Segment Block Changes
64 HSC OLTP Non Compressible Blocks
64 Heap Segment Array Inserts
64 Heap Segment Array Updates
64 sql area purged
64 sql area evicted
64 CCursor + sql area evicted
64 session cursor cache hits
64 session cursor cache count
64 workarea memory allocated
64 workarea executions - optimal
64 workarea executions - onepass
64 workarea executions - multipass
64 parse time cpu
72 Batched IO vector read count
72 no buffer to keep pinned count

72 Batched IO single block count
72 Batched IO zero block count
72 Batched IO block miss count
72 Batched IO double miss count
72 Batched IO (full) vector count
72 Batched IO (space) vector count
72 Batched IO (bound) vector count
72 Batched IO same unit count
72 Batched IO buffer defrag count
72 Batched IO slow jump count
72 buffer is pinned count
72 buffer is not pinned count
72 Batched IO vector block count
128 OTC commit optimization attempts
128 OTC commit optimization hits
128 OTC commit optimization failure - setup
128 IMU- failed to get a private strand
128 Misses for writing mapping
128 TBS Extension: tasks created
128 TBS Extension: tasks executed
128 TBS Extension: files extended
128 total number of times SMON posted
128 SMON posted for undo segment recovery
128 SMON posted for txn recovery for other instances
128 SMON posted for instance recovery
128 SMON posted for undo segment shrink
128 SMON posted for dropping temp segment
128 queue update without cp update
128 leaf node splits
128 leaf node 90-10 splits
128 branch node splits
128 failed probes on index block reclamation
128 recursive aborts on index block reclamation
128 index fetch by key
128 index scans kdixs1
128 queue splits
128 queue flush
128 queue position update
128 queue single row
128 queue ocp pages
128 queue qno pages
128 HSC OLTP positive compression
128 HSC OLTP negative compression
128 HSC OLTP recursive compression
128 HSC OLTP inline compression
128 HSC OLTP Drop Column
128 HSC OLTP Compression skipped rows
128 HSC OLTP compression block checked
128 securefile allocation bytes

128 securefile allocation chunks
128 securefile direct read bytes
128 securefile direct write bytes
128 securefile direct read ops
128 securefile direct write ops
128 securefile inode read time
128 securefile inode write time
128 securefile inode ioreap time
128 securefile bytes non-transformed
128 securefile number of non-transformed flushes
128 securefile bytes encrypted
128 securefile bytes cleartext
128 securefile compressed bytes
128 securefile uncompressed bytes
128 securefile bytes deduplicated
128 securefile create dedup set
128 securefile destroy dedup set
128 securefile add dedupd lob to set
128 securefile rmv from dedup set
128 securefile reject deduplication
128 securefile dedup prefix hash match
128 securefile number of flushes
128 securefile dedup flush too low
128 securefile dedup callback oper final
128 securefile dedup hash collision
128 DX/BB enqueue lock foreground requests
128 DX/BB enqueue lock foreground wait time
128 DX/BB enqueue lock background gets
128 DX/BB enqueue lock background get time
128 cursor authentications
128 LOB table id lookup cache misses
128 CPU used when call started
128 process last non-idle time
128 messages sent
128 messages received
128 background timeouts
128 calls to kcmgcs
128 calls to kcmgrs
128 calls to kcmgas
128 shared io pool buffer get success
128 shared io pool buffer get failure
128 transaction lock foreground requests
128 transaction lock foreground wait time
128 transaction lock background gets
128 transaction lock background get time
128 undo change vector size
128 transaction tables consistent reads - undo records applied
128 transaction tables consistent read rollbacks
128 data blocks consistent reads - undo records applied

128 no work - consistent read gets
128 cleanouts only - consistent read gets
128 rollbacks only - consistent read gets
128 cleanouts and rollbacks - consistent read gets
128 RowCR attempts
128 RowCR hits
128 RowCR - row contention
128 RowCR - resume
128 rollback changes - undo records applied
128 transaction rollbacks
128 immediate (CURRENT) block cleanout applications
128 immediate (CR) block cleanout applications
128 deferred (CURRENT) block cleanout applications
128 commit txn count during cleanout
128 active txn count during cleanout
128 cleanout - number of ktugct calls
128 immediate CR cleanouts (index blocks)
128 deferred CUR cleanouts (index blocks)
128 Commit SCN cached
128 Cached Commit SCN referenced
128 Block Cleanout Optim referenced
128 auto extends on undo tablespace
128 drop segment calls in space pressure
128 total number of undo segments dropped
128 doubling up with imu segment
128 tune down retentions in space pressure
128 steps of tune down ret. in space pressure
128 space was found by tune down
128 space was not found by tune down
128 global undo segment hints helped
128 global undo segment hints were stale
128 local undo segment hints helped
128 local undo segment hints were stale
128 undo segment header was pinned
128 IMU commits
128 IMU Flushes
128 IMU contention
128 IMU recursive-transaction flush
128 IMU undo retention flush
128 IMU ktichg flush
128 IMU bind flushes
128 IMU mbu flush
128 IMU pool not allocated
128 IMU CR rollbacks
128 IMU undo allocation size
128 IMU Redo allocation size

469 rows selected.

APPENDIX E

A Complete List of All Statistics with the V\$SYSSTAT View

*If a man will begin with certainties, he shall end in doubts; but if he will be content to begin
with doubts, he shall end in certainty.*
—Francis Bacon

This appendix lists all statistics stored in the V\$SYSSTAT view in Oracle 11g. The purpose of providing this complete list is to help you recognize the relevant statistics for troubleshooting various Oracle performance and scalability issues that might be specific to your product. The meaning of each statistic is self-explanatory by its name. You can infer the context of each statistic metric by its class represented in a number as described in the preceding appendix. A quiz here: if you randomly pick a few statistic metrics from the list below and you know clearly what they are about, then you already have an above-average skill set in troubleshooting Oracle performance and scalability issues.

The SQL query executed against the V\$SYSSTAT and the corresponding output are shown as follows (Note that we omitted the VALUE and STAT_ID columns

to save space. When you troubleshoot a real Oracle performance issue, it actually is the VALUE column that would be most interesting.):

```
SQL> select statistic# || '[' || class || ']' || name from V$SYSSTAT;
STATISTIC# [ CLASS] NAME
```

```
-----
0[ 1] OS CPU Qt wait time
1[ 1] logons cumulative
2[ 1] logons current
3[ 1] opened cursors cumulative
4[ 1] opened cursors current
5[ 1] user commits
6[ 1] user rollbacks
7[ 1] user calls
8[ 1] recursive calls
9[ 1] recursive cpu usage
10[ 1] pinned cursors current
11[ 1] session logical reads
12[ 1] session stored procedure space
13[ 128] CPU used when call started
14[ 1] CPU used by this session
15[ 1] DB time
16[ 1] cluster wait time
17[ 1] concurrency wait time
18[ 1] application wait time
19[ 1] user I/O wait time
20[ 1] scheduler wait time
21[ 1] non-idle wait time
22[ 1] non-idle wait count
23[ 1] session connect time
24[ 128] process last non-idle time
25[ 1] session uga memory
26[ 1] session uga memory max
27[ 128] messages sent
28[ 128] messages received
29[ 128] background timeouts
30[ 128] remote Oradebug requests
31[ 1] session pga memory
32[ 1] session pga memory max
33[ 128] recursive system API invocations
34[ 4] enqueue timeouts
35[ 4] enqueue waits
36[ 4] enqueue deadlocks
37[ 4] enqueue requests
38[ 4] enqueue conversions
39[ 4] enqueue releases
40[ 32] global enqueue gets sync
41[ 32] global enqueue gets async
42[ 32] global enqueue get time
43[ 32] global enqueue releases
44[ 8] physical read total IO requests
```

```
45[ 8] physical read total multi block requests
46[ 8] physical read requests optimized
47[ 8] physical read total bytes
48[ 8] physical write total IO requests
49[ 8] physical write total multi block requests
50[ 8] physical write total bytes
51[ 64] cell physical IO interconnect bytes
52[ 128] spare statistic 1
53[ 128] spare statistic 2
54[ 128] spare statistic 3
55[ 128] spare statistic 4
56[ 32] IPC CPU used by this session
57[ 32] gcs messages sent
58[ 32] ges messages sent
59[ 32] global enqueue CPU used by this session
60[ 4] max cf enq hold time
61[ 4] total cf enq hold time
62[ 4] total number of cf enq holders
63[ 8] db block gets
64[ 8] db block gets from cache
65[ 8] db block gets from cache (fastpath)
66[ 8] db block gets direct
67[ 8] consistent gets
68[ 8] consistent gets from cache
69[ 8] consistent gets from cache (fastpath)
70[ 8] consistent gets - examination
71[ 8] consistent gets direct
72[ 8] physical reads
73[ 8] physical reads cache
74[ 8] physical read flash cache hits
75[ 8] physical reads direct
76[ 8] physical read IO requests
77[ 8] physical read bytes
78[ 8] db block changes
79[ 8] consistent changes
80[ 8] recovery blocks read
81[ 8] recovery blocks read for lost write detection
82[ 8] recovery blocks skipped lost write checks
83[ 8] physical writes
84[ 8] physical writes direct
85[ 8] physical writes from cache
86[ 8] physical write IO requests
87[ 8] flash cache inserts
88[ 8] physical reads direct temporary tablespace
89[ 8] physical writes direct temporary tablespace
90[ 8] physical write bytes
91[ 8] flash cache eviction: invalidated
92[ 8] flash cache eviction: buffer pinned
93[ 8] flash cache eviction: aged out
94[ 8] flash cache insert skip: not current
95[ 8] flash cache insert skip: DBWR overloaded
```

96[8] flash cache insert skip: exists
97[8] flash cache insert skip: not useful
98[8] flash cache insert skip: modification
99[8] flash cache insert skip: corrupt
100[8] db corrupt blocks detected
101[8] db corrupt blocks recovered
102[8] physical writes non checkpoint
103[8] summed dirty queue length
104[8] DBWR checkpoint buffers written
105[8] DBWR thread checkpoint buffers written
106[8] DBWR tablespace checkpoint buffers written
107[8] DBWR parallel query checkpoint buffers written
108[8] DBWR object drop buffers written
109[8] DBWR transaction table writes
110[8] DBWR undo block writes
111[8] DBWR revisited being-written buffer
112[8] DBWR lru scans
113[8] DBWR checkpoints
114[40] DBWR fusion writes
115[8] prefetch clients - keep
116[8] prefetch clients - recycle
117[8] prefetch clients - default
118[8] prefetch clients - 2k
119[8] prefetch clients - 4k
120[8] prefetch clients - 8k
121[8] prefetch clients - 16k
122[8] prefetch clients - 32k
123[8] change write time
124[8] redo synch writes
125[8] redo synch time
126[8] exchange deadlocks
127[8] free buffer requested
128[8] dirty buffers inspected
129[8] pinned buffers inspected
130[8] hot buffers moved to head of LRU
131[8] free buffer inspected
132[8] commit cleanout failures: write disabled
133[8] commit cleanout failures: block lost
134[8] commit cleanout failures: cannot pin
135[8] commit cleanout failures: hot backup in progress
136[8] commit cleanout failures: buffer being written
137[8] commit cleanout failures: callback failure
138[8] commit cleanouts
139[8] commit cleanouts successfully completed
140[8] recovery array reads
141[8] recovery array read time
142[8] CR blocks created
143[8] current blocks converted for CR
144[8] switch current to new buffer
145[8] write clones created in foreground
146[8] write clones created in background

147[8] write clones created for recovery
148[8] recovery block gets from cache
149[8] physical reads cache prefetch
150[8] physical reads prefetch warmup
151[8] prefetched blocks aged out before use
152[8] prefetch warmup blocks aged out before use
153[8] prefetch warmup blocks flushed out before use
154[8] physical reads retry corrupt
155[8] physical reads direct (lob)
156[8] physical writes direct (lob)
157[8] cold recycle reads
158[8] shared hash latch upgrades - no wait
159[8] shared hash latch upgrades - wait
160[8] physical reads for flashback new
161[128] calls to kcmgcs
162[128] calls to kcmgrs
163[128] calls to kcmgas
164[32] calls to get snapshot scn: kcmgss
165[2] redo blocks read for recovery
166[2] redo k-bytes read for recovery
167[2] redo k-bytes read for terminal recovery
168[2] redo entries
169[2] redo size
170[2] redo entries for lost write detection
171[2] redo size for lost write detection
172[2] redo size for direct writes
173[2] redo buffer allocation retries
174[2] redo wastage
175[2] redo writes
176[2] redo blocks written
177[2] redo write time
178[2] redo blocks checksummed by FG (exclusive)
179[2] redo blocks checksummed by LGWR
180[2] redo log space requests
181[2] redo log space wait time
182[2] redo ordering marks
183[2] redo subscn max counts
184[2] redo write broadcast ack time
185[2] redo write broadcast ack count
186[2] redo k-bytes read total
187[2] redo k-bytes read (memory)
188[2] redo k-bytes read total by LNS
189[2] redo k-bytes read (memory) by LNS
190[1] file io service time
191[1] file io wait time
192[40] gc cr blocks served
193[40] gc cr block read wait time
194[40] gc cr block build time
195[40] gc cr block flush time
196[40] gc cr block send time
197[40] gc current blocks served

```

198 [ 40] gc current block pin time
199 [ 40] gc current block flush time
200 [ 40] gc current block send time
201 [ 40] gc cr blocks received
202 [ 40] gc cr block receive time
203 [ 40] gc current blocks received
204 [ 40] gc current block receive time
205 [ 40] gc local grants
206 [ 40] gc remote grants
207 [ 40] gc kbytes sent
208 [ 40] gc kbytes saved
209 [ 40] gc blocks compressed
210 [ 40] gc blocks lost
211 [ 40] gc claim blocks lost
212 [ 40] gc blocks corrupt
213 [ 40] gc CPU used by this session
214 [ 40] gc reader bypass grants
215 [ 8] total number of slots
216 [ 8] Effective IO time
217 [ 8] Number of read IOs issued
218 [ 8] background checkpoints started
219 [ 8] background checkpoints completed
220 [ 8] number of map operations
221 [ 8] number of map misses
222 [ 2] flashback log writes
223 [ 2] flashback log write bytes
224 [ 64] cell physical IO bytes saved during optimized file creation
225 [ 64] cell physical IO bytes saved during optimized RMAN file restore
226 [ 64] cell physical IO bytes eligible for predicate offload
227 [ 8] cell physical IO bytes saved by storage index
228 [ 64] cell smart IO session cache lookups
229 [ 64] cell smart IO session cache hits
230 [ 64] cell smart IO session cache soft misses
231 [ 64] cell smart IO session cache hard misses
232 [ 64] cell smart IO session cache hwm
233 [ 64] cell num smart IO sessions in rdbms block IO due to user
234 [ 64] cell num smart IO sessions in rdbms block IO due to big payload
235 [ 64] cell num smart IO sessions using passthru mode due to user
236 [ 64] cell num smart IO sessions using passthru mode due to cellsrv
237 [ 64] cell num smart IO sessions using passthru mode due to timezone
238 [ 64] cell num smart file creation sessions using rdbms block IO mode
239 [ 64] cell physical IO interconnect bytes returned by smart scan
240 [ 64] cell session smart scan efficiency
241 [ 72] Batched IO vector read count
242 [ 72] Batched IO vector block count
243 [ 72] Batched IO single block count
244 [ 72] Batched IO zero block count
245 [ 72] Batched IO block miss count
246 [ 72] Batched IO double miss count
247 [ 72] Batched IO (full) vector count
248 [ 72] Batched IO (space) vector count

```

249[72] Batched IO (bound) vector count
250[72] Batched IO same unit count
251[72] Batched IO buffer defrag count
252[72] Batched IO slow jump count
253[128] shared io pool buffer get success
254[128] shared io pool buffer get failure
255[1] temp space allocated (bytes)
256[1] serializable aborts
257[128] transaction lock foreground requests
258[128] transaction lock foreground wait time
259[128] transaction lock background gets
260[128] transaction lock background get time
261[128] undo change vector size
262[128] transaction tables consistent reads - undo records applied
263[128] transaction tables consistent read rollbacks
264[128] data blocks consistent reads - undo records applied
265[128] no work - consistent read gets
266[128] cleanouts only - consistent read gets
267[128] rollbacks only - consistent read gets
268[128] cleanouts and rollbacks - consistent read gets
269[128] RowCR attempts
270[128] RowCR hits
271[128] RowCR - row contention
272[128] RowCR - resume
273[128] rollback changes - undo records applied
274[128] transaction rollbacks
275[128] immediate (CURRENT) block cleanout applications
276[128] immediate (CR) block cleanout applications
277[128] deferred (CURRENT) block cleanout applications
278[128] commit txn count during cleanout
279[128] active txn count during cleanout
280[128] cleanout - number of ktugct calls
281[128] immediate CR cleanouts (index blocks)
282[128] deferred CUR cleanouts (index blocks)
283[128] Commit SCN cached
284[128] Cached Commit SCN referenced
285[128] Block Cleanout Optim referenced
286[128] min active SCN optimization applied on CR
287[128] auto extends on undo tablespace
288[128] drop segment calls in space pressure
289[128] total number of undo segments dropped
290[128] doubling up with imu segment
291[128] tune down retentions in space pressure
292[128] steps of tune down ret. in space pressure
293[128] space was found by tune down
294[128] space was not found by tune down
295[1] commit batch/immediate requested
296[1] commit batch requested
297[1] commit immediate requested
298[1] commit batch/immediate performed
299[1] commit batch performed


```
300 [ 1] commit immediate performed
301 [ 1] commit wait/nowait requested
302 [ 1] commit nowait requested
303 [ 1] commit wait requested
304 [ 1] commit wait/nowait performed
305 [ 1] commit nowait performed
306 [ 1] commit wait performed
307 [128] global undo segment hints helped
308 [128] global undo segment hints were stale
309 [128] local undo segment hints helped
310 [128] local undo segment hints were stale
311 [128] undo segment header was pinned
312 [128] IMU commits
313 [128] IMU Flushes
314 [128] IMU contention
315 [128] IMU recursive-transaction flush
316 [128] IMU undo retention flush
317 [128] IMU ktichg flush
318 [128] IMU bind flushes
319 [128] IMU mbu flush
320 [128] IMU pool not allocated
321 [128] IMU CR rollbacks
322 [128] IMU undo allocation size
323 [128] IMU Redo allocation size
324 [128] IMU- failed to get a private strand
325 [128] Misses for writing mapping
326 [128] segment dispenser load tasks
327 [128] segment dispenser load empty
328 [128] segment dispenser allocations
329 [128] segment cfs allocations
330 [128] segment chunks allocation from dispenser
331 [128] segment total chunk allocation
332 [128] TBS Extension: tasks created
333 [128] TBS Extension: tasks executed
334 [128] TBS Extension: files extended
335 [128] TBS Extension: bytes extended
336 [128] total number of times SMON posted
337 [128] SMON posted for undo segment recovery
338 [128] SMON posted for txn recovery for other instances
339 [128] SMON posted for instance recovery
340 [128] SMON posted for undo segment shrink
341 [128] SMON posted for dropping temp segment
342 [128] segment prealloc tasks
343 [128] segment prealloc ops
344 [128] segment prealloc bytes
345 [128] segment prealloc time (ms)
346 [128] segment prealloc ufs2cfs bytes
347 [ 64] table scans (short tables)
348 [ 64] table scans (long tables)
349 [ 64] table scans (rowid ranges)
350 [ 64] table scans (cache partitions)
```

351[64] table scans (direct read)
352[64] table scan rows gotten
353[64] table scan blocks gotten
354[64] table fetch by rowid
355[64] table fetch continued row
356[64] cluster key scans
357[64] cluster key scan block gets
358[64] rows fetched via callback
359[64] cell scans
360[128] cell blocks processed by cache layer
361[128] cell blocks processed by txn layer
362[128] cell blocks processed by data layer
363[128] cell blocks processed by index layer
364[64] cell commit cache queries
365[64] cell transactions found in commit cache
366[64] cell blocks helped by commit cache
367[64] cell blocks helped by minscn optimization
368[64] cell blocks skipped due to chained rows
369[192] cell simulated physical IO bytes eligible for predicate offload
370[192] cell simulated physical IO bytes returned by predicate offload
371[192] cell simulated session smart scan efficiency
372[64] cell CUs sent uncompressed
373[64] cell CUs sent compressed
374[64] cell CUs sent head piece
375[64] cell CUs processed for uncompressed
376[64] cell CUs processed for compressed
377[64] cell IO uncompressed bytes
378[128] queue update without cp update
379[64] index crx upgrade (prefetch)
380[64] index crx upgrade (found)
381[64] index crx upgrade (positioned)
382[128] leaf node splits
383[128] leaf node 90-10 splits
384[128] branch node splits
385[128] root node splits
386[128] failed probes on index block reclamation
387[128] recursive aborts on index block reclamation
388[128] index reclamation/extension switch
389[64] native hash arithmetic execute
390[64] native hash arithmetic fail
391[8] lob reads
392[8] lob writes
393[8] lob writes unaligned
394[64] cell index scans
395[64] index fast full scans (full)
396[64] index fast full scans (rowid ranges)
397[64] index fast full scans (direct read)
398[128] index fetch by key
399[128] index scans kdiixs1
400[128] queue splits
401[128] queue flush

402 [128] queue position update
403 [128] queue single row
404 [128] queue ocp pages
405 [128] queue qno pages
406 [64] heap block compress
407 [64] HSC OLTP Space Saving
408 [64] HSC OLTP Compressed Blocks
409 [64] HSC IDL Compressed Blocks
410 [64] HSC Compressed Segment Block Changes
411 [64] HSC Heap Segment Block Changes
412 [64] HSC OLTP Non Compressible Blocks
413 [128] HSC OLTP positive compression
414 [128] HSC OLTP negative compression
415 [128] HSC OLTP recursive compression
416 [128] HSC OLTP inline compression
417 [128] HSC OLTP Drop Column
418 [128] HSC OLTP Compression skipped rows
419 [128] HSC OLTP compression block checked
420 [64] Heap Segment Array Inserts
421 [64] Heap Segment Array Updates
422 [128] securefile allocation bytes
423 [128] securefile allocation chunks
424 [128] securefile direct read bytes
425 [128] securefile direct write bytes
426 [128] securefile direct read ops
427 [128] securefile direct write ops
428 [128] securefile inode read time
429 [128] securefile inode write time
430 [128] securefile inode ioreap time
431 [128] securefile bytes non-transformed
432 [128] securefile number of non-transformed flushes
433 [128] securefile bytes encrypted
434 [128] securefile bytes cleartext
435 [128] securefile compressed bytes
436 [128] securefile uncompressed bytes
437 [128] securefile bytes deduplicated
438 [128] securefile create dedup set
439 [128] securefile destroy dedup set
440 [128] securefile add dedupd lob to set
441 [128] securefile rmv from dedup set
442 [128] securefile reject deduplication
443 [128] securefile dedup prefix hash match
444 [128] securefile number of flushes
445 [128] securefile dedup flush too low
446 [128] securefile dedup callback oper final
447 [128] securefile dedup hash collision
448 [128] securefile dedup fits inline
449 [128] CC CUs Compressed
450 [128] CC Query Low CUs Compressed
451 [128] CC Query High CUs Compressed
452 [128] CC Archive CUs Compressed

453 [128] CC Compressed Length Compressed
454 [128] CC Decompressed Length Compressed
455 [128] CC Rows Compressed
456 [128] CC Rows Not Compressed
457 [128] CC CU Row Pieces Compressed
458 [128] CC CUs Decompressed
459 [128] CC Query Low CUs Decompressed
460 [128] CC Query High CUs Decompressed
461 [128] CC Archive CUs Decompressed
462 [128] CC Compressed Length Decompressed
463 [128] CC Decompressed Length Decompressed
464 [128] CC Columns Decompressed
465 [128] CC Total Columns for Decompression
466 [128] CC Total Rows for Decompression
467 [128] CC Pieces Buffered for Decompression
468 [128] CC Total Pieces for Decompression
469 [128] CC DML CUs Decompressed
470 [128] CC Scan CUs Decompressed
471 [128] CC Turbo Scan CUs Decompressed
472 [128] CC Rowid CUs Decompressed
473 [128] CC Analyze CUs Decompressed
474 [128] CC Dump CUs Decompressed
475 [128] CC Check CUs Decompressed
476 [128] CC Analyzer Calls
477 [64] sql area purged
478 [64] sql area evicted
479 [64] CCursor + sql area evicted
480 [1] No. of Encrypt ops
481 [1] No. of Decrypt ops
482 [1] No. of XS Sessions Created
483 [1] No. of XS Sessions Attached
484 [1] No. of Namespaces Created
485 [1] No. of User Callbacks Executed
486 [1] No. of Roles Enabled or Disabled
487 [1] No. of Principal Cache Misses
488 [1] No. of Principal Invalidations
489 [128] DX/BB enqueue lock foreground requests
490 [128] DX/BB enqueue lock foreground wait time
491 [128] DX/BB enqueue lock background gets
492 [128] DX/BB enqueue lock background get time
493 [33] Clusterwide global transactions
494 [33] Clusterwide global transactions spanning RAC nodes
495 [33] Forwarded 2PC commands across RAC nodes
496 [32] GTX processes spawned by autotune
497 [32] GTX processes stopped by autotune
498 [64] session cursor cache hits
499 [64] session cursor cache count
500 [1] java call heap total size
501 [1] java call heap total size max
502 [1] java call heap used size
503 [1] java call heap used size max

```

504[ 1] java call heap live size
505[ 1] java call heap live size max
506[ 1] java call heap object count
507[ 1] java call heap object count max
508[ 1] java call heap live object count
509[ 1] java call heap live object count max
510[ 1] java call heap gc count
511[ 1] java call heap collected count
512[ 1] java call heap collected bytes
513[ 1] java session heap used size
514[ 1] java session heap used size max
515[ 1] java session heap live size
516[ 1] java session heap live size max
517[ 1] java session heap object count
518[ 1] java session heap object count max
519[ 1] java session heap live object count
520[ 1] java session heap live object count max
521[ 1] java session heap gc count
522[ 1] java session heap collected count
523[ 1] java session heap collected bytes
524[ 128] cursor authentications
525[ 32] queries parallelized
526[ 32] DML statements parallelized
527[ 32] DDL statements parallelized
528[ 32] DFO trees parallelized
529[ 32] Parallel operations not downgraded
530[ 32] Parallel operations downgraded to serial
531[ 32] Parallel operations downgraded 75 to 99 pct
532[ 32] Parallel operations downgraded 50 to 75 pct
533[ 32] Parallel operations downgraded 25 to 50 pct
534[ 32] Parallel operations downgraded 1 to 25 pct
535[ 32] PX local messages sent
536[ 32] PX local messages recv'd
537[ 32] PX remote messages sent
538[ 32] PX remote messages recv'd
539[ 72] buffer is pinned count
540[ 72] buffer is not pinned count
541[ 72] no buffer to keep pinned count
542[ 64] workarea memory allocated
543[ 64] workarea executions - optimal
544[ 64] workarea executions - onepass
545[ 64] workarea executions - multipass
546[ 128] LOB table id lookup cache misses
547[ 64] parse time cpu
548[ 64] parse time elapsed
549[ 64] parse count (total)
550[ 64] parse count (hard)
551[ 64] parse count (failures)
552[ 64] parse count (describe)
553[ 64] frame signature mismatch
554[ 64] execute count

```

```
555[ 1] bytes sent via SQL*Net to client
556[ 1] bytes received via SQL*Net from client
557[ 1] SQL*Net roundtrips to/from client
558[ 1] bytes sent via SQL*Net to dblink
559[ 1] bytes received via SQL*Net from dblink
560[ 1] SQL*Net roundtrips to/from dblink
561[ 1] bytes via SQL*Net vector to client
562[ 1] bytes via SQL*Net vector from client
563[ 1] bytes via SQL*Net vector to dblink
564[ 1] bytes via SQL*Net vector from dblink
565[ 64] sorts (memory)
566[ 64] sorts (disk)
567[ 64] sorts (rows)
568[ 128] OTC commit optimization attempts
569[ 128] OTC commit optimization hits
570[ 128] OTC commit optimization failure - setup
571[ 8] cell flash cache read hits
572[ 1] Workload Capture: size (in bytes) of recording
573[ 1] Workload Capture: dbtime
574[ 1] Workload Capture: user calls
575[ 1] Workload Capture: user calls flushed
576[ 1] Workload Capture: unreplayable user calls
577[ 1] Workload Capture: user txns
578[ 1] Workload Capture: user logins
579[ 1] Workload Capture: unsupported user calls
580[ 1] Workload Capture: errors
581[ 1] Workload Replay: dbtime
582[ 1] Workload Replay: network time
583[ 1] Workload Replay: think time
584[ 1] Workload Replay: time gain
585[ 1] Workload Replay: time loss
586[ 1] Workload Replay: user calls
587[ 1] Workload Replay: deadlocks resolved

588 rows selected.
```

The above list is lengthy, but it gives you a full view of what is available from the V\$SYSSTAT view to assist you in your Oracle performance and scalability troubleshooting efforts. If you are interested only in a few statistics from this list, you can just query them without having to rely fully on the Enterprise Manager DB Console or any other tools.

Index

- /WEB-INF/, 361
- 1NF (first-normal form), 287
- 2NF (second-normal form), 288
- 3NF (third-normal form), 288, 293
- 4NF (fourth normal form), 290
- 5NF (5th normal form), 292–293
- :B_n, 549
- :n, 549
- <<StereoType>>, 355
- @Autowired, 356
- @Component, 357
- @Controller, 354–355
- @ModelAttribute, 355–356
- @PathVariable, 356, 363
- @RequestMapping, 354, 355, 363
- @RequestParam, 356
- @SessionAttribute, 354–356

- ACID properties of transactions, 141
- ACL table, 304
- ACL. See Access control list
- ACL_CLASS, 305, 395
- ACL_ENTRY, 306, 395
- ACL_OBJECT_IDENTITY, 305, 395

- ACL_SID, 304, 395
- ADDM. See Automatic database diagnostic monitor
- AIO. See Asynchronous I/O
- AIX, 15, 510
- AMM. See Automatic memory management
- ANSI/ISO, 10
- AOP. See Aspect-oriented programming
- AQ. See Advanced queuing
- ARCn. See Archiver
- ARRAY_SIZE, 482
- ASH. See Active session history
- ASM. See Automatic storage management, 3
- AUTOCOMMIT, 638
- AUTOTRACE, 425
- AWR. See Automatic workload repository
- Access control, 387
- Access control list (ACL), 395
- Account status, 28
- Account unlock, 41
- AclAfterInvocationProvider, 406
- AclCache, 398
- AclEntryVoter, 406
- AclService, 398

- Actions, 273
- Active session history, 460
- Actor, 273
- Adams, Douglas, 504
- Adams, Scott, 34
- Adapter pattern, 319
- Adaptive cursor sharing, 249, 569
- Administrator, 29
- Advanced queuing (AQ), 3
- Advisor
 - data recovery, 462
 - memory, 462
 - MTTR, 466
 - partition, 253
 - segment, 466
 - SQL, 467
- Advisor central, 459
- Advisory
 - JAVA pool, 205
 - PGA aggregate target, 239
 - shared pool usage, 239
- Advisory statistics, 199
- Agile development, 269
- All metrics, 656
- All statistics, 667
- All wait events, 648
- All-key, 282
- Allocation history, 464
- Anomaly
 - delete, 281
 - insert, 281
 - update, 281
- Antijoin, 436
- Application controller, 347
- Application-transparent, 2
- Application/json, 384
- Archiver (ARcN), 84
- Armstrong's Axioms, 285
- Array fetching, 485
- Array processing, 478, 482
- Aspect-oriented programming (AOP), 333
- Asynchronous I/O (AIO), 510
- Asynchronous commit, 244
- Atomicity. *See* ACID properties of transactions
- Attributes, 279, 283
- Augmentation. *See* Armstrong's Axioms
- Authentication
 - form-based, 387
 - HTTP basic, 387
- Authentication types, 388
- Authorization, 387
- Auto-extension, 123
- Auto-tune features, 459
- Automatic database diagnostic monitor (ADDM), 460
- Automatic maintenance tasks, 19
- Automatic memory management (AMM), 2, 24, 112, 249
- Automatic optimizer statistics
 - gathering, 245
- Automatic shared memory management (ASMM), 102
- Automatic shared memory tuning, 245
- Automatic storage management (ASM), 3, 242
- Automatic undo management, 462
- Automatic workload repository (AWR), 136, 161–226, 162, 262
- Availability, 36
- B+ tree, 513
- B-tree, 451
- BCNF (Boyce-Codd normal form), 289
- BFILE, 65
- BITMAP_MERGE_AREA_SIZE, 106
- BLOB. *See* Binary large object.
- BSTAT, 162
- BSTAT-ESTAT, 162
- Background processes, 83
- Bacon, Francis, 667
- Bailey, James, 227
- Beecher, Henry Ward, 111
- Best practices, 322
- Bigfile tablespace, 120
- Binary large object (BLOB), 65
- Bind variable, 548, 569
- Block changes, 168
- Block size, 24
 - standard, 167
- Blocks changed per read, 168
- Bottleneck, 532
- Boyce, Raymond, 10
- Braque, Georges, 127

- Buffer cache, 88
 - database block, 166
 - management, 236
- Buffer pinned count, 187
- Buffer pool advisory, 200
- Buffer pool statistics, 199
- Buffer wait statistics, 206
- Build
 - installable, 321
 - testable, 321
- Built-in features, 36
- Bulk transaction, 571
- Business operations, 404
- Business processes, 274
- Business rules, 274
- Business rules and data integrity
 - enforcing, 309
- C#, 11
- C++, 11
- CBO statistics, 421
 - gathering, 424
 - locking and unlocking, 425
- CBO. See Cost-based optimizer
- CJQ0. See Job queue coordinator (CJQ0)
- CKPT. See Checkpoint
- CLI. See Command-line interface
- CLOB. See Character large object
- CONNECT BY PRIOR, 444
- CPU GHz power, 262
- CPU metrics, 185
- CPUs, 2
- CR blocks created, 186
- CR. See Consistent read
- CR. See Cost reduction
- CREATE_BITMAP_AREA_SIZE, 106
- CURSOR_SHARING, 249, 547, 549
- Cache buffers LRU chain, 209
- Cache buffers chains, 208
- Cache hit, 87
- Cache miss, 87
- Calls to Kcmxxx metrics, 187
- Candidate key, 282
- Cardinality, 58, 276, 277, 283
- Cartesian join, 436
- Chamberlin, Donald, 10
- Change_on_install, 41
- Character large object (CLOB), 65
- Character sets, 24
- Checkpoint (CKPT), 84
- Checkpoint queue latch, 209
- Cheever, John, 473
- Chen, Peter, 277
- Chesterton, Gilbert Keith, 14
- Chinese proverb, 1
- Churchill, Winston, 459
- Cleanout-related metrics, 187
- Cloning production database, 240
- Cluster key scan metrics, 187
- Clustering
 - active/active, 237
 - active/passive, 237
- Codd, Edgar F., 282
- Coding path, 315
- Column, 283
- Command line interface (CLI) versus
 - GUI-based console, 35
- Command-line interface (CLI), 34
- Concurrency model
 - scalable, 151
- Concurrency wait time, 187
- Confucius, 531, 571, 594
- Connect descriptor, 37
- Consistency. See ACID properties of
 - transactions
- Consistent gets metrics, 187
- Consistent read (CR), 188
- Constraint, 61
 - check, 61
 - foreign key, 311
 - NOT NULL, 310
 - primary key, 311
- Content, 283
- Continuous improvements, 322
- Control files, 117
- Cost reduction (CR), 446
- Cost-based optimizer (CBO), 81, 417
- Covering index, 452
- Creating AWR report, 643
- Creating an Oracle database, 18–24
- Creating application schema
 - object, 299
- Crevier, Daniel, 417
- Crosscutting, 335

- Cursor sharing
 - intelligent, 249
- Curtis, George William, 547
- Custom, 30
- Customer
 - escalations, 323
 - feedback, 322
- DAO
 - Hibernate, 373
- DB2, 10
- DBA_LOCK. See Enqueue
- DBA_LOCK_INTERNAL. See Enqueue
- DBWR metrics, 186
- DBWR. See Database writer
- DB_BLOCK_BUFFERS, 91
- DB_BLOCK_SIZE, 57, 92, 199
- DB_CACHE_SIZE, 199
- DB_FILE_MULTIPLE_READ_COUNT, 132
- DCA. See Database configuration assistant.
- DCL See Data control language
- DDL locks, 149
- DDL. See Data definition language
- DESC <table>, 42
- DI. See Dependency injection
- DII. See Data_in_index
- DIO. See Direct I/O
- DML lock allocation, 209
- DML. See Data manipulation language
- DOMAIN, 64
- DX – Distributed transaction enqueue.
 - See Enqueue
- Da Vinci, Leonardo, 52
- Data access path, 504
- Data block, 52, 56
- Data buffering, 507
- Data consistency, 280
- Data consistency and concurrency, 139–160
- Data control language (DCL), 10
- Data definition language (DDL), 10
- Data files, 119
- Data guard, 236
- Data manipulation language (DML), 10
- Data mining, 3
- Data provisioning, 247
- Data_in_index (DII), 452
- Database
 - replay, 244
 - redo log groups, 27
 - storage, 23, 27
- Database block buffer cache, 102
- Database configuration assistant (DCA), 18
- Database control, 19
- Database file locations, 21
- Database resident connection pool (DRCP), 249, 263
- Database smart flash cache, 251
- Database statistics
 - gathering, 423
- Database writer (DBWR), 84
- Date-Fagin 5NF golden rule, 294
- Db block metrics
 - db block changes, 188
 - db block gets, 188
 - db block gets direct, 188
 - db block gets from cache, 188
- Db file scattered read, 132
- Dbfile sequential read, 133, 507
- Db2, 40
- Deadlock, 150
- Decomposition
 - lossless, 285
- Decomposition. See Armstrong's Axioms
- Dedicated architecture, 90
- Dedicated versus shared Oracle server
 - architecture, 89–91
- Dedicated versus shared server models, 260
- Degree, 283
- Denormalization, 294
- Dependency, 281
- Dependency injection (DI), 333
- Derived parameters, 91
- Design
 - conceptual, 275
 - internal, 280
 - logical, 280
 - physical, 295
- Design patterns
 - application, 318
 - database, 316
- Dictionary cache, 88
- Dictionary cache stats, 218
- Direct I/O (DIO), 509

- Dirty read. See read phenomena and data inconsistencies
- Disk groups, 243
- Disk stripping, 243
- Dispatcher servlet, 347
- Document, 377
- Domain, 58, 283
- Domain index type, 65
- Domain object, 398
- Double-buffering, 504
- Durability. See ACID properties of transactions
- Dynamic memory pools, 241
- Dynamic sampling
 - optimizer statistics, 239
- EJB. See Enterprise Java bean
- EM DBConsole
 - creating, 646
- ER diagram, 46–47
- ER diagram. See Entity-relational diagram
- ERD and UML
 - mapping between, 279
- ERD. See Entity-relational diagram
- ESTAT, 162
- EXACT, See CURSOR_SHARING
- EXECUTION PLANS
 - optimum, 428
 - suboptimum, 428
- Einstein, Albert, 139
- Elastic scalability, 235
- Emctl, 43
- Emerson, Ralph Waldo, 620
- Enqueue, 150, 209
- Enqueue activity, 206
- Enqueue hash chains, 209
- Enqueue metrics
 - enqueue conversions, 189
 - enqueue releases, 189
 - enqueue timeouts, 189
 - exchange deadlocks, 189
 - execute count, 189
 - free buffer inspected and requested, 189
 - heap block compression, 189
 - hot buffer moved to head of LRU, 190
 - index operational metrics, 190
 - leaf node metrics, 190
 - lob metrics, 190
 - logons cumulative, 190
 - messages received/sent, 190
 - no buffer to keep pinned count, 190
 - no work-consistent read gets, 190
 - open cursors cumulative, 190
 - physical IO metrics, 190
 - prefetch metrics, 190
 - recursive metrics, 191
 - redo metrics, 191
 - rollback metrics, 191
 - rows fetched via callback, 191
 - session metrics, 191
 - sort metrics, 191
 - switch current to new buffer, 191
 - table metrics, 191
 - transaction metrics, 191
 - user metrics, 191
 - workarea executions-optimal, 191
 - write clones created in foreground, 191
- Enterprise Java bean (EJB), 333
- Enterprise manager, 19
- Enterprise manager DBConsole, 27, 29
- Entities, 274
- Entity, 279
- Entity relationship diagramming (ERD), 276
- Entity-relational diagram (ERD), 44, 72
- Equijoin, 286, 432
- Exadata, 251
- Exclusive (XCUR), 188
- Executes and transactions, 168
- Execution, 419
- Expdp/impdp, 642
- Extent, 52, 56
- Extent management, 120
- Extreme programming (XP), 268
- FAT. See File allocation table
- FBI. See Function-based index
- FD. See Functional dependency
- FORCE, See CURSOR_SHARING
- Facade pattern, 319
- Factory pattern, 319
- Feasibility study, 271
- Fielding, Roy, 376
- File allocation table (FAT), 512
- Filter security interceptor, 389

- FilterProxy, 388
- First-order logic, 282
- Fixed_SGA sub-area, 103
- Flash cache
 - creating flash drives out of, 252
 - pinning objects, 251
- Flashback query, 241
- Full index table scans, 63
- Function, 42, 68, 311
- Function-based index (FBI), 453
- Functional dependency (FD), 283, 285
- Fuzzy read. *See* read phenomena and data inconsistencies

- GA. *See* General availability
- GATHER_STATS_JOB, 246
- GIS. *See* Geographic information system.
- GV\$, 96
- Gates, Bill, 269
- General availability (GA), 321
- Geographic information system (GIS), 3
- Get and advance SCN (GAS), 187
- Get current SCN (GCS), 187
- Get recent SCN (GRS), 187
- GetTransactionIsolation, 152
- Global database name, 19
- Greenspan, Alan, 326
- Grid computing, 31, 81, 247
- Grid control management service, 19
- Grid management, 248
- Gropius, Walter, 79
- Guillemets, 355

- HA. *See* High availability
- HASH_AREA_SIZE, 106
- HBA. *See* Host bus adapter (HBA), 258
- HOST, 638
- HOT. *See* Heap organized table
- HP-UX Itanium, 15
- HP-UX PA-RISC, 15
- HR schema, 28
- HTTP, 26
- HTTPS, 26, 384
- HTTPS channel, 388
- Handler mapping, 350
- Hardware sizing, 322
- Hash join, 437
- Heap organized table (HOT), 58–59
- Heap-organized versus index-organized, 485
- Heartbeat, 234
- Hibernate, 368
- Hibernate.cfg.xml, 372–373
- Hierarchical SQL, 445
- High availability (HA), 81, 235
- Host bus adapter (HBA), 258
- Hubbard, Elbert, 477

- I/O calibration, 252
- IDE (integrated development environment), 43
- IDENTIFIED BY, 42
- IMDB. *See* In-memory database
- IMU metrics, 186
- IO stats, 197
 - file, 198
 - tablespace, 198
- IOT. *See* Index-organized table
- Implementation, 315
- In memory undo latch, 208
- In-Memory Database Cache, 2
- In-memory database (IMDB), 263
- Index, 57
 - adding, 313
 - bitmap, 454, 456
 - composite, 63
 - compressed-composite, 455
 - covering, 531
 - dense versus sparse, 64
 - function-based (FBI), 232
 - function-based (FBI), 67
 - invisible, 254
 - one-dimensional versus multi-dimensional, 64
 - online creation and rebuild, 231
 - reverse key, 455
 - sorted versus unsorted, 63
 - type, 57
 - unique, 456
 - unique versus non-unique, 63
 - unsorted, 456
 - zero-size unusable, 254
- Index fast full scan, 427
- Index range scan, 63, 427
- Index skip scan, 427

- Index unique scan, 427
- Index-organized table (IOT), 58–59, 452
- Indexing
 - alternate or secondary indexes, 63
 - foreign key, 63
 - non-key columns, 63
 - primary key, 62
 - rules of thumb, 450
- Information engineering (IE) format, 277
- Init.ora parameters, 224
- Initialization parameters, 21
- Inner join, 435
- Inode locking, 509
- Installing Oracle 11g client software, 28–31
- Installing Oracle software, 14–28
- Instance activity stats, 185, 196
- Instance efficiency, 129
 - buffernowait% and buffer hit%, 169
 - in-memory sort%, 169
 - library hit%, 169
 - % non-parse CPU, 170
 - parse CPU to parse elapsed%, 170
 - redonowait%, 169
 - soft parse%, 169
- Instance recovery stats, 200
- InstantClient, 29
- InterMedia, 229
- Internet connect
 - high-speed, 259
- Intuitiveness versus efficiency, 35
- Inversion of Control (IoC), 335
- IoC. See Inversion of control
- Isolation. See ACID properties of transactions
- Isql, 40

- JDBC
 - case study, 152
- JDBC. See Java database connectivity
- JFS. See Journaling file system
- Java, 11
 - AQ API, 228
 - JMS API, 229
- Java database connectivity (JDBC), 34, 47
- Java pool, 102
- Job queue coordinator (CJQ0), 85
- Jobs, Steve, 269
- Join algorithm, 437
- Join conditions, 432
- Join order, 436
- Joint development, 269
- Journaling, 511
- Journaling File System (JFS), 511
- Json, 384

- KD. See Key dependency
- Keep pool, 103
- Key
 - compound, 291, 293
 - compound versus composite, 283
 - foreign, 283
 - foreign key (FK), 61
 - primary key (PK), 61
 - secondary or alternative, 283
 - simple, 283, 291, 293
 - unique key (UK), 61
- Key dependency (KD), 293
- Key-non-key dependency,
 - See BCNF
- Keytool, 329
- Kyte, Tom, 89, 104, 261, 502

- LGWR). See Log writer
- LINESIZE, 639
- LOB
 - in-line, 66
 - out-line, 66
- LOB. See Large object
- LONG, 66
- LONG RAW, 66
- LRU. See Least recently used
- LUN. See Logic unit number
- LVM. See Logical volume manager
- Large object (LOB), 64–65
- Large pool, 102
- Latch, 149
- Latch activity, 208, 210
- Latch miss sources, 214
- Latch sleep breakdown, 213
- Latch statistics, 208
- Lean development, 269
- Least recently used (LRU), 87
- Library cache, 88, 209
- Library cache activity, 219
- Library cache pin, 209

- Life of a SQL statement in Oracle, 418
- Linux x86, 15
- Listener, 27
 - configuring, 18
- Load balancing
 - client, 233
 - connection, 233
- Load profile, 167
- Lock
 - automatically acquired, 148
 - conversion, 149
 - DDL (dictionary locks), 146
 - DML (data locks), 146
 - escalation, 149
 - internal, 146
 - row-level locking, 146
- Log buffer, 167
- Log writer (LGWR), 84
- Logic unit number (LUN), 243
- Logical block, 56
- Logical volume manager (LVM), 506
- Logons current, 196
- Lookup
 - passive, 333
- Lookup service, 333
- LookupStrategy, 398
- Lossless, 286
- Lowell, Amy, 431

- MEMORY_MAX_TARGET, 112
- MEMORY_TARGET, 112
- MFT. See Master file table
- MMAN. Memory manager
- MRU. See Most recently used (MRU)
- MVC (model-view-controller), 319
- MVC
 - architecture, 337
 - Spring, 340
 - Web form, 348
- MVD-JD conversion law, 292
- MVD. See Multi-valued dependency
- Management options, 20
- Management tasks, 20
- Margolius, Hans, 75
- Master file table (MFT), 512
- Materialized view, 68
- Max Plank, 485

- Maximum availability architecture (MAA), 237
- Memory areas
 - specialized, 79
- Memory manager (MMAN), 85
- Memory statistics, 219
- Metadata, 94
- Metadata mapping
 - Hibernate, 370
- Michelangelo, 415
- Microsoft Windows, 15
- Mirroring, 243
 - 2-way, 244
 - 3-way, 244
- Missing statistics, 594
- Moore, George, 648
- Most recently used (MRU), 87–88
- Multi-core CPUs, 261
- Multi-threaded server (MTS)
 - configuration, 83
- Multi-valued columns. See 1NF
- Multi-valued dependency (MVD), 291.
 - See also 4NF
- Multi-version concurrency control (MVCC), 145
- Multi-version read consistency, 236
- MySQL, 10
- Mysql, 40

- N-tier, 327
- NCLOB. See National character large object
- NTFS. See New technology file system
- Naming conventions, 297
- National character large object (NCLOB), 65
- Nested loop join, 437
- New features
 - 10g, 241
 - 11g, 248
 - 8i, 227
 - 9i, 233
- New technology file system (NTFS), 512
- Non-additive, 286
- Non-block OCI (Oracle call interface), 231
- Non-bulk transaction, 571

Non-repeatable read. See read phenomena
and data inconsistencies
Normalization, 280

OCI. See Oracle call interface
ODBC. See Open database connectivity
OEMJC. See Oracle enterprise manager Java
console (OEMJC)
OLAP. See Online analysis package
OLTP. See Online transaction processing
OPEN_CURSORS, 418
OPS. See Oracle Parallel Server
OPTIMIZER_DYNAMIC_SAMPLING,
240, 246
ORA-08177, 151
ORACLE
Enterprise Manager DBConsole, 42
grid control versus DB control, 31–32
static data dictionary views, 94–95
ORACLE_BASE, 41
ORACLE_HOME, 41
ORACLE_HOME environment variable, 30
ORM. See Object-relational mapping
OS, 2
OUI. See Oracle universal installer
OWI. See Oracle wait interface
Object-relational mapping (ORM), 327, 333
Observer pattern, 319
Ojdbc6.jar, 329
Online analytical processing (OLAP), 17
Online transaction processing (OLAP), 6
Open cursor current, 196
Open database connectivity, 34
Open database connectivity (ODBC), 44
Optimizer hints, 421
Optimizer plan stability, 230
Oracle
10g, 81
11g, 81
11g R2, 81
5.1, 81
6, 81
7, 81
8, 81
8i, 81
9i, 81
architecture, 79–100

database, 82
Dedicated versus shared server mode, 24
dynamic performance (V\$) views,
95–97
features of, 2–4
instance, 82
instance versus database, 11
JDeveloper, 34, 43
memory areas, 87
pre-compilers, 43
processes, 82
relational versus object-oriented, 11
server process, 26
SQL Developer, 34, 43
V2, 80
V3, 80
V4, 80
V5, 81
version history, 80–81
Oracle 10g memory management,
101–110
Oracle 11g R2, 15
Oracle Isolation level, 145
Oracle Net, 237
Oracle Parallel Server (OPS), 81, 230
Oracle Server, 79
Oracle Spatial, 229
Oracle block, 56
Oracle call interface (OCI), 29
Oracle call interface(OCI), 43
Oracle client software, 28
Oracle clusterware software, 259
Oracle enterprise manager Java console
(OEMJC), 29, 34, 36
Oracle listener, 18
Oracle page, 56
Oracle partitioning option, 17
Oracle universal installer (OUI), 15
Oracle wait interface (OWI), 127
Oracle.ODCI, 229
Oracle.xml.parser API, 229
Ordinality, 277
Outer join, 435

PAGESIZE, 639
PATH, 30
PFILE, 93

- PGA Aggr
 - summary, 201
 - target histogram, 202
 - target stats, 202
- PGA memory advisory, 203
- PGA sizing, 106
- PGA. See Program global area
- PGA_AGGREGATE_TARGET, 102, 106, 106
- PJNF. See Projection-join normal form.
 - See also 5NF
- PL/SQL, 10
- PL/SQL native compilation, 2
- PL/pgSQL, 10
- PLUSTRACE, 641
- PMON. See Process monitor
- POJO. See Plain old Java object
- PSPO. See Process spawner
- Package, 42
- Parallel processing, 261
- Parameter, 94
 - dynamic, 92
 - OS-dependent, 91
- Parent and child latch statistics, 215
- Parse
 - hard, 168
- Parsing
 - hard, 419
 - soft, 419
- Partial dependency, See 2NF
- Partitioning, 3
 - composite, 232
 - hash, 232
 - interval, 252
 - list, 241
 - range, 232
- Password, 20, 28, 41
- Patterns
 - behavioral, 319
 - creational, 319
 - structural, 319
- Performance, 6
 - IN versus EXISTS, 443
 - IN versus OR, 492
 - subquery versus join, 439
- Performance model, 479
- Performance versus scalability, 6
- Permanent tablespace, 120
- Persona, 273
- Phantom read. See read phenomena and data inconsistencies
- Ping <host>, 31
- Plain old Java object (POJO), 333, 364
- Pool size
 - shared, 167
- PostgreSQL, 10
- Principal, 387
- Process memory summary, 219
- Process monitor (PMON), 83
- Process spawner (PSPO), 85
- Processes
 - specialized, 79
- Processor
 - multi-core, 262
 - single-core, 262
- Program global area (PGA), 87–88
- Projection-join normal form (PJNF).
 - See also 5NF
- Prototype development, 269
- Proxy pattern, 319
- Psq, 40
- Publish/subscribe pattern, 319
- Query statistics
 - actual operation-level, 239
- Queues, 130
- Queuing node, 478
- RAC storage options
 - ASM, 259
 - CFS (Cluster file system), 259
 - LVM (logical volume manager), 259
 - OCFS (Oracle CFS), 259
 - raw devices, 259
- RAC. See Real Application Cluster
- RAIDs. See Redundant array of inexpensive disks
- RBO. See Rule-based optimizer (RBO)
- READ COMMITTED, 151, 151, 152
- READ ONLY, See Oracle isolation level
- RECO. See Recoverer (RECO)
- REST. See REpresentational state transfer
- RESTful constraints
 - cacheable, 377

- client-server, 377
- code on demand, 378
- stateless, 377
- transparency, 378
- uniform interface, 378
- RESTful interface
 - design principles, 378
- RESULT_CACHE, 250
- REpresentational state transfer (REST), 376
- RMI. *See* Remote method invocation
- ROI analysis, 270
- RSL (Relational Software, Inc), 80
- RTM. *See* Release to market
- Ratio-based versus OWI-based Oracle
 - performance tuning methodologies, 128
- Raw devices, 242
- Read
 - logical, 167
 - physical, 167
- Read Committed, *See* Oracle isolation level
- Read Uncommitted, *See* Oracle isolation level
- Read consistency
 - statement-level, 145
 - transaction-level, 145
- Read phenomena and data
 - inconsistencies, 143
- Real Application Cluster (RAC), 2, 16, 234, 258
- Recoverer (RECO), 84
- Recovery options, 21, 23
- Recursive calls %, 168
- Recycle pool, 103
- Redo Apply, 238
- Redo allocation, 209
- Redo log buffers, 103
- Redo log groups, 23, 119
- Redo logs, 124
- Redo size, 167
- Redundancy, 281
- Redundant array of inexpensive disks (RAIDs), 123
- Referential integrity with foreign keys, 71–73
- Reflexivity. *See* Armstrong’s Axioms
- Relation, 283, 283
- Relation theory, 282
- Relationship, 279, 283
- Release to market (RTM), 321
- Remote method invocation (RMI), 333
- Renaming columns and constraints, 241
- Repeatable Read, *See* Oracle isolation level
- Repeating group, *See* 1NF
- Requirements gathering, 272
- Resource, 150, 376
- Resource limit stats, 224
- Result cache
 - server, 250
- Result definition, 419
- Result description, 419
- Result fetch, 419
- Result processing, 420
- Reverse engineer, 44, 46
- Richter, 636
- Rollback segments, 119
- Rollback transactions %, 168
- Row cache objects, 209
- Row locking, 236
- Rowids
 - logical, 232
- Rows per sort, 168
- Rule-based optimizer (RBO), 81, 417
- Rule-based versus cost-based, 420
- Runtime, 30
- Russell, Bertrand, 449
- SAN storage, 620
- SAN. *See* Storage area network
- SCN metrics, 186
- SCUR. *See* Shared current
- SECUREFILE, 67
- SELECT . . . FOR UPDATE statement, 140
- SERIALIZABLE, 151
- SERVICE_NAME, 38
- SESSION_CACHED_CURSORS, 418
- SGA breakdown difference, 221
- SGA memory summary, 220
- SGA sizing, 104
- SGA target advisory, 204
- SGA. *See* System global area
- SGA_TARGET, 104, 199

- SID. See System Identifier
- SIMILAR. See CURSOR_SHARING
- SMON. See System monitor
- SOBA. See Secure online backing application
- SORT_AREA_SIZE, 106
- SPFILE, 93
- SPOOL, 639
- SQ – Sequence number enqueue.
 - See Enqueue
- SQL
 - SQL-86, 10
 - SQL:2008, 10
- SQL Apply, 238
- SQL Developer, 29
- SQL PL, 10
- SQL Plus, 29
- SQL Server, 10
- SQL optimization engine, 261
- SQL ordered
 - by CPU time, 180
 - by elapsed time, 179
 - by executions, 182
 - by gets, 180
 - by parse calls, 183
 - by reads, 181
 - by sharable memory, 183
 - by version count, 183
- SQL plan management, 253
- SQL statistics, 178
- SQL text
 - complete list of, 184
- SQL trace, 489
- SQL tuning, 431
- SQL tuning set, 247
- SQL*Net roundtrips, 186
- SQL*PLUS
 - Using, 40
- SQL/CLI, 12
- SQL/Foundation, 12
- SQL/Framework, 12
- SQL/JRT, 12
- SQL/MED, 12
- SQL/OLB, 12
- SQL/PSM, 12
- SQL/Schemata, 12
- SQL/XML, 12
- SSD (solid state devices), 251
- SSL certificate, 384
- STATISTIC#, 163
- STATISTICS_LEVEL, 104
- STATSPACK, 162, 162
- STAT_ID, 163
- SYS_B_n, 184
- Sample Schema
 - Human resources (HR) Schema, 53
 - Information exchange(IX) Schema, 53
 - Online catalog (OC) Schema, 53
 - Order entry (OE) Schema, 53
 - Product media (PM) Schema, 53
 - Sales history (SH) Schema, 53
- Sample Schemas, 21
- Sample schema HR, 153
- Scalability, 6
- Scaling-down, 6
- Scaling-out or horizontal scaling, 6
- Scaling-up, 6
- Scaling-up or vertical scaling, 6
- Schema, 39
- Schema diagram for SOBA, 310
- Schema objects
 - changing, 308
- Schema user
 - creating, 299
- Scott, 52
- Scrum, 268
- Secure online backing application
 - (SOBA), 326–414
- Secured-annotations, 389
- Security, 39, 314
- Security provider, 384
- Segment, 52, 56
 - by buffer busy waits, 217
 - by ITL waits, 217
 - by logical reads, 215
 - by physical reads, 216
 - by row lock waits, 217
 - statistics, 215
- Self join, 434
- Semijoin, 436
- Sequence
 - creating, 312
- Serializable, See Oracle isolation level
- Service demands, 130

- Service wait class stats, 178
- Services management console, 26
- Session allocation, 210
- Session cursor cache count, 196
- Session idle bit, 210
- SetAutoCommit, 152, 152
- Shadow processes, 83
- Shared architecture, 90
- Shared current (SCUR), 188
- Shared pool, 87, 88, 102
- Shared pool advisory, 204
- Shared pool statistics, 170
- Shared storage, 258
- Simple join, 435
- Simplicity, 36
- Soba-security.xml, 388
- Socket, 2
- Software install and cloning, 248
- Software stack setup, 329
- Solaris, 15
- Solaris (SPARC), 15
- Sorted merge join, 437
- Sorts, 168
- Spatial, 3
- Spring ACL, 395–398
- Spring MVC Web form, 358
- Spring controllers, 353
- Spring framework, 333
- Sprint, 268
- Sqlcmd, 40
- Standard versus flavored SQLs, 10
- Standby database
 - logical, 238
 - physical, 238
- Statistics, 62
 - cumulative, 163
 - operating system, 176
 - performance, 162
 - service, 177
- Storage, 39, 62
- Storage area network (SAN), 122
- Stored procedure, 68
- Streams pool, 102
- Streams pool advisory, 205
- Streams statistics, 222
- Strindberg, August, 633
- Stripping, 243
- Subqueries
 - tuning, 437
- Superkey
 - trivial, 282
- Synonym, 42, 68, 307
 - creating, 312
- System, 273
- System global area (SGA), 87–88
- System identifier (SID), 19
- System monitor (SMON), 83
- T-SQL, 10
- TCL. See Transaction control language
- TT – Temporary table enqueue. See Enqueue
- TX – Transaction enqueue. See Enqueue
- Table, 42, 57
 - clustered, 58
 - partitioned, 58
- Table access by index rowid, 427
- Table full scan, 426
- Table lock mode
 - exclusive (X), 148
 - row exclusive (RX), 148
 - row share (RS), 147
 - share (S), 148
 - share row exclusive (SRX), 148
 - summary of, 148
- Tablespace, 39, 52, 56, 117, 119
 - checking, 644
 - creating, 298
 - locally managed, 230
 - online read-only, 231
- Temporary table, 231, 493
- Temporary tablespace, 121
- Testing
 - functional, 320
 - integration, 320
 - performance and scalability, 321
 - unit, 320
- Testing process, 319
- Tharp, Twyla, 116
- Throughput, 6, 478
- Throughput dynamics, 595
- Time model statistics, 173
- Time series, 230
- Timing command, 641
- Tkprof, 489

- Tnsnames.ora, 30, 37–38, 40, 44, 637
- Tnsping, 38
- Tnsping<connect_string>, 31
- Tonnelle, Alfred, 101
- Top 5 timed events, 170
- Transaction and concurrency models, 260
- Transaction control language (TCL), 10
- Transitive dependency, *See* 3NF
- Transitivity. *See* Armstrong's Axioms
- Transparency, 236
- Trigger, 68, 307, 311
 - fire_time, 70
 - ON table_name, 70
 - trigger_event, 70
- Tuples, 283

- UEL. *See* Unified expression language
- UFS. *See* Unix file system
- UML format, 279
- UML. *See* Unified modeling language
- UTLBSTAT, 162
- UTLETSAT, 162
- Undo global data, 210
- Undo statistics, 207
- Undo tablespace, 121
- Unicode, 24
- Unified expression language (UEL), 336
- Unified modeling language (UML), 273
- Union. *See* Armstrong's Axioms
- Unix file system (UFS), 511
- Unsecure dbconsole, 43
- User calls and logons, 168
- User gesture, 347
- User interface model, 274
- User processes, 83
- User story, 273
- User versus Schema, 52, 55–56

- V\$ACTIVE_SESSION_HISTORY, 133
- V\$EVENT_HISTOGRAM, 134
- V\$EVENT_NAME, 133
- V\$SESSION_EVENT, 132
- V\$SESSION_WAIT, 131
- V\$SESSION_WAIT_CLASS, 133
- V\$SESSION_WAIT_HISTORY, 133
- V\$SQL_PLAN, 425
- V\$SQL_PLAN_STATISTICS, 428
- V\$STATNAME, 163
- V\$SYSSTAT, 163
- V\$SYSTEM_EVENT, 132
- V\$SYSTEM_WAIT_CLASS, 133
- V\$bgsprocess, 86
- V\$process, 86
- V\$version, 86
- V\$views, 86
- VCPU, 261
- VISIO, 44
- Variable, 92
- Variable binding, 419
- Veritas, 511
- Versatility, 36
- View, 42, 68
 - adding, 312
- Virtual columns, 254
- Vision, 269
- Visual image retrieval, 230
- Volume
 - logic, 506
 - physical, 506
- Von Goethe, Johann Wolfgang, 492

- WAIT_CLASS, 132
- WORKAREA_SIZE_POLICY, 106
- Wait chain, 130
- Wait class, 174
- Wait event, 128, 130, 174
 - background, 176
 - classification, 131
- Wait event statistics, 172
- Wait statistics, 206
- Web.xml, 387
- Whitehead, Alfred North, 161
- Windows services management
 - snap-in, 26
- Write
 - physical, 167
- Write-sync daemon, 510

- XCUR. *See* Exclusive
- XML DB, 3
- XP. *See* Extreme programming
- Xserve RAID, 620