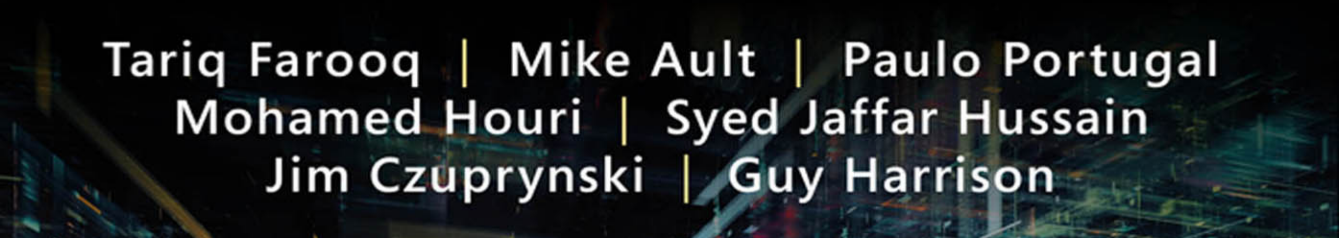
An Expert Guide for Solving Complex Oracle Database Problems

Oracle Database

Problem Solving and Troubleshooting Handbook

Tariq Farooq | Mike Ault | Paulo Portugal
Mohamed Houri | Syed Jaffar Hussain
Jim Czuprynski | Guy Harrison



Covers versions **11*g*** and **12*c***

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Oracle Database Problem Solving and Troubleshooting Handbook

This page intentionally left blank

Oracle Database Problem Solving and Troubleshooting Handbook

Tariq Farooq
Mike Ault
Paulo Portugal
Mohamed Houri
Syed Jaffar Hussain
Jim Czuprynski
Guy Harrison

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data
Names: Farooq, Tariq, author.
Title: Oracle database problem solving and troubleshooting handbook / Tariq
 Farooq [and 6 others].
Description: Boston : Addison-Wesley, [2016] | Includes indexes.
Identifiers: LCCN 2016005438 | ISBN 9780134429205 (pbk. : alk. paper)
Subjects: LCSH: Oracle (Computer file) | Relational databases. | SQL
 (Computer program language)
Classification: LCC QA76.9.D3 F358 2016 | DDC 005.75/65—dc23
LC record available at http://lccn.loc.gov/2016005438

Copyright © 2016 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

Figures 15.1, 15.2, 15.3, 15.4, 15.5, 15.6, 15.7, 15.14, 15.17, 16.1, 16.2, 16.6, and 16.7 from Farooq, Tariq; Kim, Charles; Vengurlekar, Nitin; Avantsa, Sridhar; Harrison, Guy; Hussain, Syed Jaffar; *Oracle Exadata Expert's Handbook, 1st Ed.*, © 2016. Reprinted and electronically reproduced by permission of Pearson Education, Inc., New York, NY.

Various screen shots and illustrations of Oracle products are used with permission. Copyright © 1995–2015 Oracle and/or its affiliates. All rights reserved.

ISBN-13: 978-0-13-442920-5
ISBN-10: 0-13-442920-6
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, April 2016

Contents

Preface xix

Acknowledgments xxi

About the Authors xxv

About the Technical Reviewers and Contributors xxix

Chapter 1 Troubleshooting and Tuning LOB Segment Performance 1

Introduction to the LOB Datatype 1

- Fixing a LOB Problem: A Real-World Example 2
- Another Real-World Example: HW Resolution 4
- BASICFILE LOB Issues: Toward a More Perfect Fix 8

BASICFILE versus SECUREFILE LOBs 8

- LOB New and Old Type Differences 9
- Migrating BASICFILE LOBs to SECUREFILE LOBs 12

The Impact of PCTFREE on LOBs 14

Overcoming Poor INSERT Performance 17

Summary 17

Chapter 2 Overcoming Undo Tablespace Corruption 19
Overview of Undo Management 19
The Importance of UNDO_RETENTION 20
Tuning UNDO_RETENTION 21
DTP, XA, and Rollback Segments 22
Other Unusual Rollback and Undo Segment Issues 24
Recovering from Undo Tablespace Corruption 24
Preventing, Detecting, and Repairing Corruption 24
Handling Memory Corruption 26
Handling Logical Corruption 28
Overcoming Media Corruption 29
Summary 32

Chapter 3 Handling GC Buffer Busy Wait Events 35
Overview of Buffer Busy Wait Events 35
Leveraging the ORAchk Utility 36
Installing ORAchk 37
Results of ORAchk Execution: Sample Output 38
Isolating GC Buffer Busy Waits 40
Using ADDM to Find Event Information 41
Using AWR to Find Event Information 41
Using ASH to Find Event Information 43
Isolating GC Buffer Busy Wait Event Issues 45
Using ASH Views to Find Waiting Sessions 45
Quickly Isolating Performance Bottlenecks 47
Fixes for GC Buffer Busy Waits 49
Summary 50

Chapter 4 Adaptive Cursor Sharing 51
ACS Working Algorithm 52
Bind Sensitiveness with Range Predicate 52
Bind Sensitiveness with Equality Predicate and Histogram 55
Bind Sensitiveness with Partition Keys 56
ACS in Action 58

ACS Bind-Awareness Monitoring 61

BUCKET_ID and COUNT Relationship 62

Marking Cursors Bind Aware 66

The Bind-Aware Cursor 73

A Practical Case 76

Summary 81

Chapter 5 Stabilizing Query Response Time Using SQL Plan Management 83

Getting Started 83

Creating a SQL Plan Baseline 87

Capturing Plans Automatically 87

Loading Plans from the Cursor Cache 90

Faking Baselines 92

Oracle Optimizer and SPM Interaction 96

When the CBO Plan Matches the SQL Plan Baseline 96

When the CBO Plan Doesn't Match the SQL Plan Baseline 99

When SQL Plan Baseline Is Not Reproducible 104

SQL Plan Baseline Reproducibility 108

Renaming the Index 109

Changing the Index Type 111

Adding Trailing Columns to the Index 112

Reversing the Index 113

NLS_SORT and SQL Plan Baseline Reproducibility 114

ALL_ROWS versus FIRST_ROWS 117

Adaptive Cursor Sharing and SPM 122

ACS and SPM in Oracle 11*g* Release 11.2.0.3.0 123

ACS and SPM in Oracle Database 12*c* Release 12.1.0.1.0 128

Summary 131

Chapter 6 DDL Optimization Tips, Techniques, and Tricks 133

DLL Optimization Concept 133

The DDL Optimization Mechanism 136

Table Cardinality Estimation 137

C_DDL Column in a Virtual Column 139
C_DDL Column in a Column Group Extension 140
When the Default Value of C_DDL Changes 142
C_DDL Column and Indexes 145
DDL Optimization for NULL Columns 147
Summary 152

Chapter 7 Managing, Optimizing, and Tuning VLDBs 153
Overview of Very Large Databases 153
Optimal Basic Configuration 154
Data Warehouse Template 154
Optimal Data Block Size 155
Bigfile Tablespaces 156
Adequate SGA and PGA 157
Temporary Tablespace Groups 158
Data Partitioning 158
Index Partitioning: Local versus Global 159
Data Compression 159
Table Compression 160
Heat Map and Automatic Data Optimization 160
Advanced Index Partition Compression 162
VLDB Performance Tuning Principles 162
Real-World Scenario 162
Limiting the Impact of Indexes on Data Loading 164
Maximizing Resource Utilization 165
Gathering Optimizer Statistics 166
Incremental Statistics Synopsis 166
Gathering Statistics Concurrently 168
Setting the ESTIMATE_PERCENT Value 170
Backup and Recovery Best Practices 170
Exadata Solutions 171
Utilizing a Data Guard Environment 172
Summary 172

Chapter 8 Best Practices for Backup and Recovery with Recovery Manager 173
A Perfect Backup and Recovery Plan 173
An Overview of RMAN 174
Tips for Database Backup Strategies 175
Full Backups and Incremental Backups 176
Compressed Backups 176
Incremental Backups 177
Faster Incremental Backups 177
Rewinding in Oracle Flashback Technology 178
Disk-Based Backup Solutions 179
Recover Forward Forever 180
Validating RMAN Backups 186
Backup Optimization and Tuning 187
Tuning Disk-Based Backup Performance 189
Using RMAN for RAC Databases 189
Retaining Data in a Recovery Catalog 191
Having a Robust Recovery Strategy 192
Leveraging the Data Recovery Advisor 193
Summary 194

Chapter 9 Database Forensics and Tuning Using AWR Analysis: Part I 197
What Is AWR? 197
Knowing What to Look For 199
Header Section 199
Load Profile 201
Instance Efficiencies 202
Shared Pool Memory 203
Wait Events 203
Load Average 206
Instance CPU 207
Memory Statistics 207
RAC-Specific Pages 208
RAC Statistics (CPU) 208

Global Cache Load Statistics 209
Global Cache and Enqueue Services 209
Cluster Interconnects 210
Time Model Statistics 211
Operating System Statistics 212
Foreground Wait Events 213
Background Wait Events 214
Wait Event Histograms 215
Service-Related Statistics 216
The SQL Sections 217
Total Elapsed Time 218
Total CPU Time 218
Total Buffer Gets 218
Total Disk Reads 219
Total Executions 219
Parse Calls 219
Shareable Memory 220
Version Count 220
Cluster Wait Time 220
Instance Activity Statistics 221
Consistent Get Statistics 224
DB Block Get Statistics 224
Dirty Block Statistics 224
Enqueue Statistics 224
Execution Count 225
Free Buffer Statistics 225
Global Cache (GC) Statistics 225
Index Scan Statistics 226
Leaf Node Statistics 226
Open Cursors 226
Parse Statistics 226
Physical Read and Write Statistics 227
Recursive Statistics 229

Redo-Related Statistics 229
Session Cursor Statistic 229
Sort Statistics 230
Summed Dirty Queue Length 230
Table Fetch Statistics 230
Transaction Rollback 231
Undo Change Vector Statistic 231
User Statistics 232
Work Area Statistics 232
Instance Activity Statistics—Absolute Values 232
Instance Activity Statistics—Thread Activity 233
Summary 233

Chapter 10 Database Forensics and Tuning Using AWR Analysis: Part II 235
Tablespace I/O Statistics 235
Buffer Pool Statistics 237
Buffer Pool Statistics 238
Instance Recovery Statistics 239
Buffer Pool Advisory Section 239
PGA Statistics 240
PGA Aggregate Summary 241
PGA Aggregate Target Statistics 242
PGA Aggregate Target Histogram 242
PGA Memory Advisor 243
Shared Pool Statistics 244
Other Advisories 245
SGA Target Advisory 246
Streams Pool Advisory 246
Java Pool Advisory 247
Buffer Waits Statistics 247
Enqueue Statistics 248
Undo Segment Statistics 250
Latch Statistics 251
Latch Activity 253

Latch Sleep Breakdown 253
Latches and Spin Count 254
Latch Miss Sources 254
Mutex Sleep Summary 255
Parent and Child Latches 255
Segment Access Areas 255
Library Cache Activity Sections 257
Dynamic Memory Components Sections 260
Process Memory Sections 262
Process Memory Summary 263
SGA Memory Summary 264
SGA Breakdown Difference 264
Streams Component Sections 264
Resource Limits Statistics 266
Initialization Parameter Changes 267
Global Enqueue and Other RAC Sections 268
Global Enqueue Statistics 271
Global CR Served Statistics 271
Global Current Served Statistics 271
Global Cache Transfer Statistics 271
Global Cache Transfer Times 272
Global Cache Transfer (Immediate) 272
Global Cache Times (Immediate) 272
Interconnect Ping Latency Statistics 272
Interconnect Throughput by Client 273
Interconnect Device Statistics 273
Summary 273

Chapter 11 Troubleshooting Problematic Scenarios in RAC 275
Troubleshooting and Tuning RAC 276
Start with ORAchk 276
Employ the TFA Collector Utility 276
Utilize the Automatic Diagnostic Repository 276
Check the Alert and Trace Log Files 277

Employ the Three A's 277
Check the Private Cluster Interconnect 277
Enable Tracing and Inspect the Trace Logs 278
Utilize the Cluster Health Monitor 278
Miscellaneous Tools and Utilities 278
Useful My Oracle Support Resources 278
A Well-Oiled RAC Ecosystem 279
Maximum Availability Architecture 279
Optimal and Efficient Databases in RAC 280
Troubleshooting RAC with OEM 12*c* 282
Utilities and Commands for Troubleshooting 283
Summary 288

Chapter 12 Leveraging SQL Advisors to Analyze and Fix SQL Problems 289
OEM 12*c*—SQL Advisors Home 290
SQL Tuning Advisor 290
Running SQL Tuning Advisor in OEM 12*c* 291
Running SQL Tuning Advisor Manually in SQL*Plus 294
SQL Access Advisor 295
Running SQL Access Advisor in OEM 12*c* 295
Running SQL Access Advisor Manually in SQL*Plus 298
SQL Repair Advisor 300
SQL Performance Analyzer 301
Summary 302

Chapter 13 Extending Data Pump for Data and Object Migration 303
Using Data Pump 303
Copying Objects 304
Data Pump Modes 305
Working with Private and Public Objects 306
Saving and Restoring Database Links 307
Exporting Public Database Links and Synonyms 307
Verifying Content of the Export Dump File 308
Finding Valid INCLUDE and EXCLUDE Values 309

Exporting Subsets of Data 310
Changing Object Properties 313
Importing Partitioned Tables as Nonpartitioned 313
Importing Table Partitions as Individual Tables 313
Masking Data 314
Renaming Tables or Different Tablespaces 314
Using Default Storage Parameters 314
Resizing Tablespaces during Import 315
Consolidating Multiple Tablespaces 315
Using PL/SQL API with Data Pump 317
Monitoring and Altering Resources 319
Improving Performance 320
Upgrading Databases 321
Summary 322

Chapter 14 Strategies for Migrating Data Quickly between Databases 323
Why Bother Migrating? 324
Determining the Best Strategy 324
Real-Time versus Near Real-Time Migration 325
Read-Only Tolerance 325
Reversibility 325
Considering What Data to Migrate 326
Data Migration Methods 327
Transactional Capture Migration Methods 327
Nontransactional Migration Methods 330
Piecemeal Migration Methods 345
Summary 352

Chapter 15 Diagnosing and Recovering from TEMPFILE I/O Issues 353
Overview of Temporary Tablespaces 353
Read-Only Databases 354
Locally Managed Temporary Tablespaces 354
Temporary Tablespace Groups 355
Global Temporary Tables 356

Correcting TEMPFILE I/O Waits 359
Undersized PGA 359
Inappropriate TEMPFILE Extent Sizing 364
Inappropriate Use of GTTs 364
Summary 365

Chapter 16 Dealing with Latch and Mutex Contention 367
Overview of Latch and Mutex Architecture 367
What Are Latches? 368
What Are Mutexes? 370
Latch and Mutex Internals 370
Measuring Latch and Mutex Contention 371
Identifying Individual Latches 372
Drilling into Segments and SQLs 373
Latch and Mutex Scenarios 375
Library Cache Mutex Waits 375
Library Cache Pin 378
Shared Pool Latch 378
Cache Buffers Chains Latch 379
Other Latch Scenarios 381
Intractable Latch Contention 383
Fine Tuning Latch Algorithms 383
Summary 385

Chapter 17 Using SSDs to Solve I/O Bottlenecks 387
Disk Technologies: SSD versus HDD 388
The Rise of Solid-State Flash Disks 389
Flash SSD Latency 389
Economics of SSD 390
SLC, MLC, and TLC Disks 391
Write Performance and Endurance 392
Garbage Collection and Wear Leveling 393
SATA versus PCIe SSD 395
Using SSD Devices in an Oracle Database 395

The Oracle Database Flash Cache 395
Free Buffer Waits 396
Configuring and Monitoring DBFC 398
Using the FLASH_CACHE Clause 399
Flash Cache Performance Statistics 400
Comparing SSD Options 402
Indexed Reads 403
OLTP Read/Write Workload 403
Full Table Scan Performance 404
SSD Native Caches and Full Table Scans 405
Disk Sort and Hash Operations 406
Redo Log Optimization 409
Storage Tiering 410
Using Partitions to Tier Data 410
Flash and Exadata 414
Creating Flash-Backed ASM Disk Groups on Exadata 416
Summary 418

Chapter 18 Designing and Monitoring Indexes for Optimal Performance 421
Types of Indexes 421
B-Tree Indexes 422
Bitmap Indexes 425
Partitioned Indexes 427
Other Index Types 430
Multiple Indexes on Identical Columns 431
Index Performance Issues 432
Index Statistics 432
The Impact of a Low Clustering Factor 435
Operational Considerations for Indexes 436
Hiding Unselective Indexes 439
Index Performance Issues in RAC Databases 441
Summary 442

Chapter 19 Using SQLT to Boost Query Performance 445
Installing SQLT 446
Using the XTRACT Method 447
Using the XECUTE Method 448
Leveraging Other SQLT Methods 451
A Real-World Example 452
Summary 453

Chapter 20 Dealing with XA Distributed Transaction Issues 455
Repairing Common Distributed Transaction Issues 456
Repairing Ghost Distributed Transactions 457
Information Exists, but Transaction Missing 457
ORA-1591 Has No Corresponding Information 458
Transaction Hangs after COMMIT or ROLLBACK 460
Monitoring Distributed Transactions 462
Summary 464

Index 465

This page intentionally left blank

Preface

Database administrators' lives are becoming more and more challenging, and arduous work conditions are fast becoming the norm. DBAs face problems that in some cases could lead organizations and entities to potentially lose millions of dollars per minute or, in worst-case scenarios, could bring a company's database infrastructure to a grinding halt. Yes, such cases are unlikely to happen, but to avoid and avert them, DBAs had best be prepared.

The guiding principle of this book is to show you how to fix, as rapidly as possible, serious database problems that can potentially impact the production-level environment. It guides readers through the steps necessary to fix the problem at hand by examining real-life examples that could happen any day at any time in any Oracle database.

Instead of losing time trying to find the solution for a problem that is taking your database down or has already put it down, you can turn to this book for solutions to some of the biggest problems you might face. Even if you do not find the solution for your current problem here, you will learn how to quickly search for solutions on the Internet to solve your problem.

The basic idea behind this book is to offer you light in the dark when you have serious Oracle database problems in production environments. Along with general best practices, this book explores some of the top Oracle database problems and their rapid-fire solutions, explained in a simple and easy format.

Targeted for Oracle DBAs and database machine administrators (DMAs), *Oracle Database Problem Solving and Troubleshooting Handbook* will serve as a practical technical guide for performing day-to-day troubleshooting, tuning, and problem-solving of administration operations and tasks within the Oracle Database Server family.

Authored by a world-renowned, veteran-author team of Oracle ACEs, ACE Directors, and Experts, this book is intended to be a problem-solving handbook with a blend of real-world, hands-on examples and troubleshooting of complex Oracle database scenarios. This book shows you how to

- Choose the quickest path to solve large-impact problems
- Make your day more productive with reliable working techniques learned from real field experts
- Construct your own 911 plan
- Perform routine proactive maintenance to ensure stability of your environment
- Use industry standard best-practice tools and scripts to find the best and fastest solutions

In this technical, everyday, hands-on, step-by-step book, the authors aim for an audience of intermediate-level, power, and expert users of the Oracle Database Server family of products. This book covers both Oracle Database 11*g* and Oracle Database 12*c* versions of the underlying Oracle database software.

Acknowledgments

Tariq Farooq

I would like to express boundless thanks for all good things in my life to the Almighty ALLAH, the lord of the worlds, the most gracious, the most merciful.

I dedicate this book to my parents, Mr. and Mrs. Abdullah Farooq; my wonderful wife, Ambreen; my awesome kids, Sumaiya, Hafsa, Fatima, and Muhammad-Talha; and my nephews Muhammad-Hamza, Muhammad-Saad, Muhammed-Muaz, Abdul-Karim, and Ibrahim, without whose perpetual support this book would not have come to fruition. My endless gratitude to them as I dedicated almost two years of my spare time to this book, most of which was on airplanes and in late nights and weekends at home.

My heartfelt gratitude to my friends at the Oracle Technology Network (OTN), colleagues in the Oracle ACE fellowship, my coworkers, and everyone else in the Oracle community, as well as in my workplace for standing behind me in my quest to bring this project to completion, especially Dave Vitalo.

I had been thinking about writing on the Oracle troubleshooting and problem solving subject area for quite a while. The project was finally kick-started when I met Paulo Portugal in San Francisco at Oracle Open World 2013. The one thing that I am very proud of is the amazing ensemble of some of the best minds in the industry, including Oracle ACEs, ACE directors, and Ph.D.s coauthoring and technically reviewing this book from start to finish.

From inception to writing to technical review to production, authoring a book is a complex, labor-intensive, lengthy, and at times painful process; this book would not have been possible without the endless help and guidance of the awesome Addison-Wesley team. A very special thanks goes out to Greg Doench, executive editor, and all the other folks at Addison-Wesley, who stood like a rock behind this project. Kudos to the technical reviewers, book reviewers, and editorial teams at Addison-Wesley for a great job on this book.

Many appreciative thanks to my buddies, coauthors, and technical reviewers—Paulo Portugal, Mohamed Houri, Mike Ault, Jim Czuprynski, Syed Jaffar Hussain, Kamran Agayev, Anju Garg, Bert Scalzo, and Guy Harrison—for the amazing team effort that allowed us to bring this book to you, my dear reader. A special thanks to my friend and fellow Oracle ACE Director Biju Thomas for authoring Chapter 13.

Finally, I thank you, my dear reader, for joining us on this knowledge-laden journey—my sincerest hope is that you will learn from this book and that you will enjoy reading it as much as we did researching and authoring it.

Mike Ault

I would like to acknowledge Texas Memory Systems (TMS) and IBM for allowing me the freedoms to continue writing and researching Oracle-related topics.

Paulo Portugal

I devoted a lot of time working on this book, and at that time my little princess was too young to understand my preoccupation with this project. I dedicate this book to her and to my lovely wife, who has always supported me at all times and occasions.

Mohamed Houri

This book is dedicated to my parents; to my lovely daughters, Imane Sonia, Yasmine, and Selma; and to my family and friends.

Syed Jaffar Hussain

I am thankful for everything in my life to the Almighty ALLAH, the lord of the worlds, the most gracious, the most merciful. I dedicate this book to my parents, Mr. and Mrs. Saifulla; my amazing wife, Ayesha; my wonderful children, Ashfaq, Arfan, and Aahil; my brothers Sadak, Sabdar, and Noor; and my friends and colleagues. A bundle of thanks to the Addison-Wesley team for their endless help and guidance on the book. Many appreciative thanks to my coauthors and technical reviewers, Paulo Portugal, Mohamed Houri, Mike Ault, Jim Czuprynski, Kamran Agayev, Anju Garg, Bert Scalzo, and Guy Harrison, for the amazing team effort that allowed us to bring this book to you, my dear reader. A special thanks to my friend and brother Tariq Farooq for inviting me to participate in this great project.

Jim Czuprynski

I dedicate my part of this book to my dearest wife, Ruth—my helpmate, best friend, and companion for nearly four decades. Without her careful, loving guidance, proofreading skills, and infinite patience during my long nights of writing, editing, and muttered expletives as I struggled to meet deadlines, it would have simply been impossible to complete my assignments.

Guy Harrison

Thanks to Tariq for giving me the opportunity to work with such a great group of technical authors, thanks to Anju for providing excellent technical editing, and thanks to everyone at Addison-Wesley who worked on this project. Thanks as always to my family for their love and support.

Biju Thomas

I am very honored and humbled to have been invited by Tariq to be part of this outstanding project along with highly respected and sought-after authors. My sincere thanks to Tariq and to all coauthors of this book. Thanks to Kamran for all the valuable technical edits and suggestions. I appreciate all efforts by the Addison-Wesley team to make sure the book maintains its quality in content, formatting, and appearance. I am grateful to the Oracle Technology Network (OTN) and Oracle ACE program for all the support. Last but not least, thanks to my family for always standing behind me with steadfast support and encouragement.

This page intentionally left blank

About the Authors

Tariq Farooq is an Oracle technologist, architect, and problem-solver and has been working with various Oracle technologies for more than 24 years in very complex environments at some of the world's largest organizations. Having presented at almost every major Oracle conference/event all over the world, Tariq is an award-winning speaker, community leader/organizer, author, forum contributor, and tech blogger. He is the founding president of the IOUG Virtualization & Cloud Computing Special Interest Group and the BrainSurface social network for the various Oracle communities. Tariq founded, organized, and chaired various Oracle conferences, including, among others, the OTN Middle East and North Africa (MENA) Tour, VirtaThon (the largest online-only conference for the various Oracle domains), the CloudaThon & RACaThon series of conferences, and the first-ever Oracle-centric conference at the Massachusetts Institute of Technology in 2011. He was the founder and anchor/show host of the *VirtaThon Internet Radio* series program. Tariq is an Oracle RAC Certified Expert and holds a total of 14 professional Oracle certifications. Having authored more than one hundred articles, whitepapers, and other publications, Tariq is the coauthor of the *Expert Oracle RAC 12c* (Apress, 2013), *Oracle Exadata Expert's Handbook* (Addison-Wesley, 2015), and *Building Database Clouds in Oracle 12c* (Addison-Wesley, forthcoming in 2016) Oracle books. Tariq has been awarded the Oracle ACE and ACE Director awards.

Mike Ault began working with computers in 1980—following a six-year Navy enlistment in the Nuclear Navy riding submarines—programming in Basic and Fortran IV on the PDP-11 architecture in the nuclear industry. During Mike's nuclear years, he worked with PDP, IBM-PC, Osborne, and later VAX-VMS and HP architectures, as well as with the Informix and Ingres databases. Following the downturn in the nuclear industry, Mike began working with Oracle as the only DBA at the Luka, Mississippi–based Advanced Solid Rocket Motor (ASRM) project for NASA in 1990. Since 1990, Mike has worked with a variety of industries using Oracle both in-house and as a consulting talent. Mike got extensive Flash experience as the Oracle Guru for Texas Memory Systems. Mike transitioned to IBM as the Oracle Guru for the STG Flash group when IBM purchased TMS in 2012. Mike has published over two dozen Oracle-related books, including the 7.0, 8.0, 8*i* and 9*i* versions of his *Oracle Administration and Management* with Wiley, the "Oracle8 Black Book" and "Oracle DBA OCP Exam Cram" series (for versions 8 and 8*i*) with Coriolis, and multiple titles including *Oracle9i RAC* and *Oracle10g Grid & Real ApplicationClusters* with Rampant Technical Press. Mike has written articles for Oracle, Select, DBMS, Oracle Internals, and several other database-related magazines. Mike is also a highly sought-after keynote speaker and expert instructor for local, regional, and international Oracle conferences, such as GOUSERS, SEOUG, RMOUG, NYOUG, NCOUG, IOUG, OOW, ODTUG, UKOUG, and EOUG.

Paulo Portugal has more than fifteen years of IT experience as an Oracle DBA. He is an Oracle Certified Master 11*g*; an Oracle Certified Professional (9*i*, 10*g*, 11*g*, and 12*c*); an Oracle RAC 10*g* and 11*g* Certified Specialist; an Oracle DBA 10*g* Certified Linux Administrator; Oracle Exadata Implementation Certified; IBM DB2 Certified (8 and 9 "Viper"); an Oracle GoldenGate 10 Certified Implementation Specialist; an Oracle Enterprise Manager Certified Implementation Specialist; and an Oracle 11*i* Applications Database Administrator Certified Professional. Paulo is the author of the Rampant "Advanced DBMS Packages" and many articles in blogs and some magazines and websites. Paulo has maintained a regular presence on the Oracle conference and speaking circuit: Oracle Open World—San Francisco (2005, 2006, 2011, and 2013), IBM Information on Demand—Los Angeles (2006), Burleson Oracle RAC Cruise (2009), and Oracle Training in Reading—United Kingdom (2011). Currently, Paulo works as an Oracle sales consultant for Oracle Brazil. Paulo has participated in the Oracle Beta Test 11*i* project using Data Guard and is a specialist in high availability tools such as Oracle Data Guard, Oracle Streams, Oracle GoldenGate, and Oracle RAC.

Mohamed Houri has a Ph.D. in fluid mechanics (scientific computing) from the University of Aix–Marseille II, preceded by an engineer diploma in aeronautics. He has been working around the Oracle database for more than fourteen years for different European customers as an independent Oracle consultant specializing in tuning and troubleshooting Oracle performance problems. Mohamed has also worked with the Naval Architect Society of Japan on the analysis of tsunamis and breaking waves using a powerful signal analysis called Wavelet Transform. He maintains an Oracle blog and is active in the Oracle Worldwide forum and in the French equivalent. He tweets about Oracle topics at @MohamedHouri.

Syed Jaffar Hussain is an Oracle Database expert with more than twenty years of IT experience. He has been involved with several local and large-scale international banks where he designed, implemented, and managed highly complex cluster and Exadata environments with hundreds of business-critical databases. Oracle awarded him the prestigious Best DBA of the Year and Oracle ACE Director status in 2011. He also acquired industry-best Oracle credentials, Oracle Certified Master (OCM), Oracle RAC Expert, OCP DBA 8*i*, 9*i*, 10*g*, and 11*g*, in addition to ITIL expertise. Syed is an active Oracle speaker who regularly presents technical sessions and webinars at many Oracle events. You can visit his technical blog at http://jaffardba.blogspot.com. In addition to being part of the core technical review committee for Oracle technology–oriented books, he coauthored *Oracle 11g R1/R2 Real Application Clusters Essentials* (Packt Publishing, 2011), *Expert Oracle RAC 12c* (Apress, 2013), and *Oracle Exadata Expert's Handbook* (Addison-Wesley, 2015).

Jim Czuprynski is an Oracle ACE Director with more than thirty-five years of experience in information technology, serving diverse roles at several Fortune 1000 companies in those three-plus decades—mainframe programmer, applications developer, business analyst, and project manager—before becoming an Oracle DBA in 2001.

In his current role as a strategic solutions consultant for OnX Enterprise Solutions, he focuses on providing his expertise to help customers understand how to best leverage Oracle technology to solve their most difficult IT challenges. As a senior Oracle University instructor, Jim has taught Oracle core technologies, Exadata, and GoldenGate to more than two-thousand Oracle DBAs since 2005. He was selected as Oracle Education Partner Instructor of the Year in 2009.

Jim's most recent book, *Oracle Database Upgrade, Migration & Transformation Tips & Techniques* (McGraw-Hill Education, 2015), takes a nitty-gritty approach to tackling the best ways to migrate, transform, and upgrade Oracle databases to 12*c*, Exadata, and beyond. Jim continues to write a steady stream of articles that focus on the myriad facets of Oracle database administration, with more than one hundred articles to his credit since 2003 at databasejournal.com and ioug.org. Jim's blog, *Generally* . . . *It Depends,* contains his regular observations on all things Oracle. Jim is also a sought-after public speaker on Oracle Database technology features. Since 2008, he has presented topics at Oracle OpenWorld, IOUG's COLLABORATE, Hotsos Symposium, Oracle Technology Network ACE Tours, and Oracle User Group conferences around the world.

Guy Harrison is an executive director of research and development at Dell Software, where he oversees the development of database tools such as Toad and Shareplex. Guy is the author of six books on database technology, including *Next Generation Databases* (Apress, 2015), *Oracle Performance Survival Guide* (Prentice Hall, 2010), and *MySQL Stored Procedure Programming* (O'Reilly, 2006). He also writes the "Big Data notes" column for *Database Trends and Application*s (dbta.com). Guy can be found on the Internet at www.guyharrison.net, on e-mail at guy.harrison@software.dell.com, and on Twitter at @guyharrison.

About the Technical Reviewers and Contributors

Dr. Bert Scalzo is an Oracle ACE, author, speaker, consultant, and a senior product manager for database tools at Idera. Bert spent 15 years architecting DBA features for the popular Toad product line. He has three decades of Oracle database experience and previously worked for both Oracle Education and Oracle Consulting. Bert holds several Oracle Masters certifications and his academic credentials include a B.S., an M.S., and a Ph.D. in computer science, as well as an MBA. He has presented at numerous Oracle conferences and user groups, including OOW, ODTUG, IOUG, OAUG, RMOUG, and many others. Bert's areas of interest include data modeling, database benchmarking, database tuning and optimization, "star schema" data warehouses, Linux, and VMware. He has written numerous papers and blogs for such well-respected publications as the *Oracle Technology Network* (OTN), *Oracle Magazine*, *Oracle Informant*, *PC Week* (eWeek), *Dell Power Solutions Magazine*, *The LINUX Journal*, LINUX.com, *Oracle FAQ*, and *Toad World*. Bert has authored and coauthored the following books: *Oracle DBA Guide to Data Warehousing and Star Schemas* (Prentice Hall, 2003), *TOAD Handbook* (First Edition, Sams, 2003; Second Edition, Addison-Wesley, 2010), *Database Benchmarking: Practical Methods for Oracle & SQL Server* (Rampant, 2006), *Advanced Oracle Utilities: The Definitive Reference* (Rampant, 2014), *Oracle on VMware: Expert Tips for Database Virtualization* (Rampant, 2008), *Introduction to Oracle: Basic Skills for Any Oracle User* (CreateSpace, 2010), *Introduction to SQL Server: Basic Skills for Any SQL Server User* (CreateSpace, 2011), and *Toad for Oracle Unleashed* (Sams, 2015).

Anju Garg is an Oracle ACE Associate with more than thirteen years of experience in the IT industry in various roles. Anju is a certified expert in Oracle RAC, Oracle Database Performance Tuning, and SQL Statement Tuning. Since 2010, she has been involved in teaching and has trained more than a hundred DBAs from across the world in various core DBA technologies such as RAC, Data Guard, performance tuning, SQL statement tuning, and database administration. She has also conducted various Oracle University trainings. Anju was a member of the Expert Panel at SANGAM 2014 and is a regular speaker at SANGAM and OTN Yathra. She is an author at the website All Things Oracle and is passionate about learning. She has a keen interest in RAC and performance tuning. Anju shares her knowledge via her technical blog at http://oracleinaction.com.

Kamran Aghayev A. is an Oracle Certified Master (OCM), Oracle RAC Certified Expert, Oracle Certified Professional (9*i*, 10*g*, 11*g*), and Oracle ACE Director working as a DBA team head at AzerCell Telecom LLC. He is the author of *Oracle Backup and Recovery: Expert Secrets for Using RMAN and Data Pump* (Rampant, 2013) and *Study Guide for Oracle Certified Master 11g Exam: A Comprehensive Guide* (Springer-Verlag, 2016). Kamran runs a popular blog (www.kamranagayev.com) where he shares his experience, and he contributes fairly regularly to newsgroups, forums, and user-group meetings and events around the world. He is a frequent speaker and has presented in many countries, most recently the United States, Japan, Thailand, China, India, Argentina, Uruguay, Finland, and Turkey. Kamran is president of Azerbaijan Oracle User Group (AzerOUG) and delivers a class about Oracle database administration at Qafqaz University. He is also a Brazilian Jiu Jitsu (BJJ) practitioner and Abu Dhabi Cup 2013 champion.

Biju Thomas is an Oracle ACE Director, Oracle Certified Professional, and Certified Oracle Database SQL Expert. He is a principal solutions architect at OneNeck IT Solutions. Biju has been developing and administering Oracle databases since 1993 and Oracle EBS since 2006. He spends time mentoring DBAs and performance tuning and architecting Oracle solutions. He is a frequent presenter at Oracle conferences and writes articles for Oracle technical journals. Biju has authored Oracle certification books published by Sybex since Oracle 8*i*, all versions including Oracle Database 12*c* OCA. Biju blogs at www.bijoos.com/oraclenotes, and you can follow him on Twitter (@biju_thomas) and Facebook (oraclenotes) for daily Oracle Tidbits.

16

Dealing with Latch and Mutex Contention

Contention is the proverbial bottleneck: when multiple database sessions compete for limited or serialized resources, the amount of work that can be done by the database is constrained. Some forms of contention are the result of programming practices: in particular, contention for locks is usually a consequence of application design. By comparison, latches and mutexes are internal Oracle mechanisms and contention for latches can be harder to diagnose and resolve.

In this chapter, we see how latches and mutexes work and why they are a necessary part of the Oracle architecture. We then discuss how to diagnose the root causes of latch and mutex contention and explore remedies to common contention scenarios.

Overview of Latch and Mutex Architecture

Anyone who has ever worked with a relational database and particularly with Oracle is probably comfortable with the principle of database locks. Locks are an essential mechanism in any transactional multiuser database system: the Atomic, Consistent, Independent, Durable (ACID) properties of a transaction can be implemented only by restricting simultaneous changes to table data. This restriction is achieved by placing locks on modified data.

Latches and mutexes are similar to locks, but instead of restricting simultaneous access to data in Oracle tables, they restrict simultaneous access to data in Oracle shared memory. A somewhat simplistic way of thinking about this is that whereas locks prevent corruption of data on disk, latches and mutexes prevent corruption of data in shared memory.

Oracle sessions share information in the buffer cache, shared pool, and other sections of the shared memory known as the *system global area* (SGA). It's essential that the integrity of SGA memory is maintained, so Oracle needs a way to prevent two sessions from trying to change the same piece of shared memory at the same time. Latches and mutexes serve this purpose.

The very nature of latches and mutexes creates the potential for contention. If one session is holding a latch that is required by another session, then the sessions concerned are necessarily contending for the latch. Latch contention is therefore one of the most prevalent forms of Oracle contention.

Let's spend a little time going over the latch and mutex implementation in Oracle before looking at specific contention scenarios.

What Are Latches?

Latches are serialization mechanisms that protect areas of Oracle's shared memory (the SGA). In simple terms, latches prevent two processes from simultaneously updating—and possibly corrupting—the same area of the SGA.

Latches protect shared memory structures from the following situations:

- Concurrent modification by multiple sessions leading to corruption
- Data being read by one session while being modified by another session
- Data being aged out of memory while being accessed

Oracle sessions need to update or read from the SGA for almost all database operations. For example:

- When a session reads from a database file, it often stores the block into the buffer cache in the SGA. A latch is required to add the new block.
- If a block of data exists in the buffer cache, a session reads it directly from there rather than from disk. Latches are used to "lock" the buffer for a very short time while it is being accessed.
- When a new SQL statement is parsed, it is added to the library cache within the SGA. Latches or mutexes prevent two sessions from adding or changing the same SQL.
- As modifications are made to data blocks, entries are placed in a redo buffer before being written to the redo log. Access to the redo buffers is protected by

redo allocation latches. Oracle maintains arrays of pointers to lists of blocks in the buffer cache. Modifications to these lists are themselves protected by latches.

Latches and mutexes prevent these operations—and many others—from interfering with each other and possibly corrupting the SGA.

Latches typically protect small groups of memory objects. For instance, each *cache buffers chains* latch protects a group of blocks in the buffer cache—a few dozen perhaps. However, unlike locks, which can protect even a single row, latches and mutexes almost always span multiple rows and SQL statements respectively; a single latch might protect hundreds or thousands of table rows; a single mutex might protect dozens of SQL statements.

Spin Locks

Because the duration of operations against memory is very small (typically in the order of nanoseconds) and the frequency of memory requests potentially very high, the latching mechanism needs to be very lightweight. On most systems, a single machine instruction called *test and set* is used to see if the latch has already been taken (by looking at a specific memory address), and if not, it is acquired (by changing the value in the memory address). However, there may be hundreds of lines of Oracle code surrounding this "single machine instruction."

If a latch is already in use, Oracle assumes that it will not be in use for long, so rather than go into a passive wait (relinquish the CPU and go to sleep), Oracle might retry the operation a number of times before giving up and sleeping. This algorithm is called acquiring a *spin lock*. Each attempt to obtain the latch is referred to as a *latch get*, each failure is a *latch miss*, and sleeping after spinning on the latch is a *latch sleep*.

A session can awaken from a sleep in one of two ways. Either the session awakens automatically after a period of time (a timer sleep), or it can be awoken when the latch becomes available. In modern releases of Oracle, latches are generally woken by a signal rather than after waiting for a fixed amount of time. The session that waits places itself on the *latch wait list*. When another session is relinquishing the latch in question, it looks at the latch wait list and sends a signal to the sleeping session indicating that the latch is now available. The sleeping session immediately wakes up and tries to obtain the latch.

Spin Gets

Historically, all latches would repeatedly attempt to acquire a latch before relinquishing. Because latches are held for extremely short periods of time, it can make more sense to stay on the CPU and keep trying rather than to surrender the CPU and force a relatively expensive context switch. The process of repeatedly attempting to acquire the latch is known as *spinning*.

Some latches must be acquired exclusively, while others may be acquired in shared read mode. The shareable latches may still be acquired in exclusive mode should the Oracle code determine that shared access is not appropriate.

In modern Oracle (11*g* and 12*c*), attempts to acquire a latch in exclusive mode normally result in 20,000 spin attempts before going onto the latch wait list. In other circumstances (such as an exclusive mode get on a shareable latch), the process may spin only 2,000 times or (for shared mode requests, for example) spin only a couple of times or not at all.

Most of the high-volume latch requests are made in exclusive mode, so most of the time a latch miss results in 20,000 spin gets before a latch sleep occurs.

What Are Mutexes?

Originally, all Oracle shared memory serialization mechanisms were referred to as latches. Beginning in Oracle Database 10*g*, some of the mechanisms were described as mutexes—so what's the difference, and does it matter?

In computer science, a mutex (MUTual EXclusion) is defined as a mechanism that prevents two processes from simultaneously accessing a critical section of code or memory.

Oracle latches in fact represent an implementation of the mutex pattern, and nobody would have argued had Oracle originally referred to them as mutexes. Regardless of why Oracle originally decided to describe the mechanisms as latches, over time other database vendors have followed suit, and today a latch could arguably be defined as "a mutex mechanism implemented within a database server."

Although there's no definitive difference between latches and mutexes, in practice what Oracle calls mutexes are implemented by more fundamental operating system calls that have an even lower memory and CPU overhead than a latch. The primary advantage of mutexes is that there can be more of them, which allows each mutex to protect a smaller number of objects as compared to a latch.

Latch and Mutex Internals

Originally, only the developers of the Oracle software truly understood latching mechanisms, but over the years many smart people have studied and experimented on latches and mutexes. Through their work, we have come to understand at least some of these mechanisms.

Way back in 1999, Steve Adams pioneered much research into latch algorithms and published them in a small but classic book *Oracle8i Internal Services* (O'Reilly, 1999).

This book reflected our best understanding of how latches worked in the Oracle 8*i* release. However, the mechanisms have changed substantially in every release of Oracle, and today the writings of Andrey Nikolaev at http://andreynikolaev.wordpress.com probably represent our most modern understanding of latch internals.

There was a time when it was possible to have a fairly complete understanding of latch internals without being a member of Mensa. However, today the various mechanisms have become so complex and changeable that probably only a handful of people outside of Oracle Corporation (and maybe inside) have a complete grasp of the mechanisms. The rest of us are just hurting our brains trying to keep up with it all!

Luckily, it's not necessary to understand the details of latch/mutex algorithms. The root causes of latch contention typically remain constant, even while the internal algorithms are continuously being tweaked, and the solutions almost always involve alleviating these root causes rather than tweaking the internal algorithms. Those root causes generally relate to multiple Oracle sessions competing for access to memory structures in the SGA.

Measuring Latch and Mutex Contention

As with most contention scenarios, the wait interface and time model provide the best way to determine the extent of any contention that might exist. Time spent in latch or mutex sleeps is recorded in `V$SYSTEM_EVENT` and similar tables and usually is the primary indication that a problem exists.

However, be aware that the wait interface records only latch *sleeps*; latch *misses* do not result in a wait being recorded, even though they might consume CPU (if the session spins on the latch). Therefore, latch misses should be considered to be a lesser but still important aspect of latch contention.

Prior to Oracle Database 10*g*, a single `latch free` wait event was recorded for all latch sleeps. From Oracle 10g onward, certain latches now have their own event—such as `latch: cache buffers chains`. Not all latches have their own event, though, and those that do not continue to be included in the `latch free` wait.

Mutex waits are represented by waits such as `library cache: mutex X`, which represents a wait on an exclusive library cache mutex.

To break out mutex and latch waits and compare them to other high-level wait categories, we could issue a query such as that shown in Listing 16.1.

Listing 16.1 Latch Wait Times

```
SQL> WITH system_event AS
      (SELECT CASE WHEN (event LIKE '%latch%'  or event
                         LIKE '%mutex%' or event like 'cursor:%')
                   THEN event  ELSE wait_class
                   END wait_type, e.*
```

```
     FROM v$system_event e)
SELECT wait_type,SUM(total_waits) total_waits,
       round(SUM(time_waited_micro)/1000000,2) time_waited_seconds,
       ROUND(  SUM(time_waited_micro)
             * 100
             / SUM(SUM(time_waited_micro)) OVER (), 2) pct
FROM (SELECT  wait_type, event, total_waits, time_waited_micro
      FROM    system_event e
      UNION
      SELECT   'CPU', stat_name, NULL, VALUE
      FROM v$sys_time_model
      WHERE stat_name IN ('background cpu time', 'DB CPU')) l
WHERE wait_type <> 'Idle'
GROUP BY wait_type
ORDER BY 4 DESC
/

WAIT_TYPE                         TOTAL_WAITS TIME_WAITED_SECONDS    PCT
--------------------------------- ----------- ------------------- ------
CPU                                                      1,494.63  69.26
latch: shared pool                  1,066,478              426.20  19.75
latch free                             93,672              115.66   5.36
wait list latch free                      336               58.91   2.73
User I/O                                9,380               27.28   1.26
latch: cache buffers chains             2,058                8.74    .40
Other                                      50                7.26    .34
System I/O                              6,166                6.37    .30
cursor: pin S                             235                3.05    .14
Concurrency                                60                3.11    .14
library cache: mutex X                257,469                2.52    .12
```

Of course, this query reports all waits since the database first started. To get a view over a specific period of time, you would need to run the query twice and compare totals. We can also observe the ongoing state of these statistics in the Oracle Cloud Control, in third-party tools such as Toad, or by using Automatic Workload Repository (AWR) or Statspack reports.

Identifying Individual Latches

If we're lucky, the latch that is responsible for whatever latch contention exists will be identified by its specific wait event—`latch: cache buffers chains`, for instance. However, this won't always be the case; some latches are included in the general-purpose `latch free` event, and some might be recorded against the event `wait list latch free`.

The `wait list latch free` event relates to Oracle's *latch wait posting* algorithm. Oracle implements a latch wait list that allows sessions sleeping on a latch to be woken when the latch becomes available. When a session sleeps on a latch, it normally places itself on the latch wait list and is woken by the session that releases the latch. If there's heavy contention on the wait list, then the `wait list latch free` event may occur.

If the specific latch waits are being obscured by these general-purpose `latch free` events, then you may need to examine `V$LATCH`, which includes latch statistics for each specific latch. The V$LATCH view records the number of gets, misses, sleeps, and wait times for each latch. The query in Listing 16.2 interrogates this view to identify the latches with the most sleeps and wait times.

Listing 16.2 Latch Miss Statistics

```
SQL> WITH latch AS (
      SELECT name,
             ROUND(gets * 100 / SUM(gets) OVER (), 2) pct_of_gets,
             ROUND(misses * 100 / SUM(misses) OVER (), 2) pct_of_misses,
             ROUND(sleeps * 100 / SUM(sleeps) OVER (), 2) pct_of_sleeps,
             ROUND(wait_time * 100 / SUM(wait_time) OVER (), 2)
                      pct_of_wait_time
        FROM v$latch)
     SELECT *
     FROM latch
     WHERE pct_of_wait_time > .1 OR pct_of_sleeps > .1
     ORDER BY pct_of_wait_time DESC;

                               Pct of Pct of Pct of    Pct of
NAME                             Gets Misses Sleeps Wait Time
------------------------------ ------ ------ ------ ---------
cache buffers chains            99.59  99.91  70.59     89.75
shared pool                       .07    .03  16.69      7.78
session allocation                .18    .05  11.39      1.88
row cache objects                 .07    .00    .78       .24
simulator lru latch               .01    .00    .31       .18
parameter table management        .00    .00    .08       .14
channel operations parent latc    .00    .00    .16       .02
```

Drilling into Segments and SQLs

Determining the latches associated with contention is usually not enough to identify the root cause. We most likely need to identify the SQLs and segments involved.

If you have an Oracle diagnostic pack license, then you can query the Active Session History (ASH) and/or AWR tables to identify the SQLs and segments associated with particular wait conditions. The query in Listing 16.3 identifies entries in the ASH table associated with latch contention.

Listing 16.3 Finding Latch Contention with ASH

```
SQL> l
     WITH ash_query AS (
          SELECT event, program,
                 h.module, h.action,   object_name,
                 SUM(time_waited)/1000 reltime, COUNT( * ) waits,
                 username, sql_text,
                  RANK() OVER (ORDER BY COUNT(*) DESC) AS wait_rank
           FROM  v$active_session_history h
           JOIN  dba_users u  USING (user_id)
           LEFT OUTER JOIN dba_objects o
                ON (o.object_id = h.current_obj#)
           LEFT OUTER JOIN v$sql s USING (sql_id)
          WHERE (event LIKE '%latch%' or event like '%mutex%')
```

```
 13       GROUP BY event,program, h.module, h.action,
 14           object_name,  sql_text, username)
 15  SELECT event,module, username,  object_name, waits,
 16           sql_text
 17  FROM ash_query
 18  WHERE wait_rank < 11
 19* ORDER BY wait_rank
SQL> /

EVENT                     MODULE       USERNAME OBJECT_NAME       WAITS
------------------------- ------------ -------- ------------ ----------
SQL_TEXT
------------------------------------------------------------
library cache: mutex X    SQL*Plus     OPSG                          13

latch: shared pool        SQL*Plus     OPSG                           8

latch: shared pool        SQL*Plus     OPSG     LT_SALES_PK           3
 begin     latch_test(10000,10000,1000000,10000);  end;

library cache: mutex X    SQL*Plus     OPSG     LT_SALES_PK           2

library cache: mutex X    SQL*Plus     OPSG     LT_SALES              1
 begin     latch_test(10000,1,10000,10000);  end;

latch: shared pool        SQL*Plus     OPSG                           1
SELECT  quantity_sold , amount_sold FROM lt_sales t539564 WH
ERE id BETWEEN 124410 AND 360759

latch: shared pool        SQL*Plus     OPSG                           1
SELECT  quantity_sold , amount_sold FROM lt_sales t539571 WH
ERE id BETWEEN 512313 AND 825315

library cache: mutex X    SQL*Plus     OPSG                           1
SELECT  quantity_sold , amount_sold FROM lt_sales t539563 WH
ERE id BETWEEN 698302 AND 392634

latch: shared pool        SQL*Plus     OPSG     LT_SALES_PK           1
SELECT  quantity_sold , amount_sold FROM lt_sales t539555 WH
ERE id BETWEEN 387009 AND 268338
```

If you don't have a Diagnostic Pack license, then you can indirectly identify the SQLs by focusing on those SQLs with the highest concurrency wait times. The concurrency wait class includes most commonly encountered latch and mutex waits, although it also includes some internal locks and buffer waits. However, if you're encountering high rates of latch contention, it's a fair bet that the SQLs with the highest concurrency waits are the ones you want to look at.

Listing 16.4 pulls out the SQLs with the highest concurrency waits.

Listing 16.4 Identifying High Contention SQL

```
SQL> WITH sql_conc_waits AS
  2      (SELECT sql_id, SUBSTR(sql_text, 1, 80) sql_text,
  3              concurrency_wait_time/1000 con_time_ms,
  4              elapsed_time,
  5              ROUND(concurrency_wait_Time * 100 /
  6                  elapsed_time, 2) con_time_pct,
  7              ROUND(concurrency_wait_Time* 100 /
                SUM(concurrency_wait_Time) OVER (), 2) pct_of_con_time,
            RANK() OVER (ORDER BY concurrency_wait_Time DESC) ranking
       FROM v$sql
      WHERE elapsed_time > 0)
SELECT sql_text, con_time_ms, con_time_pct,
       pct_of_con_time
FROM sql_conc_waits
WHERE ranking <= 10
ORDER BY ranking  ;

                                                       SQL Conc    % Tot
SQL Text                                 Conc Time(ms)    Time% ConcTime
---------------------------------------- ------------- -------- --------
DECLARE job BINARY_INTEGER := :job; next           899    18.41    44.21
_date DATE := :mydate;  broken BOOLEAN :

select max(data) from log_data where id<           472      .01    23.18
:id

begin   query_loops ( run_seconds=>120 ,           464      .01    22.80
 hi_val =>1000 ,                    use_

update sys.aud$ set action#=:2, returnco           143    75.46     7.02
de=:3, logoff$time=cast(SYS_EXTRACT_UTC
(
```

As expected the SQL that generated the latch waits is found (the second and third entries are from a job that generated the latch waits). However, other SQLs—associated with waits for certain internal Oracle locks—are also shown. You'll need to exercise judgment to determine which SQLs are most likely associated with your latch waits.

Latch and Mutex Scenarios

Along with these generic methods of associating latch waits with SQLs and segments, there are diagnostic techniques specific to certain types of latch contention. We look at these as we discuss specific latch/mutex wait scenarios in the sections that follow.

Library Cache Mutex Waits

The library cache is the part of the shared pool in which cached definitions of SQL, PL/SQL, and Java classes are held. Modifications to the library cache are protected by library cache mutexes. Prior to Oracle Database 10*g* Release 2, they were protected by library cache latches.

Oracle maintains a cache of SQL statements in the shared pool. If a matching SQL is found in the shared pool, then most of the overhead of parsing a statement can be avoided. Such a parse is called a *soft parse*. If no matching SQL is found, a hard parse must be performed.

The most common reason to acquire a library cache mutex in exclusive mode is to add a new entry to the cache. This happens, for instance, when we parse a new SQL statement. Oracle looks for a matching entry in the cache, and if one is not found (a “miss”), it acquires the relevant mutex and inserts the new entry. Failing to obtain the mutex will result in a `library cache: mutex X` wait.

The most common cause of library cache mutex contention is excessive hard parsing caused by a failure to use bind variables in application code. For example, in the following Java snippet, a SQL statement object is created, executed, and discarded:

```
Statement s=oracleConnection.createStatement();
s.execute("UPDATE sh.customers SET cust_valid = 'Y'"+
         " WHERE cust_id = 1");
s.close();
```

If your application does nothing but execute a single SQL, then this code is probably okay. But it’s common for a SQL statement to be executed more than once, selecting or modifying different rows with each execution. This next Java snippet issues an `UPDATE` statement once for every customer ID held in the `custIdList` array:

```
1  for (int custId : custIdList) {
2       Statement stmt = oracleConnection.createStatement();
3       stmt.execute("UPDATE sh.customers SET cust_valid = 'Y'"
4                   + " WHERE cust_id = " + custId);
5        stmt.close();
6  }
```

The loop starting on line 1 iterates through an array of `CUST_ID` values. We create a statement object (line 2) and then construct and execute an `UPDATE` statement once for each customer in the list. We concatenate the `custId` from the list into the SQL string on line 3.

This code will work, of course, but each `UPDATE` statement will need to be parsed as well as executed. This parse overhead can be significant. Furthermore, because each SQL is unique (it includes the hardcoded `custId`), we’re unlikely to find a matching SQL in the shared pool. Therefore, a *hard parse*—one in which no matching SQL is found in the shared pool—will be required.

The next code snippet shows the bind variable technique in Java. The SQL statement is created as a `PreparedStatement` and includes a bind variable—identified as `:custId`—which acts as a placeholder for the parameters to the SQL. The variable is assigned a value on line 5 prior to each execution on line 6:

```
1  PreparedStatement stmt = oracleConnection.prepareStatement(
2        "UPDATE sh.customers SET cust_valid = 'Y'"
3      + " WHERE cust_id = :custId");
4  for (int custId : custIdList) {
5       stmt.setInt(1, custId);
6       stmt.execute();
7  }
```

Using the bind variable technique radically reduces the parse overhead of SQL execution, and in particular, it reduces the amount of library cache mutex contention.

To identify the SQLs that are causing the most hard parses, we need to find those SQLs that are identical other than for the values of literals. These SQLs will show up in `V$SQL` as SQLs with the same value for `FORCE_MATCHING_SIGNATURE`, as shown in Listing 16.5.

Listing 16.5 Finding SQLs That Are Not Using Bind Variables

```
SQL> WITH force_matches AS
            (SELECT force_matching_signature,
                    COUNT( * )  matches,
                    MAX(sql_id || child_number) max_sql_child,
                    DENSE_RANK() OVER (ORDER BY COUNT( * ) DESC)
                       ranking
             FROM v$sql
             WHERE force_matching_signature <> 0
               AND parsing_schema_name <> 'SYS'
             GROUP BY force_matching_signature
             HAVING COUNT( * ) > 5)
     SELECT sql_id,     matches, parsing_schema_name schema, sql_text
       FROM       v$sql JOIN force_matches
         ON (sql_id || child_number = max_sql_child)
     WHERE ranking <= 10
     ORDER BY matches DESC;


SQL_ID           MATCHES SCHEMA
------------- ---------- --------------------
SQL_TEXT
------------------------------------------------------------------
gzxu5hs6sk4s9      13911 OPSG
select max(data) from log_data where id=717.91
```

The query reveals that there were 13,911 instances of a SQL statement that was identical except for the value of a variable—a variable that could have been represented by a bind variable.

Ideally, applications should make use of bind variables whenever possible by changing the application code, as outlined earlier in this section. However, it's not always easy or possible to rewrite an application to use bind variables. Therefore, Oracle provides a mechanism for imposing bind variables transparently; when the parameter `CURSOR_SHARING` is set to `FORCE` or `SIMILAR`, then Oracle can replace a statement such as this:

```
SELECT MAX(data) FROM log_data WHERE id=717.91
```

with a statement like this:

```
SELECT MAX(data) FROM log_data WHERE id=:"SYS_B_0"
```

Oracle will then substitute the appropriate values into the system-generated bind variables (value 717.91 would be assigned in the previous example), and the library cache miss will be avoided. As we saw earlier, this behavior reduces parse overhead—since Oracle can retrieve the already parsed version from the shared pool—and it also reduces mutex contention, since Oracle doesn't have to acquire the mutex in exclusive mode if the matching SQL is found.

Library Cache Pin

The `library cache pin` wait is not strictly a latch or mutex wait, but it often shows up in similar circumstances. A library cache pin is required whenever an object in the library cache is to be executed, parsed, or reparsed. The library cache pin is acquired in exclusive mode if, for instance, the execution plan for a SQL statement needs to be changed or a PL/SQL package is modified or recompiled. The library cache pin is acquired in exclusive mode by the sessions executing the object.

The session wanting to modify the object will attempt to acquire the library cache pin in exclusive mode; sessions executing the object will be holding a shared library cache pin.

Excessive waits on the library cache pin may suggest that PL/SQL packages are being recompiled during periods of heavy concurrent execution. If possible, schedule recompilation during maintenance windows.

Shared Pool Latch

The primary purpose of shared pool latches is to control access to the shared pool memory map. Sessions that are looking for free space in the shared pool for a new SQL statement or PL/SQL package will need to acquire shared pool latches, and many Oracle internal operations (resizing the shared pool for instance) will acquire these latches as well.

Excessive hard parsing—the primary cause of library cache mutex contention—generally results in shared pool latch contention as well, because the constant allocation of "one-off" SQL statements will fragment the shared pool and require continual deallocation of old statements. This will show up as waits for the `latch: shared pool` event.

Shared pool fragmentation has other deleterious side effects, including ORA-4031 errors ("unable to allocate x bytes of shared memory") and excessive shared pool memory consumption. Over the years, a variety of techniques have been employed to combat this fragmentation:

- Some sites flush the shared pool periodically using the `ALTER SYSTEM FLUSH SHARED_POOL` command.

- Setting a minimum size for the shared pool when using automatic SGA memory management is almost always a good idea but particularly if shared pool latch contention is present. Using automatic SGA memory management can exacerbate fragmentation issues, since the memory management algorithms are not always able to predict or measure the degree of fragmentation that will result from continual resizing.
- Pinning large but infrequently executed PL/SQL packages in the shared pool—using `DBMS_SHARED_POOL`—might help reduce fragmentation by preventing large objects moving in and out of memory.
- The `SHARED_POOL_RESERVED_SIZE` parameter controls the amount of shared pool reserved for large memory allocations. In some instances, increasing the size of this parameter may reduce pressure on the shared pool latch by reducing the amount of time it takes to find large, contiguous chunks of memory

Cache Buffers Chains Latch

A *cache buffer chain* (CBC) is a doubly linked list of buffer headers pointing to buffers in the buffer cache that hash to a common value. There are a number of CBC latches, each latch protecting multiple cache buffer chains.

When a session needs to access a buffer in the buffer cache, it must acquire a CBC latch on the “chain” that contains that buffer. Contention for this latch results in waits for the `latch: cache buffers chains` event.

The amount of time it takes to access a block in memory is very small, and there are a large number of CBC latches. Nevertheless, cache buffers chains latch contention can become significant on systems with very high logical read rates, especially if these logical reads concentrate on a small number of blocks. Another possible cause is the creation of long buffer hash chains caused by multiple consistent read copies of an individual buffer.

Ironically, cache buffers chains latch contention often occurs on systems that are almost perfectly optimized in every other respect: in order to get the very high logical read rates necessary to induce cache buffers chains contention, the system typically needs to minimize all other forms of contention and waits, such as IO, parsing, and locking.

High logical read rates and the resulting CBC latch contention can, however, be the result of poorly tuned SQL as well. For example, a nested loops join that uses an unselective index may scan the same set of blocks on the inner table many times over. These blocks will then become “hot” and may be the subject of latch contention. Tuning the SQL by creating a more selective index will reduce the redundant logical reads and reduce the latch contention as well as improve the performance of the SQL concerned.

The mapping of cache buffers to cache buffers chains latches is based on an Oracle hashing algorithm, and the number of blocks per latch can vary significantly. If you want to examine the configuration of your cache buffers chains latches, the query in Listing 16.6—which you must run as SYS—will reveal the latch to buffer ratios.

Listing 16.6 Revealing the CBC Latch to Buffer Ratio

```
SQL> SELECT COUNT(DISTINCT l.addr) cbc_latches,
            SUM(COUNT( * )) buffers,
            MIN(COUNT( * )) min_buffer_per_latch,
            MAX(COUNT( * )) max_buffer_per_latch,
            ROUND(AVG(COUNT( * ))) avg_buffer_per_latch
     FROM       v$latch_children l
          JOIN
             x$bh b
          ON (l.addr = b.hladdr)
     WHERE name = 'cache buffers chains'
     GROUP BY l.addr;

 CBC Latch Buffer Cache Min Buffer Max Buffer Avg Buffer
     Count      Buffers  Per Latch  Per Latch  Per Latch
---------- ------------ ---------- ---------- ----------
      8192        89386          3         46         11
```

On this database, an average of 11 blocks was associated with each latch, but some latches protected as few as 3 or as many as 46 blocks.

The chance that contention for a cache buffers chains latch is a result of two hot blocks being mapped to the same latch is pretty small, and while you can attempt to change the number of latches using undocumented Oracle parameters (such as `_db_block_hash_buckets`), the chances that you'll relieve latch contention by doing so are not good.

Each latch exposes its individual statistics into the view `V$LATCH_CHILDREN`. You can link these latches to the buffers they protect by examining the view `X$BH` (which, unfortunately, you can only do as the SYS user). The query in Listing 16.7 joins the two tables to identify the segments that are most heavily associated with cache buffers chains latch sleeps.

Listing 16.7 Identifying Segments with High CBC Latch Sleeps

```
SQL> WITH cbc_latches AS
         (SELECT addr, name, sleeps,
                 rank() over(order by sleeps desc) ranking
            FROM v$latch_children
           WHERE name = 'cache buffers chains')
     SELECT owner, object_name,object_type,
            COUNT(distinct l.addr) latches,
            SUM(tch) touches
```

```
  FROM cbc_latches l JOIN x$bh b
       ON (l.addr = b.hladdr)
  JOIN dba_objects o
       ON (b.obj = o.object_id)
 WHERE l.ranking <=100
 GROUP BY owner, object_name,object_type
ORDER BY sum(tch) DESC;

OWNER        OBJECT_NAME          OBJECT_TYP    LATCHES      TOUCHES
------------ -------------------- ---------- ---------- ------------
OPSG         LOG_DATA             TABLE             103        1,149
```

This query shows that the top 100 cache buffers chains latches are all associated with the `LOG_DATA` table and that it is probably the high rates of logical I/O against this table that are the root cause of the cache buffers chains latch contention we are experiencing.

Finding the segment involved in cache buffers chains contention is a good first step, but where do we go from here? There are a couple of possibilities:

- If the cache buffers chains contention is associated with an index, then you could consider reimplementing the table as a hash cluster and use a hash key lookup rather than a B-tree index lookup. B-tree indexes often become associated with cache buffers chains contention, because index root and branch blocks tend to be accessed more frequently than index leaf blocks or table blocks. If we use a hash cluster lookup instead, this potential for cache buffers chains latch contention is eliminated. If the B-tree index is part of a nested loops join, then a hash join might similarly relieve pressure on the index blocks.
- At the risk of belaboring the point, is there any way to reduce the logical I/O rate? Review and tune SQL that accesses the table: perhaps a judicious index or two could reduce the logical I/O demand. Perhaps Oracle client-side caching (`CLIENT_RESULT_CACHE_SIZE` parameter) or the Oracle server-side result set cache could be used to reduce the logical I/O rate.
- If there are multiple hot rows within the same hot block, explore options for splitting these rows across multiple blocks. Partitioning the table and its indexes can be an attractive option, especially since it requires no changes to application code.

Other Latch Scenarios

Cache buffers chains latches and library cache mutexes are the most commonly encountered forms of latch/mutex contention. However, other forms of latch contention arise from time to time. Here are some of the other latches that you might encounter:

- **Cache buffers lru chain latch:** This latch controls access to the LRU (least recently used) list in the buffer cache. Buffers move up and down this list as they are accessed and, once they reach the cold end of the list, are eventually flushed out of the pool. Contention for this latch is generally associated with cache buffers chains latch contention and will generally respond to a similar resolution. However, while the cache buffers chains latch is most sensitive to *hot* blocks, the cache buffers lru chains latch is more heavily utilized when *new* blocks are introduced into the buffer cache.
- **Simulator lru latch:** This latch controls access to the "virtual" LRU list that Oracle uses to work out the effect of increasing or decreasing the size of the buffer cache. This information is used to populate the `DB_CACHE_ADVICE` tables and to perform automatic memory management. Contention for this latch can occur under similar circumstances as for the cache buffers chains and cache buffers lru chains latches and may mask contention for those latches. Setting `DB_CACHE_ADVICE` to `OFF` will usually eliminate this contention but may merely shift the contention to the cache buffers chains latch. Note also that contention on this latch was associated with some Oracle bugs in early versions of Oracle 11.
- **Redo allocation latch:** This latch serializes entries to the redo log buffers and private "strands," both of which buffer I/O to the redo logs. This latch—and the related redo copy latch—were often implicated in latch contention issues in earlier versions of Oracle. However, Oracle made significant changes to redo handling in Oracle Database 9*i* and 10*g*, parallelizing redo generation, creating multiple independent buffers, and introducing private buffer strands. As a result, redo-related latch contention issues are rarely reported today. You may see some contention for the redo allocation latch when there are very high levels of concurrent DML activity. However, it's unlikely to dominate overall performance, since such high levels of DML generally create substantial I/O-related waits.
- **Session allocation and process allocation latches:** These latches are often involved during the creation of a new session and, in the case of process allocation, associated server process. Contention on these latches will often be seen if there is a very high rate of logon/logoff to the database. Oracle is not really optimized for sessions that connect, issue a single SQL, and then disconnect; performance will usually be better when sessions stay connected and issue multiple SQLs. Using application server connection pools is advisable if you see this sort of contention, and you might see some relief if you configure the database for multithreaded server connections.

- **kks stats latch:** This latch seems to be associated with mutex operations—one might speculate that it is involved in maintaining mutex sleep statistics. Some contention on this latch seems to be associated with other mutex contention scenarios. If you see this latch in conjunction with mutex waits, you should probably try resolving the mutex issue first with the hope of curing contention for this latch as well.
- **In-memory undo latch:** This latch is associated with Oracle's relatively new *in-memory undo* (IMU) structures in which information formerly maintained in rollback (undo) segments is held in memory. Some contention for the in-memory undo latch might be the cost you have to pay for the reduction in redo generation and undo segment I/O that the new algorithm provides. However, some users have suggested turning in-memory undo off by adjusting the undocumented parameter `_in_memory_undo` or increasing the value of the `PROCESSES` parameter, which controls the default number of IMU latches.
- **Result cache (RC) latch:** This latch protects the creation and deletion of result sets in the result set cache. Contention for the latch occurs if multiple sessions attempt to simultaneously create cached result sets. Result sets in the result set cache should generally be restricted to a relatively small number of infrequently executing SQLs.

Intractable Latch Contention

We often see latch contention—especially cache buffers chains latch contention—in the most highly tuned, high-powered databases.

This makes sense if you think about it. If we create a database configuration in which all other constraints are removed on database performance (locking, I/O, memory, CPU), then database sessions will essentially be competing for access to shared memory alone. In that scenario, latch contention will inevitably become the limiting factor.

So it may be that some degree of latch contention—especially on the cache buffers chains latch—has to be accepted in very high-throughput systems on premium hardware. When we hit this sort of intractable latch contention, we may have no other option than to attempt to manipulate the parameters that control the Oracle internal algorithms that control latching.

Fine Tuning Latch Algorithms

As we've noted, latch requests often repeatedly attempt to acquire a latch (a spin get) before surrendering the CPU and going into a latch sleep. If all else fails, fine tuning the number of spins may improve throughput for an application.

Figure 16.1 shows the results of experiments in which the variable _spin_count, which controls the default number of spins, was varied for an Oracle 11*g* database suffering from library cache latch contention (http://bit.ly/1fyARb5). In the figure, you can see that as the parameter _spin_count increases, application throughput may also increase. This occurs at the expense of overall CPU consumption, but provided there is free CPU, increasing the _spin_count has a beneficial effect.

Latch expert Andrey Nikolaev has shown similar results manipulating _mutex_spin_count (http://arxiv.org/pdf/1212.6640v1.pdf) and _spin_count (http://arxiv.org/pdf/1111.0594v1.pdf).

It remains true that if you are unable to resolve latch contention in any other fashion, you might try manipulating the amount of spinning that Oracle performs on a latch. Unfortunately, this option has become less and less attractive as Oracle changes the underlying spin algorithms.

In the past (prior to Oracle Database 11*g*), you could manipulate spins using the single dynamic parameter _spin_count. Today that approach is suitable only for shared mode latch gets. Tuning exclusive latches requires manipulation of the _latch_class_0 parameter and will require a database restart. For mutexes, three parameters control spin behavior: _mutex_spin_count sets the number of spins before yielding or sleeping, while _mutex_wait_scheme and _mutex_wait_time

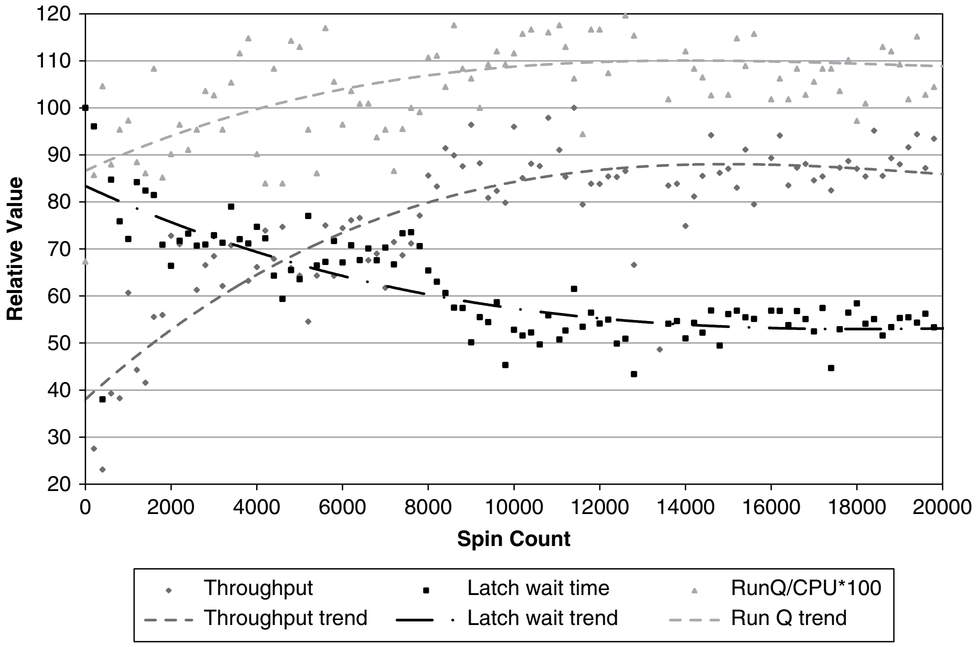


Figure 16.1 Relationship between _spin_count, latch waits, and throughput

control the waiting behavior. See Oracle Bug 10411618 for a description of how these parameters work.

In short, if all else fails, you may try to manipulate latch spinning to alleviate latch contention. However, doing so involves manipulating undocumented parameters and should be done as a last resort with extreme care.

Summary

In this chapter, we looked at how latches and mutexes work and why they are a necessary part of the Oracle architecture. We learned how to diagnose the root causes of latch and mutex contention and explored remedies for common latch contention scenarios.

Latches and mutexes protect areas of Oracle's shared memory, preventing corruption or inconsistencies that might arise if multiple sessions were to change the same area of shared memory at the same time.

Latch internal algorithms are complex and change frequently. However, most latch contention indicates a need to reduce demand by the application on shared memory. The following are the two most common causes:

- Hard parsing, in which new SQL statements are constructed for each change in a query parameter. The use of bind variables or the `CURSOR_SHARING` parameter may be indicated.
- Very "hot" blocks in the buffer cache, suggesting a need to partition a table or tune SQL.

This page intentionally left blank

Index

A

Access structures, 296
ACS (adaptive cursor sharing)
  bind-awareness monitoring, 61–73
  demonstration, 58–61
  and SPM. *See* SPM (SQL plan management), adaptive cursor sharing.
  working algorithm, 52–57
ACS (adaptive cursor sharing), bind sensitiveness with
  equality predicate histograms, 55–56
  partition keys, 56–57
  range predicate, 52–55
ACS (adaptive cursor sharing), bind-aware cursors
  creating, examples, 66–73
  description, 73–76
  performance issue, example, 76–81
Active Data Guard, 279
Active Session History (ASH). *See* ASH (Active Session History).
Actuator arms, 388
Adams, Steve, 370
`ADD_FILE` command, 317, 319
ADDM (Automatic Database Diagnostic Monitor), finding buffer busy wait event information, 41
Administrative tasks, utilities and commands, 283
ADO (Automatic Data Optimization), 160–162, 326, 410
ADR (Automatic Diagnostic Repository), 276–277
Advanced Index Compression, 162
Advanced Row Compression, 160
Alert logs, 277
`ALL_ROWS` parameter, 117–122
`ALTER DATABASE FLASHBACK OFF` command, 179
`ALTER DATABASE FLASHBACK ON` command, 179
`ALTER INDEX <`*`index name`*`> VISIBLE` command, 441
`ALTER SYSTEM FLUSH SHARED_POOL` command, 378
`Alter table` operation, 142–144, 147–152
`ALTER TABLESPACE <tablespace_name> FLASHBACK OFF` command, 179
AMM (Automatic Memory Management), 281
Antivirus software, 281
ASH (Active Session History)
  buffer busy wait event information, 43–44
  finding latch contention, 373–375
  waiting sessions, 45–46
ASM (Automatic Storage Management), disk groups on Exadata, 416–418
`ATTACH` parameter, 319
Automatic Data Optimization (ADO), 160–162, 326, 410
Automatic Database Diagnostic Monitor (ADDM), 41, 277
Automatic Diagnostic Repository (ADR), 276–277
Automatic Maintenance Tasks (AUTOTASK) framework, 290–291
Automatic Memory Management (AMM), 281
Automatic Storage Management (ASM), disk groups on Exadata, 416–418
AUTOTASK (Automatic Maintenance Tasks) framework, 290–291
Autotuning retention values, disabling, 21

AWR (Automatic Workload Repository). *See also* Performance tuning.
basic reports, 198
buffer pool advisory, 239–240
buffer pool statistics, 237–240
buffer waits statistics, 247–248
description, 197–198
disk spills, 264–266
dynamic memory components, 260–262
enqueue statistics, 248–250
excessive disk spills, 246
finding buffer busy wait event information, 41–43
initialization parameter changes, verifying, 266
instance recovery statistics, 239
Java pool advisory, 245–246, 247
library cache activity, 257–260
OOS (out-of-space) errors, 251
overloaded buffer cache, 239
resource limits, 266
segment access, 255–257
SGA target advisory, 245–246
shared pool statistics, 244–245
STO (snapshot too old), 251
stream pool size, 264–266
streams components, 264–266
streams pool advisory, 245–246
tablespace I/O statistics, 235–237
time model statistics, 211–212
timing, 266
undo segment statistics, 250–251
what to look for, 199
AWR (Automatic Workload Repository), header section
buffer hit percentage, 202
buffer nowait percentage, 202
instance CPU, 207
instance efficiencies, 202–203
latch hit percentage, 203
library hit percentage, 202
load average, 206–207
load profile, 201–202
log file stress, 206
memory sort percentage, 202–203
memory statistics, 207–208
non-parse CPU percentage, 203
overview, 199–201
redo nowait percentage, 202
shared pool memory statistics, 203
soft parse percentage, 203
wait events, 203–206
AWR (Automatic Workload Repository), instance activity statistics
absolute values, 232–233
consistent gets, 224
dirty blocks, 224
enqueue, 224
execution count, 225
free buffer, 225
GC (global cache), 225
index fetch by key, 226
index scan, 226
index scans kdiixs1, 226
leaf nodes, 226
open cursors, 226
overview, 221–224
parses, 226
physical reads and writes, 226
recursive calls, 229
redo related, 229
session cursor, 229–230
sorts, 230
summed dirty queue length, 230
table fetch, 230–231
thread activity, 233
transaction rollback, 231
undo change vector, 231–232
user I/O wait time, 232
work area, 232
AWR (Automatic Workload Repository), latch statistics
latch activity, 253
miss sources, 254–255
no latch available, 253–254
overview, 251–253
parent and child latches, 255
Pct Get Misses, 253
Pct NoWait Misses, 253
sleep breakdown, 253–254
sleep summary, 255
spin count, 254
AWR (Automatic Workload Repository), OS statistics
background wait events, 214–215
foreground wait events, 213–214
overview, 212–213
service related statistics, 216–217
wait event histograms, 215–216
AWR (Automatic Workload Repository), PGA statistics
aggregate summary, 241–242
aggregate target histogram, 242–243
aggregate target statistics, 242
cache hit percentages, 241–242
memory advisor, 243–244
OOBs (out-of-band-sorts), 243
overview, 240–241
rolled up usage data, 241–242
AWR (Automatic Workload Repository), process memory
overview, 262–263
process memory summary, 264
SGA breakdown difference, 264
SGA memory summary, 264
AWR (Automatic Workload Repository), RAC-specific pages
cluster interconnects, 210
global cache and enqueue services, 209–210, 268–273

global cache load statistics, 209
global cache transfer statistics, 271–272
global CR served statistics, 271
global current served statistics, 271
global enqueue statistics, 271
hot blocks, 271
RAC statistics (CPU), 208
AWR (Automatic Workload Repository), SQL sections
buffer gets, total, 218–219
cluster wait time, 220–221
CPU time, total, 218
disk reads, total, 219
elapsed time, total, 217–218
executions, total, 219
overview, 217–218
parse calls, 219–220
recursive calls, 218
recursive CPU usage, 218
unsafe bind variables, 220
version count, 220
awrddrpi.sql script, 198
awrddrpt.sql script, 198
awrgdrpt.sql script, 198
awrgrpt.sql script, 198
awrrpti.sql script, 198
awrrpt.sql script, 198
awrsqrpti.sql script, 198
awrsqrpt.sql script, 198

B

Background wait event statistics, 214–215
Backup and recovery. *See also* RMAN (Recovery Manager).
backup optimization and tuning, 187–188
BCT (block change tracking), 170, 178
best practices, 170–172
Data Guard configuration, 172
DRA (Data Recovery Advisor), 193–194
Exadata solutions, 171
excluding tablespaces, 179
Flashback Database features enabling/disabling, 179
FRA (fast recovery area), 179
guaranteed restore points, 179
most recently detected failures, 193
recovery factors, 174
recovery strategies, 192–193
rewinding in Oracle Flashback technology, 178–179
RPO (recovery point objective), 174
RTO (recovery time objective), 174
TSPITR (tablespace point-in-time recovery), 179
for VLABs and XLABs, 170–172
Backup and recovery, backup strategies
binary compression, 176–177
compressed backups, 176–177
cumulative incremental backups, 177
decompression, 177
differential incremental backups, 177
disk-based backup, 179, 189
full backups, 176
incremental backups, 176–178
IUIC (incrementally updatable image copies), 180–186
key elements, 175–176
null block compression, 176
RFF (recover forward forever), 180–186
unused block compression, 176
BACKUP DURATION clause, 188
BACKUP OPTIMIZATION setting, 188, 192
_backup_disk_bufcnt parameter, 189
_backup_disk_bufsz parameter, 189
_backup_file_bufcnt parameter, 189
_backup_file_bufsz parameter, 189
Balanced tree (B-tree) indexes, 422–425, 432
BASICFILE LOBs. *See also* LOB (large object) data type.
issues, 8
vs. SECUREFILE, 8–11
BASICFILE LOBs, migrating to SECUREFILE LOBs
example, 12–14
poor INSERT performance, 17
BCT (block change tracking)
definition, 170
enabling, 178
BFILE data type, managing free space, 16
Bigfile tablespaces, 156–157
Binary compression, 176–177
Binary large object (BLOB), managing free space, 16
Bind variable technique, 376–378
Bind variables
identifying, 451
SPM (SQL plan management), 86
Bind-awareness monitoring, 61–73
Bitmap indexes, 425–426
Bitmap join indexes, 431
BLOB (binary large object), managing free space, 16
Block-checking parameters, configuring, 279
Blocks, SSD, 392
blocksize designation parameter, 237–238
Books and publications
"ASMM vs. AMM and LINUX HugePages Support," 281
"HugePages and Oracle 11g Automatic Memory Management (AMM) on Linux," 281
Oracle Database 12c Reference Guide, 36
Oracle Database SecureFiles and Large Objects Developer's Guide, 1
Oracle Exadata Expert's Handbook, 414
Oracle Tuning Guide and Concepts Manual, 199
Oracle8i Internal Services, 370–371
Bottlenecks. *See* Performance bottlenecks.
Branch pages, 422
B-tree (balanced tree) indexes, 422–425, 432
BUCKET_ID, COUNT relationship, 62–66

Buffer busy wait events
buffer busy, 36
finding waiting sessions, 45–46
fixes for, 49–50
gc buffer busy, 36
gc buffer busy acquire, 36
gc buffer busy release, 36
isolating issues, 45–49
key tools. *See* ADDM (Automatic Database Diagnostic Monitor); ASH (Active Session History); AWR (Automatic Workload Repository); ORAchk utility.
overview, 35–36
performance bottlenecks, isolating, 47–49
read by other session, 36
types of, 36
Buffer busy wait events, finding event information with
ADDM, 41
ASH, 43–44
AWR, 41–43
`buffer busy waits`, 239
Buffer gets, analyzing, 218–219
Buffer hit percentage, analyzing, 202
Buffer nowait percentage, analyzing, 202
Buffer pool
advisory, 239–240
statistics, 237–240
waits statistics, 247–248

C
Cache buffers lru chain latch, 382
`CACHE` directive, 9
Cache hit percentages, 241–242
Caching, 9
CBC (cache buffer chain) latches, 379–381
`C_DDL` column
in a column group extension, 140–142
default value changes, 142–144
and indexes, 145–147
in a virtual column, 139–140
Cells, 391–392
Child latches, 255
CHM (Cluster Health Monitor), 278
`CHUNK` parameter, 9
Chunk size, specifying, 9
CLOB (character large object), managing free space, 16
Cloning databases, 330
Cluster interconnects analyzing, 210
Cluster wait time, analyzing, 220–221
Clustering factors, indexes, 435–436
Clusterware componentry status, checking, 283–284
Columns, with default values, adding to tables. *See* DDL (data definition language) optimization.
`COMPARE` method, 451
Composite indexes, 430
Compressed backups, 176–177
Compressed indexes, 431
Compression
data, 159–160
managing LOB data types, 8
SSDs (solid-state drives), 410
table, 160
Compression, VLDBs and XLDBs
Advanced Index Compression, 162
Advanced Row Compression, 160
data compression, 159–160
HCC (Hybrid Columnar Compressions), 160
Oracle Advanced Compression, 160
table compression, 160
Concatenated indexes, 430
`CONCURRENT` global preference, 169
Configuration information, displaying, 287–288
Consistent gets, instance activity statistics, 224
`CONTENT` parameter, 306
Contention
hanging databases, 22–24
interinstance, 441
latches and mutexes. *See* Latches and mutexes, contention.
sequences and indexes, 442
`CONTINUE_CLIENT` command, 319
`CONTROL_FILE_RECORD_KEEP_TIME` parameter, 191
Copying. *See also* Data Pump; Migration.
cloning databases, 330
duplicate schema objects, 305
entire databases, 305
objects between databases, 304–305
from Oracle Database 9*i* or older, 305
schemas between databases, 304–305
table metadata only, 306
tables between databases, 305
tablespaces, 306
CPT (cross-platform transport), 331, 344–345
CPU management, 281
CPU time, analyzing, 218
`CPU_COUNT` parameter, 281
Cross-platform transportable tablespaces (XTTS), 331, 340–343
`crsctl check` commands, 283
`crsctl get` commands, 284–285
`crsctl query` commands, 284
`crsctl status` commands, 284–286
`CRSCTL` (Oracle Clusterware Control) utility, 283
Cumulative incremental backups, 177
`CURSOR_SHARING` parameter, 377

D
Data, excluding from export, 321
Data block corruption, protecting against, 279
Data block size, optimal, 155–156
Data compression, 159–160. *See also* Compression.
Data definition language (DDL) optimization. *See* DDL (data definition language) optimization.
Data dictionary object block corruption, 30–31

Data Guard, physical standby database, 333
Data Guard backup and recovery configuration, 172
Data partitioning, 158–159
Data Pump. *See also* Copying; Exporting; Importing; Migration.
  changing object properties, 313–317
  copying objects, 304–305
  database directory location, specifying, 306
  database links, saving and restoring, 307
  database links and synonyms, exporting, 307–308
  default storage parameters, 314
  excluding BLOB data, 321
  exiting, 319
  Export/Import utilities, 345
  FTE (full transportable export/import), 346, 347–350
  importing partitioned tables as nonpartitioned, 313
  importing table partitions as individual tables, 313
  improving performance, 320–321
  invoking, 303
  job name, getting, 319
  job status, displaying, 319
  log file, specifying, 306
  masking database, 314
  modes, 305–306
  monitoring and altering resources, 319
  overview, 303–304
  public and private objects, 306–309
  renaming tables, 314
  return to logging mode, 319
  scrambling sensitive data, 314
  with SQL*Plus (PL/SQL API), 317–319
  upgrading databases, 321–322
Data Pump, dump files
  adding, 319
  resizing, 319
  scrambling sensitive data, 314
  specifying, 306
  verifying content of, 308–309
Data Pump, tablespaces
  consolidating, 315–317
  names, specifying, 306, 314
  resizing, 314
Data Recovery Advisor (DRA), 193–194
Data warehouse templates, 154–155
Database Configuration Assistant (DBCA), 155
Database directory location, specifying, 306
Database Flash Cache (DBFC). *See* DBFC (Database Flash Cache).
Database links
  exporting, 307
  saving and restoring, 307
  and synonyms, exporting, 307–308
Database writer (DBWR) process, 398
Databases
  cloning, 330
  hanging. *See* Hung databases.
  masking sensitive data, 314
  upgrading, 321–322
  very large. *See* VLDBs (very large databases); XLDBs (extremely large databases).
Databases, transferring data
  to another database. *See* Cloning databases; Data Pump.
  from a file. *See* Importing.
  to a file. *See* Exporting.
Datafiles
  I/O stress, 237
  limiting the number of, 156–157
`DATA_PUMP_DIR` parameter, 306
`DBA_DATAPUMP_JOBS` view, 319
`DBA_DATAPUMP_SESSIONS` view, 319
`DBA_DB_LINKS` view, 308
`DBA_SYNONYMS` view, 308
`DB_BLOCK_CHECKING` parameter, 279
`DB_BLOCK_CHECKSUM` parameter, 279
DBCA (Database Configuration Assistant), 155
`DB_CACHE_ADVICE` parameter, 382
`db_cache_size` parameter, 261
DBFC (Database Flash Cache)
  caching segments, 399–400
  configuring, 398–399
  creating, 399
  DBWR process, 398
  deferred writing of changed blocks, 396
  `FLASH_CACHE` clause, 399–400
  free buffer waits, 396–398
  lazy writes, 396
  monitoring, 398–399
  overview, 396
  performance statistics, 400–402
  writing to the flash cache, 398
`DB_FLASHBACK_RETENTION_TARGET` parameter, 179
`DB_FLASH_CACHE_FILE` parameter, 398–399
`DB_FLASH_CACHE_SIZE` parameter, 398–399
`DB_LOST_WRITE_PROTECT` parameter, 279
`DBMS DEFINITION` package, 12–14
`DBMS_FILE_TRANSFER` utility, 339, 344, 349
`DBMS_SHARED_POOL` parameter, 379
`DBMS_SQLDIAG` package, 300
`DBMS_STATS.GATHER_TABLE_PREF` package, 169
`DBMS_UNDO_ADV` package, 21
`DBMS_UNDO_ADVISOR` procedure, 21
`DBMS_WORKLOAD_REPOSITORY` package, 197–198
`dbms_xplan` package, 88
`DB_RECOVERY_FILE_DEST` parameter, 179
`DB_RECOVERY_FILE_DEST_SIZE` parameter, 179
DBWR (database writer) process, 398
`DBWR_IO_SLAVE` parameter, 188
DDL (data definition language) optimization
  `alter table` operation, 142–144, 147–152
  `C_DDL` column and indexes, 145–147
  `C_DDL` column in a column group extension, 140–142
  `C_DDL` column in a virtual column, 139–140
  `C_DDL` default value changes, 142–144

DDL (data definition language) optimization (*continued*)
 inaccurate cardinality estimates, resolving, 139–140
 for NUll columns, 147–152
 overview, 133–136
 table cardinality estimation, 137–138
DEBUG parameter, 188
Decompression, 177
Deduplication, 8
Depth statistics, indexes, 435
DETACH program, 317
diagcollection.pl script, 278
Differential incremental backups, 177
DIRECTORY parameter, 306
Dirty blocks, instance activity statistics, 224
Disk reads, analyzing, 219
Disk sort, SSDs, 406–408
Disk spills, analyzing, 246, 264–266
Disk-based backup, 179, 189
DISPATCHERS parameter, 243
Distinct key statistics, indexes, 435
Distributed transactions. *See* XA (X/Open XA).
DRA (Data Recovery Advisor), 193–194
DTP (distributed transaction processing), hanging databases, 22–24
Dump files
 adding, 319
 resizing, 319
 scrambling sensitive data, 314
 specifying, 306
 verifying content of, 308–309
DUMPFILE parameter, 306
DUPLICATE DATABASE method, 330, 332–333
Dynamic memory components, 260–262

E
Elapsed time, analyzing, 217–218
Encryption, 8
Enqueue
 definition, 2
 instance activity statistics, 224
 services, analyzing, 209–210
 statistics, 248–250
Equality predicate histograms, bind sensitiveness with ACS, 55–56
Error messages. *See specific messages*.
ESTIMATE_PERCENT value, getting, 170
EtherChannel, 279
Exachk utility, 280
Exadata
 backup and recovery solutions, 171
 SSDs (solid-state drives), 414–418
EXCLUDE parameter
 exporting public database links and synonyms, 307–308
 finding valid values, 309–310
 saving and restoring database links, 307
 specifying objects for import/export, 306
Execution count, instance activity statistics, 225
Executions, analyzing, 219
EXIT_CLIENT command, 319
Expdp, common parameters, 306
Exp/imp tools, 305
EXPLAIN PLAN FOR command, 451
Explain plans, comparing original and new, 294
Exporting. *See also* Copying; Data Pump; Importing.
 database links and synonyms, 307
 file size, predicting, 305
 help for, 306
 from a higher version to a lower one, 322
 legacy exp/imp tools, 305
 from Oracle Database 9*i* or older, 305
 subsets of table data, 310–313
EXtended Architecture (XA), hanging databases, 22–24
Extended cursor sharing. *See* ACS (adaptive cursor sharing).
Extents. adding to LOBs, 7
EXTRACT processes, 329
Extremely large databases (XLDBs). *See* XLDBs (extremely large databases).

F
Failover, configuring, 280
Fast recovery area (FRA), 179
FILESIZE command, 319
FILESPERSET setting, 192
Filtering data, during migration, 329
FIRST_ROWS parameter, 117–122
Flash SSD latency, 389–390
Flash technology. *See* SSDs (solid-state drives).
Flashback Database features enabling/disabling, 179
Flashback options, 280
Flashback technology, rewinding databases, 178–179
FLASHBACK_SCAN parameter, 347
FLASH_CACHE clause, 399–400
Forced-plan sharing issues, 86
Foreground wait event statistics, 213–214
Forensics. *See* AWR (Automatic Workload Repository).
FRA (fast recovery area), 179
Free buffer, instance activity statistics, 225
Free buffer waits, 396–398
free buffer waits statistics, 238
Free global transaction table entry wait event, 460–462
Free lists, 393
Free space, minimum percentage, setting, 14–17
Full backups, 176
FULL parameter, 305–306
Full table scans, SSDs, 404–406

G
Garbage collection, SSDs, 393–394
GATHER_DICTIONARY_STATS parameter, 320

GC (global cache)
  analyzing, 209–210
  enqueue services, analyzing, 268–273
  instance activity statistics, 225
  load statistics, analyzing, 209
  times (immediate), analyzing, 272
  transfer (immediate), analyzing, 272
  transfer statistics, analyzing, 271–272
  transfer times, analyzing, 272
GC buffer busy acquire events, 36
GC buffer busy events, 36
GC buffer busy release events, 36
Ghost transactions, 457–462
Global cache (gc). *See* GC (global cache).
Global Cache Fusion, 275
Global CR served statistics, analyzing, 271
Global current served statistics, analyzing, 271
Global enqueue statistics, analyzing, 271
Global index hash partitioning, 441
Global partitioned indexes, 427–428
Global transactions. *See* XA (X/Open XA).
GTTs (global temporary tablespace groups)
  automatic statistics gathering, 358–359
  description, 356
  separate temporary tablespaces, 356
  UNDO activation, 357–358

H
HANGANALYZE procedure, 27–28
Hanganalyze utility, 27–28
Hanging transactions, 460–462
Hard parsing, 376, 378
Hash operations, SSDs, 406–408
Hash-partitioned indexes, 432
HCC (Hybrid Columnar Compressions), 160
Health Check script, 447
Heat Map feature, 160–162
Help
  exporting, 306
  migration methods, 351–352
  MOS (My Oracle Support) resources, 278–279
HELP parameter, 306
High-watermark (HW) enqueue events, 2, 4–7
_HIGHTHRESHOLD_UNDORETENTION parameter, 21
Hints, forcing an index, 437
Histograms, wait events, 215–216
Hot blocks, analyzing, 271
HugePages and Oracle 11*g* Automatic Memory Management (AMM) on Linux, 281
Hung databases
  example, 2–4
  gathering information about, 27–28
  hanganalyze utility, 27–28
Hung databases, caused by
  contention, 22–24
  DTP (distributed transaction processing), 22–24
  rollback segments, 24
  XA (eXtended Architecture), 22–24
HW (high-watermark) enqueue events, 2, 4–7
Hybrid Columnar Compressions (HCC), 160

I
IDA (In-Database Archiving), 326
ILM (Information Lifecycle Management), 326
Impdp, common parameters, 306
Importing. *See also* Copying; Data Pump; Exporting; Migration.
  default storage parameters, 314
  legacy exp/imp tools, 305
  from Oracle Database 9*i* or older, 305
  partitioned tables as nonpartitioned, 313
  resizing tablespaces, 314
  table partitions as individual tables, 313
IMU (in-memory undo) latch, 383
INCLUDE parameter
  exporting public database links and synonyms, 307–308
  finding valid values, 309–310
  saving and restoring database links, 307
  specifying objects for import/export, 306
Incremental backups, 176–178
Incremental statistics synopsis, 166–168
Incrementally updatable image copies (IUIC), 180–186
In-Database Archiving (IDA), 326
Index fetch by key, 226
Index partitioning, local vs. global, 159
Index scan, 226
Index scans kdiixs1, 226
Indexed reads, SSDs, 403
Indexes
  bitmap, 425–426
  bitmap join, 431
  branch pages, 422
  B-tree (balanced tree), 422–425, 432
  composite, 430
  compressed, 431
  concatenated, 430
  creating, 298
  global partitioned, 427–428
  hash partitioned, 432
  IOTs (index-organized tables), 430
  leaf pages, 422
  local partitioned, 427–428
  making invisible, 439–441, 443
  multiple on identical columns, 431–432
  nonunique, 432
  parallel operation, 320
  parallelism, 425
  partial, 429
  partition pruning, 427
  partitioned, 427–429
  range partitioned, 432
  referencing multiple rows simultaneously, 426
  reverse key, 430
  root pages, 422

Indexes (*continued*)
  skip-scan operations, 430
  unique, 432
Indexes, performance issues
  average data blocks per key, 435
  average leaf blocks per key, 435
  choosing the wrong index type, 437
  clustering factor statistics, 435
  deleting index entries in a block, 438–439
  depth statistics, 435
  distinct key statistics, 435
  ever-increasing values, 441
  forcing an index via a hint, 437
  global index hash partitioning, 441
  hiding unselective indexes, 439–441
  index overuse, 439
  index statistics, 432–435
  interinstance contention, 441
  leaf block statistics, 435
  low clustering factors, 435–436
  monotonically increasing indexes, 441
  nonselective indexes, 441
  OLTP and read-mostly workload contention, 442
  operational considerations, 436–439
  Oracle sequences and index contention, 442
  outmoding initialization parameter settings, 437–438
  in RAC databases, 441–442
  reverse key, 441
Index-organized tables (IOTs), 430
Information Lifecycle Management (ILM), 326
Initialization parameter changes, verifying, 266
`init.ora` parameter, 277
In-memory undo (IMU) latch, 383
`_in_memory_undo` parameter, 383
Input/output. *See* I/O.
`INSERT` performance, after migrating `BASICFILE` LOBs to `SECUREFILE` LOBs, 17
Instance activity statistics. *See* AWR (Automatic Workload Repository), instance activity statistics.
Instance CPU, analyzing, 207
Instance efficiencies, analyzing, 202–203
Instance recovery statistics, 239
Interconnect devices, analyzing, 273
Interconnect ping latency, analyzing, 272
Interconnect throughput by client, analyzing, 273
Interinstance contention, 441
Interobject parallelism, 168
I/O
  deferred writing of changed blocks, 396
  physical read/write statistics, 227–228
  timing for writes, 236
  writing to the flash cache, 398
I/O stress, tablespaces or datafiles, 237
IOTs (index-organized tables), 430
IUIC (incrementally updatable image copies), 180–186

J
Java pool advisory, 245–246, 247
`java_pool_size` parameter, 261

K
`_kdli_sio_fileopen` parameter, 17
`KILL_JOB` command, 319
Kks stats latch, 383

L
Large object (LOB) data type. *See* LOB (large object) data type.
`large_pool_size` parameter, 261
`LARGE_POOL_SIZE` parameter, 188, 193
Latch gets, 369
Latch hit percentage, analyzing, 203
Latch misses, 369
Latch sleeps, 369
Latch statistics
  activity, 253
  miss sources, 254–255
  no latch available, 253–254
  overview, 251–253
  parent and child latches, 255
  Pct Get Misses, 253
  Pct NoWait Misses, 253
  sleep breakdown, 253–254
  sleep summary, 255
  spin count, 254
Latch wait list, 369
Latch wait posting algorithm, 372
Latches
  cache buffer chains latches, 369
  definition, 368–370
  redo allocation latches, 369
Latches and mutexes
  hard parsing, 376, 378
  soft parsing, 375
Latches and mutexes, architecture
  cache buffer chains latches, 369
  internals, 370–371
  latch gets, 369
  latch misses, 369, 371, 373
  latch sleeps, 369, 371, 373
  latch wait list, 369
  latches, definition, 368–370
  mutexes, definition, 370
  overview, 367–368
  redo allocation latches, 369
  spin gets, 369–370
  spin locks, 369
  spinning, 370
  test and set instruction, 369
Latches and mutexes, contention
  drilling into segments and SQLs, 373–375
  fine tuning latch algorithms, 383–385
  identifying individual latches, 372–373
  intractable latch contention, 383–385
  latch wait posting algorithm, 372

most common cause, 376
overview, 371–372
spinning, 383–385
Latches and mutexes, contention scenarios
bind variable technique, 376–378
cache buffers lru chain latch, 382
CBC (cache buffer chain) latches, 379–381
IMU (in-memory undo) latch, 383
kks stats latch, 383
library cache mutex waits, 375–378
library cache pin wait, 378
process allocation latch, 382
RC (result cache) latches, 383
redo allocation latch, 382
session allocation latch, 382
shared pool latches, 378–379
simulator lru latch, 382
Lazy writes, 396
Leaf block statistics, indexes, 435
Leaf nodes, instance activity statistics, 226
Leaf pages, 422
Lewis, Jonathan, 118
Library cache activity, analyzing, 257–260
Library cache mutex waits, 375–378
library cache pin wait, 378
Library hit percentage, analyzing, 202
LIST FAILURE command, 193–194
LMS (Lock Management Server), 275
LMTTs (locally managed temporary tablespaces), 354
Load average, analyzing, 206–207
Load profile, 201–202
LOB (large object) data type. *See also* BASICFILE LOBs.
caching, 9
chunk size, specifying, 9
compression, 8
deduplication, 8
encryption, 8
enqueue, definition, 2
HW (high-watermark) enqueue events, 2
introduction, 1–2
LOBINDEX, 1–2
LOBSEGMENT, 1–2
logging, enabling, 9
minimum percentage of free space, setting, 14–17
storage parameters vs. performance, 9
LOB (large object) data type, example problems
adding extents, 7
database hung, 2–4
HW resolution, 4–7
increasing throughput, 8
LOBINDEX, 1–2
LOBSEGMENT, 1–2
Local partitioned indexes, 427–428
Locally managed temporary tablespaces (LMTTs), 354
Lock Management Server (LMS), 275
Locks. *See* Latches and mutexes.
Log file, specifying, 306
Log file stress, analyzing, 206
LOGFILE parameter, 306
Logging, enabling, 9
LOGGING option, 9
Logical corruption, 25, 28–29
Logical corruption, protecting against, 280
Logical standby database, 328

M
MAA (Maximum Availability Architecture) guidelines, 279–280
Masking sensitive data, 314
Materialized views, creating, 298
MAXOPENFILES setting, 192
MAXPIECESIZE setting, 192
Media corruption, 29–32
Memory
advisor, PGA, 243–244
corruption, 24–25, 26
dynamic memory components, 260–262
managing, 281
process. *See* Process memory.
resources, optimizing, 157–158
SGA, summary, 264
sort percentage, analyzing, 202–203
statistics, analyzing, 207–208
memory_max_target parameter, 260–261
Memory-related parameters, RMAN (Recovery Manager), 189
memory_target parameter, 260–261
METADATA_FILTER program, 317
Migration. *See also* Copying; Data Pump; Exporting; Importing.
across platforms, 333–336
ADO (Automatic Data Optimization), 326
IDA (In-Database Archiving), 326
ILM (Information Lifecycle Management), 326
legacy exp/imp tools, 305
from Oracle Database 9*i* or older, 305
overview, 324
purpose of, 324
selecting data for, 326
Migration methods
EXTRACT processes, 329
filtering data, 329
help for, 351–352
logical standby database, 328
modifying data on the fly, 329
OGG (Oracle Golden Gate), 329
Oracle Streams, 329
physical standby database, 328
REPLICATE processes, 329
transactional capture, 327–329
Migration methods, nontransactional migration
CPT (cross-platform transport), 331, 344–345
database cloning, 330
DUPLICATE DATABASE method, 330, 332–333
overview, 330

Migration methods, nontransactional migration (*continued*)
 physical standby database, 330, 333
 summary of methods, 330–331
 TDB (transportable database), 330, 333–336
 transferring just what's needed, 336–340
 TTS (transportable tablespaces), 331, 336–340
 verifying database transportability, 336
 XTTS (cross-platform transportable tablespaces), 331, 340–343
Migration methods, piecemeal migration
 Data Pump Export/Import utilities, 345
 Data Pump FTE (full transportable export/import), 346, 347–350
 manual methods, 345–346, 351
 partition exchange, 346, 350–351
 partition migration, 350–351
 programmed methods, 346
 resynchronizing tables, 347
 summary of methods, 345–346
Migration strategies
 read-only tolerance, 325
 real-time vs. near real-time, 325
 reversibility, 325–326
 window of inopportunity, 325
Missing transactions, 457–458
MLC (multi-level cell) SSDs, 391–392
Monitoring
 ADDM, 41, 277
 bind-awareness, 61–73
 databases in real-time, 278
 DBFC, 398–399
 distributed transactions, 462–464
 processes, 278
 third-party monitoring tools and utilities, 281
Moore, Gordon, 388
Moore's law, 388
MOS (My Oracle Support) resources, 278–279
Mutexes. *See* Latches and mutexes.
`_mutex_spin_count` variable, 384
`_mutex_wait_scheme` variable, 384
`_mutex_wait_time` variable, 384

N
NAND flash, 389
Native caches, SSDs, 405–406
NCLOB (national character large object), managing free space, 16
`NETWORK_LINK` parameter, 347
Nikolaev, Andrey, 371, 384
`NLS_SORT` parameter, 114–117
`NOLOGGING` clause, 425
Non-parse CPU percentage, analyzing, 203
Nonselective indexes, 441
Nontransactional migration. *See* Migration methods, nontransactional migration.
Nonunique indexes, 432
Null block compression, 176
`NULL` columns, DDL optimization, 147–152

O
Object properties, changing, 313–317
Objects, copying between databases, 304–305
OGG (Oracle Golden Gate), 329
OLTP (online transaction processing)
 compression. *See* Advanced Row Compression.
 and read-mostly workload contention, 442
 read/write workload, SSDs, 403–404
OOBs (out-of-band-sorts), 243
OOS (out-of-space) errors, 251
Open cursors, instance activity statistics, 226
`OPEN` program, 317
Optimizer. *See* Oracle Optimizer.
Optimizer statistics, gathering for VLDBs and XLDBs
 backup and recovery, 170–172
 gathering statistics concurrently, 168–169
 getting `ESTIMATE_PERCENT` value, 170
 incremental statistics synopsis, 166–168
 interobject parallelism, 168
`optimizer_capture_sql_plan_baselines` parameter, 87
`optimizer_use_sql_plan_baselines` parameter, 87
ORA-00439 message, 399
ORA-600 message, 24–25
ORA-1578 message, 24–25
ORA-1591 message, 458–460
ORA-01652 message, 356
ORA-4031 message, 378
ORA-7445 message, 24–25
ORAchk utility. *See also* Buffer busy wait events.
 description, 276
 downloading, 37–38
 installing, 37–38
 sample output, 38–40
 verifying customization, 38
Oracle Advanced Compression, 160
Oracle Clusterware Control (`CRSCTL`) utility, 283
Oracle Database 12c Reference Guide, 36
Oracle Database SecureFiles and Large Objects Developer's Guide, 1
Oracle Exadata Expert's Handbook, 414
Oracle Golden Gate (OGG), 329
Oracle Optimizer, interaction with SPM
 `ALL_ROWS` parameter, 117–122
 CBO plan does not match SQL plan baseline, 99–104
 CBO plan matches SQL plan baseline, 96–99
 `FIRST_ROWS` parameter, 117–122
 optimizer mode, selecting, 117–122
 overview, 96
 SQL plan is not reproducible, 104–108
Oracle products. *See specific products.*
Oracle Streams, 329
Oracle Tuning Guide and Concepts Manual, 199
ORATOP utility, 278
OS statistics. *See* AWR (Automatic Workload Repository), OS statistics.
OSWBB (OS Watcher Black Box), 278
Out-of-band-sorts (OOBs), 243

Out-of-space (OOS) errors, 251
Overloaded buffer cache, 239
Overprovisioning, 393

P

Pages, SSDs, 392
PARALLEL command, 319
PARALLEL parameter, 320–321
Parallelism, indexes, 425
Parallelization, 282
Parent latches, 255
Parse calls, analyzing, 219–220
Parses, instance activity statistics, 226
Partial indexes, 429
PARTIAL option, 164–165
Partition exchange during migration, 346, 350–351
Partition keys, bind sensitiveness with ACS, 56–57
Partition migration, 350–351
Partition pruning, indexes, 427
Partition tables, 298
Partitioned indexes, 427–429
Partitioning
  RAC environment, 282
  SSDs (solid-state drives), 410
  tiering data with SSD partitions, 410–414
PCIe (Peripheral Component Interconnect Express), 395
Pct Get Misses, 253
Pct NoWait Misses, 253
PCTFREE parameter, 14–17
PCTSPACE parameter, 314
Performance
  effects of storage parameters, 9
  full table scan, SSDs, 404–405
  increasing throughput, example, 8
  operating system performance metrics, capturing, 278
  RAC databases. *See* Troubleshooting RAC databases.
  SSD writes, 392–393
Performance bottlenecks
  isolating, 47–49
  solving with SSDs. *See* SSDs (solid-state drives).
Performance issues
  ACS, example, 76–81
  indexes. *See* Indexes, performance issues.
  SPM, 86
Performance statistics
  DBFC (Database Flash Cache), 400–402
  indexes, 432–435
Performance tuning. *See also* AWR (Automatic Workload Repository).
  backup and recovery, 187–188
  Data Pump, 320–321
  disk-based backup performance, 189
  large databases. *See* VLDBs (very large databases), performance tuning; XLDBs (extremely large databases), performance tuning.
  queries. *See* SQLT utility.
Peripheral Component Interconnect Express (PCIe), 395
PGA (program global area), memory resources, 157–158
PGA (program global area), statistics
  aggregate summary, 241–242
  aggregate target histogram, 242–243
  aggregate target statistics, 242
  cache hit percentages, 241–242
  memory advisor, 243–244
  OOBs (out-of-band-sorts), 243
  overview, 240–241
  rolled up usage data, 241–242
pga_aggregate_target parameter, 262–263
PGA_AGGREGATE_TARGET parameter, 240–244, 359
_pga_max_size parameter, 243, 262–263
PGA_TARGET parameter tuning, 158
Physical corruption, 25
Physical standby database, 328, 330, 333
Piecemeal migration. *See* Migration methods, piecemeal migration.
Platters, 388
PL/SQL API (SQL*Plus), with Data Pump, 317
Private cluster interconnect, checking, 277
Private objects, 306–309
Process allocation latch, 382
Process memory
  overview, 262–263
  SGA breakdown difference, 264
  SGA memory summary, 264
  summary, 264
Processes, monitoring, 278
ProcWatcher script, 278
Program global area (PGA), 157–158
Programmed migration methods, 346
Public objects, 306–309

Q

QUERY parameter, 310–313
Query response time, stabilizing. *See* SPM (SQL plan management).

R

RAC (real application clusters), definition, 275
RAC Configuration Audit Tool, 278
RAC databases
  indexes, 441–442
  RMAN (Recovery Manager), 189–191
  troubleshooting. *See* Troubleshooting RAC databases.
  tuning. *See* Troubleshooting RAC databases.
RAC databases, analyzing with AWR
  cluster interconnects, 210
  global cache and enqueue services, 209–210, 268–273
  global cache load statistics, 209
  global cache times (immediate), 272
  global cache transfer (immediate), 272
  global cache transfer statistics, 271–272

RAC databases, analyzing with AWR (*continued*)
  global cache transfer times, 272
  global CR served statistics, 271
  global current served statistics, 271
  global enqueue statistics, 271
  hot blocks, 271
  interconnect devices, 273
  interconnect ping latency, 272
  interconnect throughput by client, 273
  RAC statistics (CPU), 208
RACcheck tool. *See* ORAchk utility.
RAM, SSDs, 389
Range predicate, bind sensitiveness with ACS, 52–55
Range-partitioned indexes, 432
RC (result cache) latches, 383
Reading. *See* I/O.
Read-only tolerance, migration strategy, 325
Real application clusters (RAC), definition, 275
Real-time vs. near real-time migration strategy, 325
Recover forward forever (RFF), 180–186
Recovery. *See* Backup and recovery.
Recovery catalogs, retaining data in, 191
Recovery Manager (RMAN). *See* RMAN (Recovery Manager).
Recovery point objective (RPO), 174
Recovery time objective (RTO), 174
Recursive calls, analyzing, 218
Recursive calls, instance activity statistics, 229
Recursive CPU usage, analyzing, 218
Redo allocation latch, 382
Redo nowait percentage, analyzing, 202
Redo-related instance activity statistics, 229
REMAP_DATA parameter, 314
Renaming tables, 314
REPLICAT processes, 329
Reports, SQLT utility, 447–451
Reproducing a SQL plan baseline. *See* SQL plan baseline, reproducing.
Resizing, dump files, 319
RESMGR:CPU Quantum wait events, 281
Resource limits, analyzing, 266
Resource management, 280–281, 283, 285–287
Restore points, guaranteed, 179
Restructure SQL statement, 293
Result cache (RC) latches, 383
Retention time, specifying, 20–21
Retention values, disabling autotuning of, 21
Reverse-key indexes, 430, 441
Reversibility, migration strategy, 325–326
RFF (recover forward forever), 180–186
RMAN (Recovery Manager). *See also* Backup and recovery.
  cloning databases, 331–333
  memory-related parameters, 189
  migration methods, 330–331
  overview, 174–175
  for RAC databases, 189–191
  retaining data in a recovery catalog, 191
  validating backups, 186–187
RMAN BACKUP FOR TRANSPORT command, 344
Rollback segments, hanging databases, 24
Root pages, 422
Rotational latency, 388
RPO (recovery point objective), 174
RTO (recovery time objective), 174

S
SAMPLE parameter, 311–313
SATA (serial advanced technology attachment), 395
SATA vs. PCIe SSD, 395
Schemas, copying between databases, 304–305
SCHEMAS parameter, 305–306
Scrambling sensitive data, 314
SECTION SIZE parameter, 188
SECUREFILE LOBs, migrating from BASICFILE LOBs
  example, 12–14
  poor INSERT performance, 17
SECUREFILE LOBs vs. BASICFILE, 8–11
Seek time, 388
Segment access statistics, 255–257
Sensitive data, scrambling, 314
Sequences and index contention, 442
Serial advanced technology attachment (SATA), 395
servctl config commands, 287–288
Server Control (SRVCTL) utility, 283
Service related statistics, 216–217
Session allocation latch, 382
Session cursor, instance activity statistics, 229–230
SET_GLOBAL_PREFS procedure, 358–359
SGA (system global area)
  definition, 368
  memory allocation, 157–158
  memory summary, 264
  target advisory, 245–246
sga_target parameter, 260–261
SGA_TARGET parameter tuning, 157–158
Shared pool latches, 378–379
Shared pool memory statistics, analyzing, 203
Shared pool statistics, 244–245
SHARED_POOL_RESERVED_SIZE parameter, 379
shared_pool_size parameter, 260–261
SHARED_SERVERS parameter, 243
Short stroking, 389
Sierra, Carlos, 94, 445
Simulator lru latch, 382
Single-level cell (SLC) SSDs, 391–392
Skip-scan operations, 430
SLC (single-level cell) SSDs, 391–392
Sleep, definition, 253
Sleep breakdown, 253–254
Sleep summary, 255
Snapshot too old (STO), 251
Soft parse percentage, analyzing, 203
Soft parsing, 375
Solid-state drives (SSDs). *See* SSDs (solid-state drives).
SORT_AREA_SIZE parameter, 243

Sorts, instance activity statistics, 230
Space, minimum percentage of free space, setting, 14–17
Spin gets, 369–370
Spin locks, 369
_spin_count parameter, 254
_spin_count variable, 384
Spinning, 383–385
SPM (SQL plan management). *See also* SQL plan baseline.
  bind variables, 86
  demonstration, 83–86
  forced-plan sharing issues, 86
  getting started, 83–86
  interaction with Oracle Optimizer. *See* Oracle Optimizer, interaction with SPM.
  performance issues, 86
SPM (SQL plan management), adaptive cursor sharing
  Oracle 11*g* Release 11.2.0.3.0, 123–127
  Oracle 12*c* Release 12.1.0.1.0, 128–130
  overview, 122–123
sqcreate.sql script, 446
sqcsilent.sql script, 446–447
sqdefparams.sql script, 446–447
SQL (structured query language)
  analyzing. *See* AWR (Automatic Workload Repository), SQL sections.
  executing queries on a physical standby database, 451
  optimizing. *See* SQL Tuning Advisor.
  performance improvement. *See* SQL Access Advisor; SQL Performance Advisor.
  profiles, 293
  query response time, stabilizing. *See* SPM (SQL plan management).
  repairing. *See* SQL Repair Advisor.
  source of, identifying, 218
  uppercase vs. lowercase, 218
  workload analysis. *See* SQL Access Advisor.
SQL Access Advisor
  indexes, creating, 298
  materialized views, creating, 298
  in OEM 12*c*, 295–298
  overview, 295
  partition tables, 298
  recommending new access structures, 296
  from SQL Tuning Sets page, 296
  in SQL*Plus, 298–299
  structures runtime options, 297
  verifying access structures, 296
  workload source runtime options, 296
SQL Advisors Home, 290
SQL Performance Advisor, 301
SQL plan baseline, creating, 293. *See also* SPM (SQL plan management).
  capturing plans automatically, 87–90
  loading plans from the cursor cache, 90–92
SQL plan baseline, faking, 92–96
SQL plan baseline, reproducing
  adding trailing columns to the index, 112–113
  changing the index type, 111–112
  NLS_SORT parameter, 114–117
  optimizer mode, selecting, 117–122
  overview, 108–109
  renaming the index, 109–111
  reversing the index, 113–114
  SQL plan is not reproducible, 104–108, 117–122
SQL plan management (SPM). *See* SPM (SQL plan management).
SQL Repair Advisor, 300
SQL Tuning Advisor
  on an individual SQL page, 292
  invoking, 290
  licensing, 291
  on OEM 12*c*, 291–294
  overview, 290–291
  on a set of SQL statements, 292
  in SQL*Plus, 294–295
SQL Tuning Advisor, recommendations for
  comparing original and new explain plans, 294
  creating a SQL plan baseline, 293
  a restructure SQL statement, 293
  SQL profiles, 293
  stale statistics, 293
SQLFILE parameter, 308–309
sqlhc.sql utility, 447
SQL*Plus (PL/SQL API), with Data Pump, 317
SQLT utility
  comparing query execution times, 451
  creating reports, 447–451
  creator of, 445
  example, 452–453
  executing queries on a physical standby database, 451
  identifying bind variables, 451
  identifying the worst executing query, 452
  installing, 446–447
  overview, 445
  trace analysis, 451–452
SQLT utility, methods
  COMPARE, 451
  TRCANLZR, 451
  TRCASPLIT, 452
  TRCAXTR, 452
  XECUTE, 447, 448–451
  XPLAIN, 451
  XTRACT, 447–448
  XTRSBY, 451
  XTRSET, 452
  XTRXEC, 451
SRVCTL (Server Control) utility, 283
SSDs (solid-state drives). *See also* DBFC (Database Flash Cache).
  ADO (Automatic Data Optimization), 410
  ASM disk groups on Exadata, 416–418
  compression, 410
  and Exadata, 414–418

SSDs (solid-state drives) (*continued*)
  partitioning, 410
  redo log optimization, 409–410
  storage tiering, 410–414
  tiering data with partitions, 410–414
SSDs (solid-state drives), options
  disk sort, 406–408
  full table scan performance, 404–405
  full table scans, 405–406
  hash operations, 406–408
  indexed reads, 403
  native caches, 405–406
  OLTP read/write workload, 403–404
  redo log optimization, 409–410
SSDs (solid-state drives), vs. HDDs
  actuator arms, 388
  blocks, 392
  cells, 391–392
  economics, 390–391
  endurance, 392–393
  flash SSD latency, 389–390
  free lists, 393
  garbage collection, 393–394
  MLC (multi-level cell), 391–392
  NAND flash, 389
  overprovisioning, 393
  overview, 388–389
  pages, 392
  platters, 388
  RAM, 389
  rotational latency, 388
  SATA vs. PCIe SSD, 395
  seek time, 388
  short stroking, 389
  SLC (single-level cell), 391–392
  SSDs in Oracle databases, 395
  storage hierarchy, 391–392
  stripe magnetic disks, 389
  TLC (triple-level cell), 391–392
  transfer time, 388
  wear leveling, 393–394
  write performance, 392–393
Stale statistics, 293
Star configuration, 205
START_JOB program, 317
Statistics. *See also* AWR (Automatic Workload Repository); Optimizer statistics.
  buffer pool, 237–240
  buffer pool waits, 247–248
  buffer waits, 247–248
  DBFC performance, 400–402
  enqueue, 248–250
  GC instance activity, 225
  GC load, 209
  GC transfer, 271–272
  GTTS, automatic gathering, 358–359
  instance recovery, 239
  I/O, analyzing, 207–208
  memory, 207–208
  PGA aggregate target, 242
  physical read/write, 227–228
  service related, 216–217
  shared pool, 244–245
  shared pool memory, 203
  significantly relevant samples, 259
  stale, 293
  tablespace I/O, 235–237
  time model, 211–212
  undo segment statistics, 250–251
  VLDB and XLDB optimization, gathering, 168–169
Statistics, indexes
  clustering factor, 435
  depth, 435
  distinct key, 435
  leaf block, 435
Statistics, RAC databases
  global cache load, 209
  global cache transfer, 271–272
  global CR served, 271
  global current served, 271
  global enqueue, 271
  RAC statistics (CPU), 208
STATUS command, 319
STO (snapshot too old), 251
STOP_JOB command, 319
Storage tiering, SSDs (solid-state drives), 410–414
Stream pool size, analyzing, 264–266
Streams components, 264–266
Streams pool advisory, 245–246
streams_pool_size parameter, 261
STREAMS_POOL_SIZE parameter, 245–246
Stripe magnetic disks, 389
Structured query language (SQL). *See* SQL (structured query language).
Structures runtime options, 297
Subsets of table data, exporting, 310–313
Summed dirty queue length, instance activity statistics, 230
Synonyms, exporting, 307
System global area (SGA). *See* SGA (system global area).
System statistics. *See* AWR (Automatic Workload Repository).

T
Table fetch, instance activity statistics, 230–231
Table metadata, copying, 306
TABLE_EXISTS_ACTION=APPEND option, 320–321
Tables
  cardinality estimation, 137–138
  compression, 160. *See also* Compression.
  copying between databases, 305
  exporting subsets of, 310–313
  importing partitioned as nonpartitioned, 313
  importing partitions as individual tables, 313
  renaming, 314
  resynchronizing after migration, 347

TABLES parameter, 305–306
Tablespace point-in-time recovery (TSPITR), 179
Tablespaces
  copying, 306
  corruption, undoing. *See* Undo tablespace corruption.
  excluding from recovery, 179
  I/O statistics, 235–237
  I/O stress, 237
  moving, 413–414
  temporary. *See* Temporary tablespaces.
Tablespaces, importing/exporting
  consolidating, 315–317
  names, specifying, 306, 314
  resizing, 314
TABLESPACES parameter, 306
TDB (transportable database), 330, 333–336
TEMPFILE I/O waits, correcting. *See also* Temporary tablespaces.
  inappropriate extent sizing, 364
  inappropriate use of GTTs, 364
  undersized PGA, 359–363
Temporary tablespace groups (TTGs). *See* TTGs (temporary tablespace groups).
Temporary tablespaces. *See also* TEMPFILE I/O waits, correcting.
  features, 353–354
  global. *See* GTTs (global temporary tablespace groups).
  LMTTs (locally managed temporary tablespaces), 354, 355
  overview, 353–359
  read-only databases, 354
Test and set instruction, 369
TFA (Transparent File Analyzer), 276
Thread activity, instance activity statistics, 233
Three A's of troubleshooting, 277
Tiering data with partitions, SSDs (solid-state drives), 410–414
Time model statistics, 211–212
Timing, analyzing, 266
TLC (triple-level cell) SSDs, 391–392
Trace analysis, SQLT utility, 451–452
Trace logs, 277, 278
TRACE parameter, 188
Transaction rollback, instance activity statistics, 231
Transactional capture migration, 327–329
transactions_per_rollback_segment parameter, 248, 250–251
Transfer time, SSDs, 388
Transparent File Analyzer (TFA), 276
Transportable database (TDB), 330, 333–336
Transportable tablespaces (TTS), 331, 336–340
TRANSPORT_TABLESPACES parameter, 306
TRCANLZR method, 451
TRCASPLIT method, 452
TRCAXTR method, 452
TRIM command, 393
Triple-level cell (TLC) SSDs, 391–392
Troubleshooting. *See also* AWR (Automatic Workload Repository).
  backup and recovery, 188
  DEBUG parameter, 188
  TRACE parameter, 188
Troubleshooting RAC databases. *See also* ASH (Active Session History); AWR (Automatic Workload Repository), RAC-specific pages.
  ADDM (Automatic Database Diagnostic Monitor), 277
  ADR (Automatic Diagnostic Repository), 276–277
  alert logs, 277
  CHM (Cluster Health Monitor), 278
  monitoring processes, 278
  MOS (My Oracle Support) resources, 278–279
  with OEM 12c, 282–283
  operating system performance metrics, capturing, 278
  ORAchk health-check tool, 276
  ORATOP utility, 278
  OSWBB (OS Watcher Black Box), 278
  private cluster interconnect, 277
  ProcWatcher script, 278
  RAC Configuration Audit Tool, 278
  real-time monitoring, 278
  TFA (Transparent File Analyzer), 276
  three A's, 277
  trace logs, 277, 278
Troubleshooting RAC databases, best practices
  Active Data Guard, 279
  AMM (Automatic Memory Management), 281
  antivirus software, 281
  backup and recovery strategies. *See* Backup and recovery, backup strategies.
  configuring block-checking parameters, 279
  configuring failover, 280
  CPU management, 281
  EtherChannel, 279
  Exachk utility, 280
  Flashback options, 280
  MAA (Maximum Availability Architecture) guidelines, 279–280
  maintaining current versions, 280
  memory management, 281
  parallelization, 282
  partitioning, 282
  periodic health checks, 280
  protecting against data block corruption, 279
  protecting against logical corruption, 280
  resource management, 280–281
  third-party monitoring tools and utilities, 281
  tuning RAC parameters, 281–282
  undo retention, setting, 280
Troubleshooting RAC databases, utilities and commands
  administrative tasks, 283
  checking Clusterware componentry status, 283–284
  configuration information, displaying, 287–288

Troubleshooting RAC databases, utilities and commands (*continued*)
  crsctl check commands, 283
  crsctl get commands, 284–285
  crsctl query commands, 284
  crsctl status commands, 284–286
  CRSCTL (Oracle Clusterware Control) utility, 283
  resource management, 283, 285–287
  servctl config commands, 287–288
  SRVCTL (Server Control) utility, 283
TSPITR (tablespace point-in-time recovery), 179
TTGs (temporary tablespace groups), 158
  adding an existing LMTT, 355
  creating, 355
  description, 158, 355
  multiple, 355
TTS (transportable tablespaces), 331, 336–340
_TUNED_UNDORETENTION parameter, 21

U
UNDO activation, 357–358
Undo Advisor, 21
Undo block corruption, 31–32
Undo change vector, instance activity statistics, 231–232
Undo header corruption, 31–32
Undo retention, setting, 280
Undo segment statistics, 250–251
Undo tablespace corruption
  autotuning retention values, disabling, 21
  data dictionary object block corruption, 30–31
  detecting, 24–26
  _HIGHTHRESHOLD_UNDORETENTION parameter, 21
  logical corruption, 25, 28–29
  media corruption, 29–32
  memory corruption, 24–25, 26
  physical corruption, 25
  preventing, 24–26
  repairing, 24–26
  retention time, specifying, 20–21
  _TUNED_UNDORETENTION parameter, 21
  Undo Advisor, 21
  undo block corruption, 31–32
  undo header corruption, 31–32
  _UNDO_AUTOTUNE parameter, 21
  UNDO_RETENTION parameter, 20–21
  UNDO_TABLESPACE parameter, 19
_UNDO_AUTOTUNE parameter, 21
UNDO_RETENTION parameter, 20–21
UNDO_TABLESPACE parameter, 19
Unique indexes, 432
Unsafe bind variables, 220
Unselective indexes, hiding, 439–441
Unused block compression, 176
Upgrading databases, 321–322
User I/O wait time, instance activity statistics, 232
USER_ADVISOR_ACTIONS view, 298–299
USER_ADVISOR_SQLA_WK_STMTS view, 298–299
USER_ADVISOR_TASKS view, 294

V
V$ADVISOR_PROGRESS view, 294
VALIDATE BACKUPSET command, 187
Validating backups, 186–187
V$BACKUP_ASYNC_IO view, 188
V$BACKUP_SYNC_IO view, 188
Verifying database transportability, 336
Version control
  exporting from a higher version to a lower one, 322
  maintaining current versions, 280
Version count, 220
V$LATCH view, 372
VLDBs (very large databases)
  ADO (Automatic Data Optimization), 160–162
  Advanced Index Compression, 162
  Advanced Row Compression, 160
  basic configuration, 154–162
  bigfile tablespaces, 156–157
  creating, 154–155
  data compression, 159–160
  data partitioning, 158–159
  data warehouse templates, 154–155
  HCC (Hybrid Columnar Compressions), 160
  Heat Map feature, 160–162
  index partitioning, local vs. global, 159
  limiting the number of datafiles, 156–157
  memory resources, 157–158
  optimal data block size, 155–156
  Oracle Advanced Compression, 160
  overview, 153–154
  PGA (program global area), 157–158
  SGA (system global area), 157–158
  table compression, 160
  temporary tablespace groups, 158
VLDBs (very large databases), gathering optimizer statistics
  backup and recovery, 170–172
  gathering statistics concurrently, 168–169
  getting ESTIMATE_PERCENT value, 170
  incremental statistics synopsis, 166–168
  interobject parallelism, 168
VLDBs (very large databases), performance tuning
  common issues, 162
  indexes and data loading, example, 164–165
  maximizing resource utilization, 165–166
  parallelism, 165–166
  suboptimal application coding, example, 162–163
V$MEMORY_TARGET_ADVICE view, 157–158
V$PGA_TARGET_ADVICE view, 157–158
V$SGA_TARGET_ADVICE view, 157–158

W
Wait event histograms, 215–216
Wait events, analyzing, 203–206
Wait for other processes, 239
WAIT_FOR_JOB program, 317
Wear leveling, 393–394

Window of inopportunity, 325
Work area instance activity statistics, 232
Workload source runtime options, 296
`write complete waits`, 238–239
Writing. *See* I/O.

X

XA (eXtended Architecture), hanging databases, 22–24
XA (X/Open XA), distributed transaction issues
  common issues, 456–457
  free global transaction table entry wait event, 460–462
  ghost transactions, 457–462
  hanging transactions, 460–462
  information exists, transaction missing, 457–458
  missing transactions, 457–458
  monitoring distributed transactions, 462–464
  ORA-1591 has no corresponding information, 458–460
  repairing, 456–462
  transaction hangs after `COMMIT` or `ROLLBACK`, 460–462
`XECUTE` method, 447, 448–451
XLDBs (extremely large databases)
  ADO (Automatic Data Optimization), 160–162
  Advanced Index Compression, 162
  Advanced Row Compression, 160
  basic configuration, 154–162
  bigfile tablespaces, 156–157
  creating, 154–155
  data compression, 159–160
  data partitioning, 158–159
  data warehouse templates, 154–155
  HCC (Hybrid Columnar Compressions), 160
  Heat Map feature, 160–162
  index partitioning, local vs. global, 159
  limiting the number of datafiles, 156–157
  memory resources, 157–158
  optimal data block size, 155–156
  Oracle Advanced Compression, 160
  overview, 153–154
  PGA (program global area), 157–158
  SGA (system global area), 157–158
  table compression, 160
  temporary tablespace groups, 158
XLDBs (extremely large databases), gathering optimizer statistics
  backup and recovery, 170–172
  gathering statistics concurrently, 168–169
  getting `ESTIMATE_PERCENT` value, 170
  incremental statistics synopsis, 166–168
  interobject parallelism, 168
XLDBs (extremely large databases), performance tuning
  common issues, 162
  indexes and data loading, example, 164–165
  maximizing resource utilization, 165–166
  parallelism, 165–166
  suboptimal application coding, example, 162–163
`XPLAIN` method, 451
`XTRACT` method, 447–448
`XTRSBY` method, 451
`XTRSET` method, 452
`XTRXEC` method, 451
XTTS (cross-platform transportable tablespaces), 331, 340–343